Versione 260216

https://www.basic32.com

# Introduzione

## Introduction to Basic32 a basic interpreter for esp32

Basic32 is a powerful yet lightweight BASIC interpreter developed for the ESP32 board, designed to make ESP32 programming accessible even without knowledge of C/C++ or complex development environments. With Basic32 you can write, save, and run BASIC code in real time using any serial terminal. This approach completely eliminates the need to recompile the firmware every time the program is modified. Main features

- Direct writing of BASIC code from a serial terminal (e.g. Basic32Terminal, PuTTY, Arduino Serial Monitor, etc.)
- Saving and loading listings to internal memory (SPIFFS) or to an SD card (if present)
- Program storage in RAM with support for:

o numeric variables, strings, arrays o user-defined functions o control flow (IF, GOTO, GOSUB, FOR/NEXT)

- GPIO I/O control: digital and analog read/write, pin configuration
- Timing functions and random number generation
- Built-in commands: LIST, RUN, NEW, HELP, SAVE, LOAD
- Interactive interpreter: each line can be typed and evaluated in real time

BASIC file management via SD or SPIFFS commands. Basic32 is designed for:

- retro-programming enthusiasts
- makers who want to control the ESP32 in a simple way
- those looking for an educational and interactive environment
- those who want rapid prototyping without continuous compilation

Hardware requirements

- ESP32 board (DEV, S3, C3 models with SPIFFS support and optional SD interface)
- Serial connection to the PC
- (Optional) SD card connected to the pins defined in the code:
- MOSI → GPIO 23
- MISO → GPIO 19
- SCK → GPIO 18
- CS → GPIO 15

Using multiple SPI devices on ESP32 with Basic32

The ESP32 provides hardware SPI buses that can be shared among multiple peripherals (TFT display, RFID reader, touch screen, SD card, etc.). All SPI devices share the MOSI, MISO, and SCK pins, but each device must have its own dedicated Chip Select (CS).

To avoid conflicts on the SPI bus:

1. Assign a dedicated CS pin to each device (e.g. TFTCS=17, SDCS=13, TOUCHCS=4, RFIDCS=25).

2. Keep CS pins high (HIGH) when the device is not in use, so it does not interfere with others.

3. Set CS pins as OUTPUT and drive them HIGH before initializing the modules.

4. Initialize the devices with their respective commands (INITSD, ILI INIT, RFID INIT, etc.).

Example: ILI9341 display with integrated SD and Touch

10 PINMODE 17 OUTPUT NOPULL ' TFTCS as output

20 DWRITE 17 1 ' keeps the display inactive

30 PINMODE 4 OUTPUT NOPULL ' TOUCHCS as output

40 DWRITE 4 1 ' keeps the touch inactive

50 PINMODE 13 OUTPUT NOPULL ' SD_CS as output

60 DWRITE 13 1 ' keeps SD inactive

70 INITSD 13 23 19 18 ' initialize SD (CS=13, MOSI=23, MISO=19, SCK=18)

80 ILI INIT 17 16 5 3 ' initialize TFT (CS=17, DC=16, RST=5, rotation=3)

90 ILI LED 32 1 ' turn on backlight on GPIO32

100 ILI TEXT 10 50 2 SDFREE 0 255 255 ' print free space on SD in yellow

Note: If you add other SPI peripherals (e.g. RFID RC522), assign them a dedicated CS, set PINMODE <CS> OUTPUT and DWRITE <CS> 1 before executing the corresponding INIT. With this sequence, each device is ready to operate without interfering with others on the SPI bus. What can you do with Basic32?

- Write and test BASIC algorithms in real time
- Build interactive applications on ESP32 without compiling
- Save programs for reuse or future modification
- Control sensors and actuators with simple BASIC commands

## Installation and First Startup

This section guides you step by step through installing Basic32 on an ESP32 board using precompiled firmware. There is no need to use the Arduino IDE: the firmware can be flashed directly from the browser or via installation scripts provided by the project.

1. Requirements

- Compatible ESP32 board:
- ESP32 DEV or ESP32 S3 or ESP32 C3
- USB cable to connect the ESP32 to the PC
- PC with a WebSerial-compatible browser (Chrome, Edge, or derivatives)

or

- Environment with Python installed (for script-based installation)

2. Installation Methods

Method 1 – Direct installation from the website (recommended)

The easiest and fastest way to install Basic32 is via the official website, which allows flashing the firmware directly onto the ESP32 from the browser.

- 1. Connect the ESP32 board to the PC via USB
- 2. Open the site: https://www.basic32.com/flash.html
- 3. Click Connect

- 4. Authorize access to the serial port when prompted
- 5. Wait for the flashing process to complete. At the end, the board is ready to use with Basic32.

Method 2 – Installation via firmware and script (advanced)

Alternatively, you can install Basic32 by downloading the firmware files and using the official scripts provided in the project's GitHub repository.

- 1. Go to the firmware section of the project:
- 2. https://www.basic32.com/download-firmware.html
- 3. Download the folder corresponding to your ESP32 model
- 4. Connect the ESP32 to the PC
- 5. Run the provided installation script (Windows, Linux, or macOS)
- 6. Wait for the flashing process to complete

This method is intended for more experienced users or for offline installations.

3. First Startup

- 1. Once flashed, restart the ESP32.
- 2. Open a serial terminal at 115200 baud.
- 3. You should see the prompt: BASIC32 v1.0 READY. You can now type BASIC commands directly:

10 PRINT "HELLO BASIC32"

20 GOTO 10

RUN

- 4. Using SPIFFS: you can save and load BASIC listings with the commands: SAVE "programma.bas" LOAD "programma.bas"
- 5. Using SD card (optional): if your hardware has an SD card connected to the following pins: Signal / ESP32 GPIO — MISO 19, MOSI 23, SCK 18, CS 15

## File and Memory Management

Basic32 supports both the ESP32 internal SPIFFS memory and an optional microSD card. Both storage options can be used to save, load, and organize BASIC (.bas) files without recompiling the firmware.

1. SPIFFS Memory (internal) SPIFFS is the internal file system of the ESP32, automatically mounted at startup. It is useful when an SD card is not available.

- File names are case-insensitive.
- The .bas extension is conventional, but not mandatory.
- The available size depends on the SPIFFS partition in the firmware (typically 1MB–2MB).

2. SD Card (external, optional) If you have a microSD card connected to the ESP32 (with pins configured in the Basic32.ino file), you can use it as additional or primary storage.

- The system automatically detects the presence of the SD card.
- The SD card must be formatted in FAT32.

The SD card must be formatted in FAT32.

# ABS(x)

Categoria: Mathematical functions

## Sintassi

```
ABS(x)
```

## Descrizione

The ABS(x) function returns the absolute value of x, i.e. the unsigned number. It can be used in arithmetic expressions, assignments and logical conditions. Accepts both integers and decimals. If the number is already positive or zero, it is not changed.

## Esempi

```
10 A = -42
20 B = ABS(A)
30 PRINT "ABSOLUTE VALUE:"; B

Expected output:
ABSOLUTE VALUE: 42
```

Show the use of ABS with an integer.

```
10 PRINT "ABS(-3.14) ="; ABS(-3.14)

Expected output:
ABS(-3.14) = 3.14
```

This also works with float (floating point) numbers:

```
10 A = -7
20 IF ABS(A) = 7 THEN PRINT "IT IS EQUAL TO 7"

Expected output:
IT IS EQUAL TO 7
```

ABS can be used directly in an IF condition:

## Note

- ABS is a pure function: it does not modify variables, it only returns a numerical value.

# ACS CALIB SETOFFSET

## Sintassi

```
ACS CALIB SETOFFSET mv
```

## Descrizione

Manually set the zero offset in mV (typically ~Vcc/2). Useful if you know the offset measured with a multimeter or want to force it.

## Esempi

```
10 ACS INIT 34 20
20 ACS CALIB SETOFFSET 1650
30 ACS READ I
40 PRINT I
```

## Note

- Does not replace automatic calibration; you can combine ZERO and SETOFFSET (the last one called wins).

# ACS CALIB SHOW

## Sintassi

```
ACS CALIB SHOW
```

## Descrizione

Shows the current configuration on the serial line: pin, model, mV/A sensitivity, vref, zero in mV, averaged samples.

## Esempi

```
10 ACS INIT 34 5
20 ACS CALIB SHOW
```

## Note

- Output via Serial.printf(...) (like other diagnostic commands that print to serial).

# ACS CALIB ZERO

## Sintassi

```
ACS CALIB ZERO [samples]
```

## Descrizione

[[[P]]]Performs zero calibration (no current in the sensor!). Reads samples and sets the offset in mV. Shows the current configuration on the serial line: pin, model, sensitivity mV/A, vref, zero in mV, averaged samples.[[[/P]]]

## Esempi

```
10 ACS INIT 34 5
20 PRINT "Turn off the power and press ENTER"
30 WAIT 3000
40 ACS CALIB ZERO 256
50 ACS CALIB SHOW
```

Accurate calibration

## Note

- Make sure there is no current flowing through the cable during calibration.
- More samples → more stable offset (recommendation 128–512).

# ACS INIT (acs712)

## Sintassi

```
ACS INIT pin model [vref_mv] [avgSamples]
```

## Descrizione

Initialize the ACS712 sensor.

- pin: GPIO ADC1 recommended (32–39).
- model: 5, 20 or 30 (Ampere) → automatically set mV/A (185 / 100 / 66).
- vref_mv: optional (fallback if you don't use analogReadMilliVolts).
- avgSamples: optional; default 32.

Performs zero calibration (no current in the sensor!). Reads samples and sets the offset in mV.

Shows the current configuration on the serial line: pin, model, mV/A sensitivity, vref, zero in mV, averaged samples.

## Esempi

```
10 ACS INIT 34 5
20 ACS CALIB SHOW
```

5 A module on GPIO34

## Note

- Connect OUT of the ACS712 module to the chosen ADC pin (preferably ADC1).
- Module power supply according to specifications (often 5 V). The output is centered at Vcc/2.
- For ESP32: 11 dB attenuation (~3.3 V full-scale) already set in the code.

# ACS READ

## Sintassi

```
ACS READ var
```

## Descrizione

Reads the average DC current (in Amperes) using avgSamples samples. Save the value in var.

## Esempi

```
10 ACS INIT 34 20
20 ACS CALIB ZERO 256
30 ACS READ I
40 PRINT "I=";I;" A"
50 WAIT 200
60 GOTO 30
```

Continuous reading

## Note

- For DC or rectified/filtered AC. For pure AC, use RMS ACS.

# ACS RMS

## Sintassi

```
ACS RMS window_ms var
```

## Descrizione

Measures AC RMS current (Amperes) over a time window of window_ms. Removes the offset (zero) at each sample and calculates sqrt(average(x^2)).

## Esempi

```
10 ACS INIT 34 20
20 ACS CALIB ZERO 256
30 ACS RMS 500 IRMS
40 PRINT "Irms=";IRMS;" A"
50 WAIT 500
60 GOTO 30
```

50 Hz over 500 ms

## Note

- For 50 Hz, windows of 200–500 ms are ok. Longer = more stable.
- Make sure you have calibrated zero with no load.

# ACS SAMPLES

## Sintassi

```
ACS SAMPLES n
```

## Descrizione

Set the number of samples for DC averaged readings (ACS READ).

## Esempi

```
10 ACS INIT 34 5
20 ACS SAMPLES 64
30 ACS READ I
40 PRINT I
```

## Note

- Does not affect ACS RMS, which uses its own time window.

# ACS SENS

## Sintassi

```
ACS SENS mv_per_A
```

## Descrizione

Manually set the sensitivity (mV/A). Useful if your module is not the classic ACS712 or you want to calibrate the slope.

## Esempi

```
10 ACS INIT 34 5
20 ACS SENS 100
30 ACS READ I
40 PRINT I
```

Force 100 mV/A

## Note

- Typical ACS712 values: 185 (5 A), 100 (20 A), 66 (30 A).

# ACS VREF

## Sintassi

```
ACS VREF mv
```

## Descrizione

Set the ADC reference voltage in mV (used only as a fallback if analogReadMilliVolts is not available).

## Esempi

```
10 ACS INIT 34 20 3300
20 ACS VREF 3300
```

## Note

- On ESP32 with analogReadMilliVolts it is not needed. Leaves 3300 mV.

# ACS WATCH

## Sintassi

```
ACS WATCH limit GOTO line
ACS WATCH limit ABOVE GOTO line
ACS WATCH limit BELOW GOTO line
ACS WATCH limit GOTO line RMS window_ms
ACS WATCH limit ABOVE|BELOW GOTO line RMS window_ms
```

## Descrizione

Performs a current reading and jumps to line if the condition is true. Default: ABOVE (jump if current ≥ limit). With BELOW: skip if current ≤ limit.

```
With RMS window_ms measures AC RMS on the specified window; without RMS use DC averaging.
```

## Esempi

```
10 ACS INIT 34 20
20 ACS CALIB ZERO 256
30 PRINT "Motor loop..."
40 '... engine controls here ...
50 ACS WATCH 2.5 GOTO 200
60 WAIT 50
70 GOTO 40
200 PRINT "ALARM: overcurrent!"
210 'Turn off/stop the load here
```

Overcurrent protection (DC)

```
10 ACS INIT 34 5
20 ACS CALIB ZERO 256
30 PRINT "Minimum current monitor"
40 ACS WATCH 0.20 BELOW GOTO 100
50 WAIT 200
60 GOTO 40
100 PRINT "Low current! Check load/cable."
```

Minimum threshold (current drop)

```
10 ACS INIT 34 30
20 ACS CALIB ZERO 256
30 PRINT "AC RMS control"
40 ACS WATCH 0.80 GOTO 100 RMS 400
50 WAIT 100
60 GOTO 40
100 PRINT "Irms threshold reached!"
```

AC at 50 Hz with RMS (400 ms window)

```
10 ACS INIT 34 5
20 ACS CALIB ZERO 256
30 PRINT "Servant in motion"
40 'SERVOWRITE 1 90 ' example
50 ACS WATCH 1.2 GOTO 300
60 WAIT 50
70 GOTO 40
300 PRINT "STOP: current too high"
310 'SERVOWRITE 1 0
```

Servo stop when threshold is exceeded

### Note

- Use in loop; the command does not maintain internal state: it evaluates and, if true, skips.
- With RMS, the call is blocking for ~window_ms. Choose an appropriate window (for 50 Hz typically 200–500 ms) or stick to DC if you want faster responses.
- Make sure you have done ACS CALIB ZERO without load for a stable zero.
- You can combine multiple ACS WATCH in the loop with different thresholds (e.g. warning and shutdown).

# ADC CAL GAIN

## Sintassi

```
ADC CAL GAIN <fattore>
```

## Descrizione

Set the multiplicative gain of the ADC converter. It is used to correct scale differences between the real voltage and the measured one. The default value is 1.0.

## Esempi

```
10 ADC CAL GAIN 1.015
```

Increase the scale by 1.5 %

```
10 ADC CAL GAIN 0.98
```

Reduce scale by 2%

## Note

- GAIN must be > 0, otherwise error:
- ?ADC CAL ERROR
- GAIN must be > 0
- Typically varies between 0.95 and 1.10.
- Automatically applies to VREAD and RREAD.

# ADC CAL MEASURE

## Sintassi

```
ADC CAL MEASURE <pin> <volt_noto> [campioni]
```

## Descrizione

Performs an automatic GAIN calibration by measuring a known voltage present on the ADC pin. The firmware reads the actual (uncalibrated) voltage at the pin, compares it to the known value, and updates the GAIN so that future readings match.

## Esempi

```
ADC CAL RESET
ADC CAL REF 3.30
ADC CAL MEASURE 34 3.300 32
```

Calibration at 3,300 V on GPIO34

```
ADC CAL MEASURE 34 2.500 16
```

Calibration at 2,500 V with 16 samples

```
ADC CAL STATUS
```

Check Output: ADC REF=3.300000 GAIN=1.160772 OFFSET=0.000000

## Note

- known_volt must correspond to a real and stable voltage on the pin.
- [samples] is optional (default 16).
- Recommended values: between 1.0 V and 3.0 V.
- If the voltage is 0 or absent:
- ?ADC CAL ERROR
- Measured ~0V (check wiring/ref)
- MEASURE updates GAIN only (not REF or OFFSET).

# ADC CAL OFFSET

## Sintassi

```
ADC CAL OFFSET <volt>
```

## Descrizione

Applies an additive offset (in volts) to all ADC readings. It is used to correct zero errors or constant differences between real 0 V and ADC reading.

## Esempi

```
10 ADC CAL OFFSET -0.010
```

Subtract 10 mV

```
10 ADC CAL OFFSET 0.005
```

Add 5 mV

## Note

- OFFSET is added after the gain (V = (Raw × REF / Counts) × GAIN + OFFSET).
- Typical values: −0.020 V to +0.020 V.
- Useful for correcting constant offsets at 0V.
- No errors if set to 0.

# ADC CAL REF

## Sintassi

```
ADC CAL REF <volt>
```

## Descrizione

Set the ADC reference voltage, i.e. the maximum value corresponding to the full scale of the conversion. It is used to adapt the calculation of volts to the actual power supply of the microcontroller.

## Esempi

```
10 ADC CAL REF 3.30
```

Set 3.30V

```
10 ADC CAL REF 3.28
```

Custom reference

## Note

- REF must be > 0, otherwise error:
- ?ADC CAL ERROR
- REF must be > 0
- Set it to the real value of the supply voltage (e.g. 3.28 V).
- Combines with GAIN and OFFSET to calculate the final voltage.

# ADC CAL RESET

## Sintassi

```
ADC CAL RESET
```

## Descrizione

Restore calibration values to factory defaults:

```
REF = 3.30 V
GAIN = 1.0
OFFSET = 0.0
```

Useful for canceling any previous calibration and returning to standard parameters.

## Note

- No errors if run multiple times.
- Perform before a new calibration (ADC CAL MEASURE).
- Ensures consistent and repeatable readings.

# ADC CAL STATUS

## Sintassi

```
ADC CAL STATUS
```

## Descrizione

Shows the currently active calibration parameters: Reference (REF) Gain (GAIN) Offset (OFFSET) Allows you to check the values applied to VREAD and RREAD.

## Esempi

```
10 ADC CAL STATUS
```

Prints the current parameters on the serial monitor.

## Note

- Does not change any values.
- Use it after each calibration or change to confirm parameters.
- It is the main verification command of the ADC system.

# ALEXABRIGHT

## Sintassi

```
ALEXABRIGHT nome
```

## Descrizione

The ALEXABRIGHT command returns the brightness value (dimmer) sent by Alexa for a paired device.

The returned value is between 0 and 255, where:

- 0 → device turned off (OFF)
- 255 → maximum brightness (100%)
- intermediate values → proportional brightness

The command:

[[[P]]]DOES NOT consume the event can be used in PRINT, IF, LET it is ideal for PWM LEDs, dimmable lights and analog controls[[[/P]]]

## Esempi

```
10 WIFI "ssid" "password"
20 IF WIFICONNECTED = 0 THEN GOTO 20
30 ALEXACALL "Luce"
40 PRINT "Brightness:"; ALEXABRIGHT "Luce"

Expected output:
Prints a value from 0 to 255 based on the Alexa command.
```

View the Alexa brightness value

```
10 WIFI "ssid" "password"
20 IF WIFICONNECTED = 0 THEN GOTO 20
30 PINMODE 25 OUTPUT NOPULL
40 ALEXACALL "Luce"
60 LET B = ALEXABRIGHT "Luce"
70 AWRITE 25 B
80 GOTO 60

Expected output:
The brightness of the LED on pin 25 follows the level set by Alexa.
```

PWM LED control with Alexa

```
10 WIFI "ssid" "password"
20 IF WIFICONNECTED = 0 THEN GOTO 20
30 ALEXACALL "Luce"
40 LET B = ALEXABRIGHT "Luce"
50 LET P = INT(B * 100 / 255)
60 PRINT "Brightness:"; P; "%"
70 GOTO 40
```

View brightness in percentage

## Note

- The value is always between 0 and 255
- OFF equals 0

- The command reads the value, does not modify it
- Can be combined with ALEXASTATE

# ALEXACALL

## Sintassi

```
ALEXACALL nome GOTO linea
```

## Descrizione

The ALEXACALL command registers an Alexa device and allows you to intercept voice commands as events. Main functions: creates an Alexa device intercepts ON/OFF commands can force an immediate jump (GOTO) in the program The command is one-shot: it returns 1 only once when an event arrives after reading the event is consumed Ideal for: gate buttons locks momentary actuators

## Esempi

```
10 WIFI "ssid" "password"
20 IF WIFICONNECTED = 0 THEN GOTO 20
30 ALEXACALL "Open Gate" GOTO 100
40 PRINT "Waiting for command..."
50 DELAY 1000
60 GOTO 40
100 PRINT "COMMAND RECEIVED"
110 GOTO 40
```

Alexa button with GOTO

```
10 WIFI "ssid" "password"
20 IF WIFICONNECTED = 0 THEN GOTO 20
30 ALEXACALL "Pulsante"
40 IF ALEXACALL "Pulsante" THEN PRINT "PRESSED!"
50 DELAY 500
60 GOTO 40
```

Alexa event with IF

```
10 WIFI "ssid" "password"
20 IF WIFICONNECTED = 0 THEN GOTO 20
30 PINMODE 2 OUTPUT NOPULL
40 ALEXACALL "Impulso" GOTO 100
60 DELAY 1000
70 GOTO 60
100 DWRITE 2 1
110 DELAY 300
120 DWRITE 2 0
130 GOTO 60
```

Momentary actuator

## Note

- The event is consumed after reading
- Ideal as a button
- For continuous ON/OFF use ALEXASTATE
- Requires active Wi-Fi

# ALEXASTATE

## Sintassi

```
ALEXASTATE nome
```

## Descrizione

The ALEXASTATE command returns the current ON/OFF state of an Alexa device. Return values:

```
1 → device turned on
0 → device off
```

The command: DOES NOT consume the event can be read multiple times is ideal for driving LEDs, relays and real loads

## Esempi

```
10 WIFI "ssid" "password"
20 IF WIFICONNECTED = 0 THEN GOTO 20
30 ALEXACALL "Luce"
40 PRINT ALEXASTATE "Luce"
```

Print Alexa status

```
10 WIFI "ssid" "password"
20 IF WIFICONNECTED = 0 THEN GOTO 20
30 ALEXACALL "Luce" GOTO 60
40 DELAY 200
50 GOTO 40
60 IF ALEXASTATE "Luce" = 1 THEN PRINT "ON"
70 IF ALEXASTATE "Luce" = 0 THEN PRINT "OFF"
80 GOTO 40
```

Turn an LED on and off

```
10 WIFI "ssid" "password"
20 IF WIFICONNECTED = 0 THEN GOTO 20
30 ALEXACALL "Luce"
40 IF ALEXACALL "Luce" THEN PRINT "EVENTO"
50 PRINT "STATE:"; ALEXASTATE "Luce"
60 DELAY 500
70 GOTO 40
```

State + event combined

## Note

- OFF equals 0
- The status remains valid until Alexa changes it
- Can be used in IF, LET, PRINT
- Ideal for relays and real lights

# AND, OR, NOT

## Sintassi

```
A AND B
A OR B
NOT A
```

## Descrizione

The logical operators AND, OR, and NOT are used to perform logical or bitwise comparisons within conditional or arithmetic expressions. AND returns 1 only if both operands are different from zero OR returns 1 if at least one of the two is different from zero NOT inverts the logical value: NOT 0 is -1, NOT 1 is 0 In BASIC32, these operators can be used: In logical comparisons (IF ... THEN) In assignments (LET) In binary operations (e.g. bit masks)

## Esempi

```
10 A = 1
20 B = 2
30 IF A = 1 AND B = 2 THEN PRINT "BOTH TRUE"

Expected output:
BOTH TRUE
```

Use with IF and AND

```
10 A = 0
20 B = 5
30 IF A <> 0 OR B <> 0 THEN PRINT "AT LEAST ONE IS DIFFERENT FROM ZERO"

Expected output:
AT LEAST ONE IS DIFFERENT FROM ZERO
```

Use with OR

```
10 A = 0
20 IF NOT A THEN PRINT "A IS ZERO"

Expected output:
A IS ZERO
```

Use of NOT

```
10 X = 7         'binary: 0111
20 MASK = 4      'binary: 0100
30 RESULT = X AND MASK
40 PRINT "RESULT:"; RESULT

Expected output:
RESULT: 4
```

Bitmask with AND

```
10 A = 5
20 B = (A > 0) AND (A < 10)
30 PRINT B

Expected output:
1
```

Use in logical assignment

### Note

- AND, OR, NOT return numeric values (0 or 1/-1)
- Values other than zero are treated as TRUE.
- Values equal to zero are FALSE (FALSE)

# AREAD(p)

## Sintassi

```
AREAD(p)
```

## Descrizione

The AREAD(p) function reads the analog value from the ESP32 pin p. Returns a value between 0 and 4095, where:

```
0 corresponds to 0V
4095 corresponds to approximately 3.3V
```

It is used to read analog sensors (e.g. potentiometers, light sensors, temperature, etc.) connected to one of the analog pins of the ESP32. Common pins for analog readout include: GPIO 36, 39, 34, 35, 32, 33. Normal digital pins do not support analog readout. Make sure you are using the correct ADC pins.

## Esempi

```
10 PRINT "ANALOG READING:"
20 V = AREAD(36)
30 PRINT "VALUE:"; V
40 WAIT 1000
50 GOTO 20

Expected output:
(variable values ■■from 0 to 4095, depends on the position of the potentiometer)
ANALOG READING:
VALUE: 512
```

Reads a value from the GPIO36 pin every second:

```
10 LUX = AREAD(36)
20 IF LUX < 1000 THEN PRINT "LIGHT OK" ELSE PRINT "LOW LIGHT"
30 WAIT 1000
40 GOTO 10

Expected output:
LIGHT OK
LOW LIGHT
```

Check brightness threshold

# ASC(A$)

## Sintassi

```
ASC(stringa$)
```

## Descrizione

The ASC function returns the ASCII code of the first character of the string A$. It is useful for: Identifying the numerical value of a character Creating comparisons between letters Handling keyboard input character by character (e.g. with GET) If the string is empty (""), the result is 0 or may generate an error (depending on the implementation).

## Esempi

```
10 A$ = "A"
20 PRINT ASC(A$)

Expected output:
65
```

ASCII code of a letter

```
10 C$ = "Z"
20 IF ASC(C$) = 90 THEN PRINT "It's Z"

Expected output:
It's Z
```

Comparison with a specific letter

```
10 T$ = "C"
20 COD = ASC(T$)
30 PRINT CHR$(COD)

Expected output:
C
```

From character to code and back

```
10 S$ = "ABC"
20 FOR I = 1 TO LEN(S$)
30 PRINT MID$(S$, I, 1); " = "; ASC(MID$(S$, I, 1))
40 NEXT I

Expected output:
A = 65
B = 66
C = 67
```

Multiple reading from string

## Note

- Only the first character of the string is considered
- If the string is empty (""), it may return 0 or generate an error
- Use together with CHR$, LEFT$, GET, MID$ for complex manipulations

# AUTORUN

## Sintassi

```
AUTORUN "file.bas"
AUTORUN "file.bas" PIN <numero>
AUTORUN OFF
```

## Descrizione

The AUTORUN command sets a BASIC program to run automatically when the system starts. The name of the file to be executed must be enclosed in quotation marks and must have a .bas extension. It is possible to specify a security PIN (PIN ), i.e. a GPIO that can be read at startup to decide whether or not to autorun (optional implementation on the firmware side). If AUTORUN OFF is used, autorun is disabled and the configuration file is removed. If no PIN is specified, PIN 0 is automatically used.

## Esempi

```
AUTORUN "startup.bas"

Avvierà automaticamente startup.bas all'power on, with PIN 0 as default.
```

Enable autorun on a file

```
AUTORUN "demo.bas" PIN 5

Salva la configurazione per eseguire demo.bas all'boot, with control over GPIO5.
```

Enable autorun with a specific PIN

```
AUTORUN OFF

Disattiva completamente l’avvio automatico
```

Disable autorun

## Note

- The specified file must exist on SPIFFS and have a .bas extension
- If the file is not found, an error is returned
- The safety GPIO is optional and can be used to block automatic execution depending on the logic value of the GPIO
- The /autorun.cfg configuration file is overwritten every time you set a new autorun
- Program execution occurs immediately after configuration

# AWRITE(p, v)

## Sintassi

```
AWRITE(pin valore)
```

## Descrizione

The AWRITE command sets a PWM (pulse width modulation) signal on a pin of the ESP32, simulating an analog value. It is used to: Adjust the brightness of an LED Control the speed of a motor Manage analog devices via PWM The value must be between 0 and 255, where:

```
0 → signal completely off (0% duty cycle)
255 → maximum signal (100% duty cycle)
intermediate values → proportional (e.g. 128 = 50%)
```

The pin must first be configured as OUTPUT using PINMODE.

## Esempi

```
10 PINMODE 13 OUTPUT NOPULL
20 AWRITE 13 128

Expected output:
The LED connected to GPIO13 turns on at half intensity.
```

50% LED brightness

```
10 PINMODE 2 OUTPUT NOPULL
20 FOR A = 0 TO 255 STEP 5
30 AWRITE 2 A
40 DELAY 50
50 NEXT A
60 FOR I = 255 TO 0 STEP -5
70 AWRITE 2 I
80 DELAY 50
90 NEXT I

Expected output:
The LED connected to GPIO12 gradually increases and then decreases its brightness.
```

Continuous fading (fading effect)

```
10 PINMODE 14 OUTPUT NOPULL
20 INPUT L
30 AWRITE 14 L

Expected output:
The value of the variable L (e.g. from a sensor) adjusts the brightness of the LED on pin 14
```

Variable-based analog control

## Note

- AWRITE requires ESP32 with Arduino core 3.x or higher, where analogWrite() is available.
- If the pin does not support PWM, the command may have no visible effect.
- Values over 255 are automatically limited to 255.

# BREAKPIN

## Sintassi

```
BREAKPIN pin
BREAKPIN pin PULLUP|PULLDOWN|NOPULL
BREAKPIN OFF
BREAKPIN ?
```

## Descrizione

Set a “safety pin” that, when pressed, immediately stops the running BASIC program.

- pin (required): GPIO to use as a stop button.
- Input mode:
- PULLUP (default): the pin uses internal pull-up; the button goes to GND; pressed = LOW.
- PULLDOWN: use internal pull-down; the button goes to VCC; pressed = HIGH.
- NOPULL: no internal resistance (use external hardware); pressed = HIGH.
- OFF: disables the function.

The configuration is saved in EEPROM and reloaded at startup.

## Esempi

```
BREAKPIN 0
```

Stop with pull-up (pushbutton towards GND)

```
BREAKPIN 12 PULLDOWN
```

Stop with pull-down (button to VCC)

```
BREAKPIN 25 NOPULL
```

Without internal pulls (external hardware)

```
BREAKPIN OFF
```

Disable BREAKPIN

## Note

- • The effect is immediate: when pressed, the program stops without waiting for the end of the current line (the check occurs at the beginning of each executed line).
- Avoid critical boot pins (e.g. 0, 2, 15) if your hardware does not handle them correctly at startup.
- With PULLUP the button must close to ground; with PULLDOWN to VCC.
- If the pin is not configured (or is OFF), no shutdown occurs.
- The function is independent of the autorun SAFE MODE: SAFE MODE blocks automatic startup, BREAKPIN stops execution while the program runs.

# BT AT

## Sintassi

```
BT AT "cmd" RESP var$ [timeout]
```

## Descrizione

Sends an AT command (automatically adds CRLF) and reads a response line within millisecond timeouts. If timeout is omitted, use the value set with BT TIMEOUT. The response is written in var$.

## Esempi

```
10 BT ATMODE ON
20 BT AT "AT" RESP R$ 1000
30 PRINT R$

Expected output:
OK
```

AT test

```
10 BT ATMODE ON
20 BT AT "AT+NAME?" RESP N$ 1000
30 PRINT N$

Expected output:
+NAME:BASIC32
```

Read the module name

```
10 BT INIT 15 12 9600 33
20 BT ATMODE OFF
30 BT AT "AT" RESP R$ 500
40 PRINT "DATA9600=[";R$;"]"
50 BT END
60 BT INIT 15 12 38400 33
70 BT ATMODE OFF
80 BT AT "AT" RESP R$ 500
90 PRINT "DATA38400=[";R$;"]"

Expected output:
DATA9600=[]
DATA38400=[OK]
```

AT baud diagnostics

## Note

- Requires AT mode active (BT ATMODE ON) or KEY high at startup (HC-05).
- On many HC-05s the true AT is at 38400.

# BT ATMODE OFF

## Sintassi

```
BT ATMODE OFF
```

## Descrizione

Exits AT mode: sets KEY low (if configured) and reopens the UART at the saved data baud.

## Esempi

```
10 BT ATMODE OFF
20 PRINT "DATA MODE"

Expected output:
DATA MODE
```

## Note

- After changing baud to AT, then realign the data port with BT INIT

# BT ATMODE ON

## Sintassi

```
BT ATMODE ON [keyPin]
```

## Descrizione

Enter AT mode: set KEY high (use keyPin if passed, otherwise BT INIT) and reopen the UART at AT baud (typically 38400 on HC-05).

## Esempi

```
10 BT INIT 15 12 9600 33
20 BT ATMODE ON
30 BT AT "AT" RESP R$ 1000
40 PRINT R$

Expected output:
OK
```

## Note

- On many HC-05 the real AT requires KEY high when turning on
- Some HC-06s remain in AT at 9600

# BT AVAILABLE

## Sintassi

```
BT AVAILABLE var
```

## Descrizione

Writes in var how many bytes are available in the RX buffer of the BT UART.

## Esempi

```
10 BT INIT 15 12 9600 33
20 BT AVAILABLE N
30 PRINT N

Expected output:
0
```

## Note

- Returns an integer ≥ 0, usable in PRINT/LET/IF

# BT END

## Sintassi

```
BT END
```

## Descrizione

Closes UART2 and disables BT communication.

## Esempi

```
10 BT END
20 PRINT "CHIUSO"

Expected output:
CHIUSO
```

## Note

- Use it before changing pin/baud to a new BT INIT

# BT FLUSH

## Sintassi

```
BT FLUSH
```

## Descrizione

Empties the RX buffer of the Bluetooth UART.

## Esempi

```
10 BT INIT 15 12 9600 33
20 BT FLUSH
30 BT READ S$
40 PRINT LEN(S$)

Expected output:
0
```

## Note

- Useful for starting "clean" before a new parsing

# BT INIT (HC-05/HC-06)

## Sintassi

```
BT INIT rx tx baud [keyPin]
```

## Descrizione

Initialize the UART2 to the Bluetooth module in data mode at the baud rate. Optionally store keyPin to use BT ATMODE ON/OFF.

## Esempi

```
10 PINMODE 32 OUTPUT NOPULL
20 DWRITE 32 0
30 BT INIT 15 12 9600 33
40 PRINT "READY: SEND ON/OFF"

Expected output:
READY: SEND ON/OFF
```

Init + Ready LED

```
10 BT INIT 15 12 9600
20 PRINT "BT OK"

Expected output:
BT OK
```

Init without KEY

## Note

- Syntax: BT INIT rx tx ... → with your pins it is BT INIT 15 12 ...
- Use before BT SEND/READ/...

# BT PRINT

## Sintassi

```
BT PRINT valore
```

## Descrizione

Converts value to text and sends it over BT (no automatic newlines).

## Esempi

```
10 BT INIT 15 12 9600 33
20 LET ST=1
30 BT SEND "LED="
40 BT PRINT ST
50 BT SEND "\r\n"

Expected output:
(“42” arrives on BT)
```

## Note

- To wrap add BT SEND "\r\n"

# BT READ

## Sintassi

```
BT READ var$ [n]
```

## Descrizione

Reads up to n bytes from the RX buffer. If n is omitted, read everything available.

## Esempi

```
10 PINMODE 32 OUTPUT NOPULL
20 DWRITE 32 0
30 BT INIT 15 12 9600 33
40 BT AVAILABLE N
50 IF N=0 THEN GOTO 40
60 BT READ C$ 1
70 IF C$="1" THEN DWRITE 32 1
80 IF C$="0" THEN DWRITE 32 0
90 GOTO 40
```

Single control for LED

```
10 BT INIT 15 12 9600 33
20 BT AVAILABLE N
30 IF N=0 THEN GOTO 20
40 BT READ S$
50 PRINT "RX=[";S$;"]"
```

Debugging

## Note

- Don't wait for newlines.

# BT READLN

## Sintassi

```
BT READLN var$
```

## Descrizione

Reads up to LF (\n), ignoring \r. If \n has not yet arrived, it returns what is present at that moment.

## Esempi

```
10 PINMODE 32 OUTPUT NOPULL
20 DWRITE 32 0
30 BT INIT 15 12 9600 33
40 PRINT "BT LED READY (ON/OFF)"
50 LET CMD$=""
60 BT AVAILABLE N
70 IF N=0 THEN GOTO 60
80 BT READ T$
90 IF LEN(T$)=0 THEN GOTO 60
100 IF T$=CHR$(13) THEN GOTO 200
110 IF T$=CHR$(10) THEN GOTO 200
120 LET CMD$=CMD$+T$
130 IF LEN(CMD$)>10 THEN LET CMD$=""
140 GOTO 60
200 IF CMD$="ON" THEN GOTO 300
210 IF CMD$="OFF" THEN GOTO 400
220 BT SEND "CMD ERR\r\n"
230 GOTO 500
300 DWRITE 32 1
310 BT SEND "LED ON\r\n"
320 GOTO 500
400 DWRITE 32 0
410 BT SEND "LED OFF\r\n"
420 GOTO 500
500 LET CMD$=""
510 GOTO 60
```

ON / OFF LED (recommended pattern)

## Note

- In your BASIC it is more reliable to accumulate characters and stop on CR/LF.

# BT SEND

## Sintassi

```
BT SEND "testo"
BT SEND var$
```

## Descrizione

Send the string (literal or var$) as it is on the Bluetooth UART.

## Esempi

```
10 BT INIT 15 12 9600 33
20 BT SEND "LED ON\r\n"

Expected output:
(no console output; string goes to Bluetooth)
Note:
```

## Note

- If the peer requires CRLF, include \r\n in the text

# BT SETBAUD

## Sintassi

```
BT SETBAUD rate
```

## Descrizione

```
Set the module data baud (e.g. 9600). Try syntax HC-05 (AT+UART=rate,0,0). Update the baud u
```

## Esempi

```
10 BT INIT 15 12 9600 33
20 BT ATMODE ON
30 BT SETBAUD 19200
40 BT ATMODE OFF
50 BT END
60 BT INIT 15 12 19200 33
70 PRINT "BAUD OK"

Expected output:
BAUD OK
```

## Note

- On some HC-05 AT+RESET is needed to apply the new UART

# BT SETNAME

## Sintassi

```
BT SETNAME "Nome"
```

## Descrizione

```
Helper: Try HC-05 (AT+NAME=Name) and HC-06 (AT+NAMEName). He doesn't read the answer.
```

Change the name of the bluetooth module.

## Esempi

```
10 BT SETNAME "BASIC32"
20 BT AT "AT+NAME?" RESP N$ 1000
30 PRINT N$

Expected output:
+NAME:BASIC32
```

## Note

- Check with BT AT … RESP … if you want echo/OK

# BT SETPIN

## Sintassi

```
BT SETPIN "1234"
```

## Descrizione

```
Helper: Try HC-05 (AT+PSWD=1234) and HC-06 (AT+PIN1234). He doesn't read the answer.
```

Change the bluetooth module password.

## Esempi

```
10 BT SETPIN "1234"
20 BT AT "AT+PSWD?" RESP P$ 1000
30 PRINT P$

Expected output:
+PSWD:1234
```

## Note

- PIN reading syntax may vary with firmware

# BT TIMEOUT

## Sintassi

```
BT TIMEOUT ms
```

## Descrizione

Set the default timeout (ms) for AT readings (BT AT … RESP …, BT VERSION).

## Esempi

```
10 BT TIMEOUT 1500
20 PRINT "TIMEOUT OK"

Expected output:
TIMEOUT OK
```

## Note

- Affects line-response AT readings only

# BT VERSION

## Sintassi

```
BT VERSION var$
```

## Descrizione

Try AT+VERSION? (HC-05) and, if empty, AT+VERSION (HC-06). Writes the response in var$.

## Esempi

```
10 BT ATMODE ON
20 BT VERSION V$
21 IF LEN(V$)=0 THEN PRINT "NO RESPONSE"
30 PRINT V$

Expected output:
+VERSION:2.0-20100601
```

## Note

- Requires AT mode active

# CALLFUNC

## Sintassi

```
CALLFUNC <nome>
CALLFUNC <nome>(arg1, arg2, arg3, ...)
```

## Descrizione

Execute a function defined with FUNC, only once. Execution is blocking: the program waits for the function to finish before continuing. You can pass numeric and string arguments via parentheses. Important: CALLFUNC does not work with LOOP functions (in that case use STARTFUNC). _____ Arguments: how to read them inside the function Inside a FUNC called with arguments, you can read the parameters with:

```
ARGC() → number of arguments passed
```

ARG(n) → numeric argument n (from 1 to N). If it does not exist or is not numeric → returns 0 ARG$(n) → argument as string n. If it does not exist → return ""

## Esempi

```
5  PINMODE 2 OUTPUT NOPULL
10 FUNC LAMP
20 DWRITE 2 1
30 DELAY 500
40 DWRITE 2 0
50 DELAY 500
60 ENDFUNC
70 CALLFUNC LAMP

Expected output:
The LED turns on and off once with a 500 ms wait.
```

Call an LED function

```
10 PINMODE 2 OUTPUT NOPULL
20 FUNC LAMP
30 DWRITE 2 1
40 DELAY 200
50 DWRITE 2 0
60 DELAY 200
70 ENDFUNC
80 CALLFUNC LAMP
90 DELAY 1000
100 GOTO 80

Expected output:
The LED flashes every second (a single “flash” is made by the function).
```

Repeated use of the function

```
10 FUNC SALUTA
20 PRINT "ARGC="; ARGC()
30 PRINT "Text:"; ARG$(1)
40 PRINT "Number:"; ARG(2)
50 ENDFUNC
60 CALLFUNC SALUTA("Ciao", 123)

Expected output:
ARGC=2
```

```
Text:Hello
Number:123
```

CALLFUNC with arguments (“command” style)

```
10 FUNC SOMMASTAMPA
20 PRINT "Result:"; ARG(1) + ARG(2)
30 ENDFUNC
40 A = 10
50 CALLFUNC SOMMASTAMPA(A+5, 2+3)

Expected output:
Result:20
```

Arguments as expressions

## Note

- The function must already be defined with FUNC
- The name must match exactly
- It can be called multiple times in the program
- It is blocking: the program waits for it to finish
- Does not work with LOOP functions (in that case use STARTFUNC)
- Arguments can be numbers, strings, or expressions
- Inside the function, the parameters are read with ARGC(), ARG(n), ARG$(n)

# CARDKB

## Sintassi

```
CARDKB
```

## Descrizione

CARDKB returns the ASCII code of the last key pressed on the CardKB keyboard connected via I²C. If no key was pressed → returns 0. If a key is available, the function returns the ASCII code (0–255). Reading is non-blocking (does not stop the program). This feature is intended for: game controls interactive menus

```
input without using the INPUT
```

command

fast and continuous reading from compact I2C keyboards The keyboard must be correctly connected to the I2C bus via the command: WIRE

## Esempi

```
10 PRINT "Press a key on the CardKB"
20 K = CARDKB
30 IF K > 0 THEN PRINT "Code:", K, "Char:", CHR$(K)
40 GOTO 20

output:
reads the keyboard continuously
```

Returned value

```
10 PRINT "Press keys, ESC to exit"
20 K = CARDKB
30 IF K = 27 THEN END
40 IF K > 0 THEN PRINT "You pressed:", CHR$(K)
50 GOTO 20
```

exit with ESC

## Note

- CARDKB reads only one byte per call. If you hold down a key, the keyboard sends repeats (autorepeat).
- It is compatible with both CardKB, PS/2, and I2C emulated keyboards (works on all because it passes through devPs2HandleChar).
- ESC (ASCII 27) also activates program STOP if ESC-STOP mode is enabled.

# CHAIN

## Sintassi

```
CHAIN "file"
CHAIN SD "file"
CHAIN SPIFFS "file"
CHAIN <expr$>
CHAIN SD <expr$>
CHAIN SPIFFS <expr$>
```

## Descrizione

The CHAIN command allows you to load and start another BASIC listing from within a running listing, in “multiload” style (similar to the behavior of old home computers). When executed: the current program ends the new file is loaded in place of the current program if CHAIN is called from a listing, the new program is executed immediately (automatic RUN) the variables are NOT deleted (they remain available in the new listing) The file can be loaded from either SPIFFS or SD, and the name can be: a literal string ("P2") a string variable (F$) a string expression ("PART"+X$)

## Esempi

```
P1.bas
10 A=5
20 PRINT "I AM P1 - A=";A
30 CHAIN "P2.bas"

P2.bas
10 PRINT "I AM P2 - A=";A
20 END

Expected output:
I AM P1 - A=5
I AM P2 - A=5
```

Moving to another listing (same default device)

```
10 PRINT "LOAD FROM SD..."
20 CHAIN SD "GIOCO2"

Expected output:
GIOCO2.bas is loaded from the SD and starts immediately.
```

Load from SD

```
10 P$="PARTE"
20 N$="2"
30 P2$=”.bas”
40 CHAIN P$+N$+P2$

Load and launch:
PART2.bas
```

Filename in variable/expression

## Note

- CHAIN replaces the program in memory with the new one (does not "merge").
- It is intended for dividing long programs into multiple files and switching between them.
- If the file does not exist or is not accessible, a runtime error is generated.

• If you do not specify SD or SPIFFS, the system default device is used.

• The variables remain in memory: useful for passing parameters between one listing and another (e.g. LIV=3, SCORE=1200, etc.).

• The loaded file should contain numbered BASIC lines (standard listing).

# CHR$(x)

## Sintassi

```
CHR$(codice)
```

## Descrizione

The CHR$ function returns the ASCII character corresponding to the numeric value x (between 0 and 255). It is often used to construct dynamic strings or print special characters (e.g. ENTER, TAB, accented letters, etc.).

## Esempi

```
10 PRINT CHR$(65)

Expected output:
A
```

From number to character

```
10 FOR I = 65 TO 90
20 PRINT CHR$(I);
30 NEXT I

Expected output:
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Print letters from A to Z

```
10 PRINT "RIGA1" + CHR$(10) + "RIGA2"

Expected output:
RIGA1
RIGA2
```

Use CHR$(10) to wrap

```
10 PRINT "NOME" + CHR$(9) + "VALORE"

Expected output:
NAME VALUE
```

Insert a TAB between two words

```
10 T$ = CHR$(72) + CHR$(73)
20 PRINT T$

Expected output:
HI
```

Construct strings from code

## Note

- CHR$(10) = newline (LF), CHR$(13) = carriage return (CR)
- CHR$(32) = space, CHR$(9) = tab
- Works well together with ASC, LEFT$, RIGHT$, MID$

# CLS

## Sintassi

```
CLS
```

## Descrizione

The CLS command clears the serial terminal screen by sending a certain number of blank lines. It's a simple method to "simulate" cleaning the screen, as was done in old BASIC environments. It does not delete variables or code. It's just a visual command for the user.

## Esempi

```
10 CLS
20 PRINT "WELCOME TO THE SYSTEM"

Expected output:
(Blank screen)
```

Screen cleaning with next message

```
10 PRINT "BEFORE CONCRETE"
20 WAIT 2000
30 CLS
40 PRINT "AFTER THE CONCRETE"

Expected output:
It displays one message first, then “hides” it and shows the second.
```

CLS between two prints

# CLSANSI

## Sintassi

```
CLSANSI
```

## Descrizione

The CLSANSI command clears the terminal screen using the ANSI control code ESC[2J, which is interpreted by modern ANSI-compliant terminals (such as PuTTY, TeraTerm, minicom, etc.). It's faster and more elegant than CLS, but it only works if the terminal supports ANSI escape codes.

## Esempi

```
10 CLSANSI
20 PRINT "READY FOR INPUT"

Expected output:
(The terminal is cleaned instantly)
```

Cleaning with ANSI sequence

```
10 PRINT "CONCRETE USE:"
20 CLS
30 PRINT "FATTO"
40 WAIT 2000
50 PRINT "USE CLSANSI:"
60 CLSANSI
70 PRINT "FINITO"

Expected output:
It depends on the terminal; CLSANSI is more "professional", while CLS is more compatible.
```

Comparison between CLS and CLSANSI

# COPY

## Sintassi

```
COPY "nomefile"
COPY variabile$
COPY "*"
COPY SD "nomefile"
COPY SD variabile$
COPY SD "*"
COPY SPIFFS "nomefile"
COPY SPIFFS variabile$
COPY SPIFFS "*"
```

## Descrizione

The COPY command copies files between the SPIFFS internal storage and the SD card. Without specifying the memory, copy from the SD to the SPIFFS (SD → SPIFFS). With SD, copy from SPIFFS to SD (SPIFFS → SD). With SPIFFS, force copy from SD to SPIFFS (explicit). Accepts: file names in quotes ("setup.bas") string variables or expressions (FILE$, LEFT$(N$,3)+".bas") the asterisk "*" to copy all files from source to destination. If the file already exists on the destination, it is overwritten.

## Esempi

```
COPY SD "config.bas"

Output atteso:
Copied SPIFFS -> SD: /config.bas
```

Copy from SPIFFS to SD (default SD)

```
COPY "main.bas"

Output atteso:
Copied SD -> SPIFFS: /main.bas
```

Copy from SD to SPIFFS

```
COPY "*"

Output atteso:
Copied SD -> SPIFFS: /file1.bas
Copied SD -> SPIFFS: /file2.bas
...
```

Copy all files

```
10 NAME$ = "init.bas"
20 COPY SD NAME$
30 PRINT "File copied to SD"
```

Use a string variable

## Note

- If the SD is not mounted or the file is not present, a “Skip” message is shown.
- The operation is not recursive: it copies only the files in the root directory.
- You can copy a single file or all files with "*".
- Paths are automatically normalized (/ is added if missing).

# COS(x)

## Sintassi

```
COS(x)
```

## Descrizione

The COS(x) function returns the cosine of the angle x, where x is expressed in degrees (not radians). The returned value is a number between -1 and 1, as expected by the cosine function. Can be used in mathematical calculations, graphs or conditions. If you want to use radians, you need to manually convert: COS(x * 180 / PI)

## Esempi

```
10 PRINT "COS(60) ="; COS(60)

Expected output:
COS(60) = 0.5
```

Cosine of 60 degrees

```
10 FOR A = 0 TO 360 STEP 30
20 PRINT "COS("; A; ") = "; COS(A)
30 NEXT A

Expected output:
COS(0) = 1
COS(30) = 0.866
...
```

Cycle cosine calculation

```
10 A = 135
20 IF COS(A) < 0 THEN PRINT "NEGATIVE COSINE"

Expected output: NEGATIVE COSINE
```

Use in a conditional expression

## Note

- The argument x is expressed in degrees
- The result is always between -1 and +1
- COS(90) returns a value very close to zero (floating point error)

# DATA

## Sintassi

```
DATA valore1, valore2, valore3, ...
```

## Descrizione

The DATA command is used to declare a sequence of constant values (numeric or strings) that can be read later with the READ command. DATA is not executed directly during the program, but serves as an internal archive. Reading occurs in sequential order and can be restarted with the RESTORE command.

## Esempi

```
10 DATA 100, 200, 300
20 READ A, B, C
30 PRINT A, B, C

Expected output:
100 200 300
```

Reading numbers from DATA

```
10 DATA "UNO", "DUE", "TRE"
20 READ A$, B$, C$
30 PRINT A$, B$, C$

Expected output:
ONE TWO THREE
```

Reading strings from DATA

```
10 DATA 1, 2, 3, 4, 5
20 FOR I = 1 TO 5
30 READ X
40 PRINT "VALORE "; I; ": "; X
50 NEXT I

Expected output:
VALUE 1: 1
VALUE 2: 2
VALUE 3: 3
VALUE 4: 4
VALUE 5: 5
```

Progressive loop reading

```
10 DATA 10, 20
20 READ A, B
30 PRINT A, B
40 RESTORE
50 READ C
60 PRINT C

Expected output:
10 20
10
```

Use combined with RESTORE

# DATED

## Sintassi

```
DATED
```

## Descrizione

The DATED command returns the current day (numeric value from 1 to 31), according to the system's internal clock. This is useful for conditional operations or date checks.

## Esempi

```
10 PRINT DATED

Expected output:
15
```

Print the current day

## Note

- Returns an integer
- The value depends on the set or synchronized date
- Does not require parameters

# DATEM

## Sintassi

```
DATEM
```

## Descrizione

The DATEM command returns the current month (numeric value from 1 to 12), according to the internal clock. Useful for operations that depend on the time of year.

## Esempi

```
10 PRINT DATEM

Expected output:
6
```

Print the current month

## Note

- Returns an integer
- The value depends on the set or synchronized date
- Does not require parameters

# DATEY

## Sintassi

```
DATEY
```

## Descrizione

The DATEY command returns the current year (e.g. 2025), according to the internal clock. Allows you to obtain the year for controls, time logics or automatic labels.

## Esempi

```
10 PRINT DATEY

Expected output:
2025
```

Print the current year

## Note

- Returns a four-digit integer
- The value depends on the set or synchronized date
- Does not require parameters

# DEBUGMEM

## Sintassi

```
DEBUGMEM
```

## Descrizione

The DEBUGMEM command prints detailed information about the state of memory and active resources in the system to the serial monitor. It is useful for debugging, analyzing memory consumption, checking leaks, remaining stack, active variables and sprites still in use. Data displayed: Amount of available heap (free RAM) Residual stack of the current task (if on ESP32) Number of numerical variables and their value Number of string variables and their contents Number of defined arrays Active sprites with: position ID (X, Y) associated data name (e.g. HEART)

## Esempi

```
10 DEBUGMEM

Expected output in the serial monitor:
==== DEBUG MEMORY ====
Available heap: 128400 bytes
Current task residual stack: 3792 bytes
Numeric variables: 0
String variables: 0
Arrays defined: 0
Active sprites: 0
=======================
```

Debugging at startup

```
10 GOSUB 1000
20 DEBUGMEM

Outputs:
Shows the status of sprites, variables and RAM after the graphics function, useful for under
```

Debugging after running animations

## Note

- DEBUGMEM doesn't change anything, it's just diagnostic.
- When used in a loop, it can help spot memory leaks over time.
- Useful during development to avoid overflow or full heap crashes.

# DEF FN

## Sintassi

```
DEF FNnome(arg1, arg2, ...) = espressione
```

## Descrizione

DEF FN allows you to define a custom function within the program. The function takes one or more arguments and returns the value of an expression. It is useful for reusing mathematical operations or complex formulas without having to write them multiple times. Functions must have a name starting with FN (e.g. FNADD, FNSQUARE). They cannot contain interactive commands or control commands (e.g. PRINT, GOTO, IF, etc.): only expressions.

## Esempi

```
10 DEF FNADD(X, Y) = X + Y
20 PRINT "5 + 3 = "; FNADD(5, 3)

Expected output:
5 + 3 = 8
```

Sum function with two arguments

```
10 DEF FNSQ(X) = X * X
20 PRINT "7^2 = "; FNSQ(7)

Expected output:
7^2 = 49
```

Calculation of the square

```
10 DEF FNAREA(R) = 3.14 * R * R
20 INPUT R
30 PRINT "AREA ="; FNAREA(R)

Expected output (if you enter 2):
AREA = 12.56
```

Use with variables and formulas

```
10 DEF FNTRIPLA(X) = X * 3
20 FOR I = 1 TO 5
30 PRINT "TRIPLE OF"; I; " = "; FNTRIPLA(I)
40 NEXT I

Expected output:
TRIPLE OF 1 = 3
TRIPLE OF 2 = 6
TRIPLE OF 3 = 9
TRIPLE OF 4 = 12
TRIPLE OF 5 = 15
```

Multiple recall

# DEL

## Sintassi

```
DEL "<nomefile>"
DEL variabile$
DEL SD "<nomefile>"
DEL SD variabile$
DEL SPIFFS "<nomefile>"
DEL SPIFFS variabile$
```

## Descrizione

The DEL command deletes a file from system memory. If executed without specifying memory, the file is deleted from SPIFFS internal memory. By specifying SD, the file is deleted from the SD card. You can explicitly indicate SPIFFS for clarity, but it is not required. The file name can be indicated directly in quotes or via a string variable (e.g. FN$). This command is useful for managing the files present in the different memories of the ESP32, without ambiguity.

## Esempi

```
DEL "main.bas"
```

Delete a file from SPIFFS

```
DEL SD "demo.bas"
```

Delete a file from SD

```
10 LET FN$ = "prog1.bas"
20 DEL FN$
```

Use a variable to specify the file to delete

```
10 LET FN$ = "test.bas"
20 DEL SD FN$
```

Delete via variable from SD

## Note

- DEL accepts only one optional leading argument: SD or SPIFFS.
- You can use string variables to specify the file name.
- If the SD card is not connected or mounted, DEL SD will return a deletion failed error.
- If the file does not exist, an error is reported.

# DELAY n

## Sintassi

```
DELAY n
```

## Descrizione

The DELAY command suspends program execution for n milliseconds. During the delay, the microcontroller does not execute any other code: it is a blocking pause. It is useful for: Waiting between two operations Creating animations or flashes Simulating loading times

## Esempi

```
10 PRINT "CIAO"
20 DELAY 1000
30 PRINT "MONDO"

Expected output (with 1 second pause between the two lines):
CIAO
(monitor pause)
MONDO
```

Pause between two messages

```
10 CLS
20 PRINT "■"
30 DELAY 500
40 CLS
50 DELAY 500
60 GOTO 10

Expected output:
An infinite flash of the "■" symbol every half second.
```

Simulated flashing

```
10 INPUT "ENTER YOUR NAME:"; N$
20 PRINT "ATTENDI..."
30 DELAY 2000
40 PRINT "BENVENUTO "; N$

Expected output:
After input, a 2 second pause before the welcome message.
```

Delay after reading

```
10 FOR I = 5 TO 1 STEP -1
20 PRINT I
30 DELAY 1000
40 NEXT I
50 PRINT "GO!"

Expected output:
A countdown from 5 to 1 with 1 second between each number.
```

Countdown

## Note

- DELAY completely blocks execution (including INPUT, GET, etc.)

- The unit of measurement is the millisecond (1000 = 1 second)

# DELN

## Sintassi

```
DELN n
DELN n1,n2,n3
DELN a-b
```

## Descrizione

The DELN command deletes one or more lines of the program in memory. It can remove a single line, multiple lines separated by commas, or a continuous range. If only the line number is typed (without the DELN keyword), the line is still deleted. If a specified row does not exist, it is ignored without generating errors. This function is useful for quickly editing the listing without having to rewrite it from scratch.

## Esempi

```
10 PRINT "Ciao"
20 PRINT "Line to delete"
30 PRINT "Fine"

Command:
DELN 20
LIST

Expected output:
10 PRINT "Ciao"
30 PRINT "Fine"
```

Remove a specific row

```
10 PRINT "Riga1"
20 PRINT "Riga2"
30 PRINT "Riga3"
40 PRINT "Riga4"

Command:
DELN 10.30
LIST

Expected output:
20 PRINT "Riga2"
40 PRINT "Riga4"
```

Delete multiple rows at once

```
100 PRINT "A"
110 PRINT "B"
120 PRINT "C"
130 PRINT "D"

Command:
DELN 100-120
LIST

Expected output:
130 PRINT "D"
```

Delete a range of rows

```
DELN 500

Se la riga 500 non è presente, il programma resta invariato e non si genera nessun errore.
```

DELN on non-existent line

## Note

- DELN acts only on the list in memory.
- To make changes permanent, use SAVE "filename.bas".
- Works both with single rows and with lists of rows (10,20,30) and ranges (100-200).
- Use it together with EDIT, LIST, NEW, SAVE, LOAD for complete management of BASIC programs.

# DELVAR

## Sintassi

```
DELVAR "file"
DELVAR "file" "chiave"
DELVAR SPIFFS "file"
DELVAR SPIFFS "file" "chiave"
DELVAR SD "file"
DELVAR SD "file" "chiave"
```

## Descrizione

DELVAR deletes data from a JSON file. If you specify only "file" → delete the entire file If you specify "file" "key" → delete only the indicated key, leaving the others intact

## Esempi

```
10 DELVAR SPIFFS "prefs.json" "USER"
20 LISTVARS SPIFFS "prefs.json"
```

Delete a key from SPIFFS

```
10 DELVAR SD "data.json"
```

Completely erase a file on SD

## Note

- If the file does not exist, deleting the file does not produce an error.
- If the key does not exist, the file remains unchanged.
- Deletion of the file is irreversible.

# DEV AUTO ON/OFF

## Sintassi

```
DEV AUTO ON
DEV AUTO OFF
```

## Descrizione

Set whether developer mode should start automatically when the ESP32 boots. When it is active, upon restart the configured display and any PS/2 keyboard are initialized without having to re-enter the commands manually.

## Esempi

```
10 DEV AUTO ON
20 DEV ILI SET 17 16 5 32 18 23 19 3
30 DEV DISPLAY ILI
40 DEV ON
```

The next time you turn it on, the ESP32 will start directly in developer mode with the display ready.

# DEV CLEAR

## Sintassi

```
DEV CLEAR
```

## Descrizione

Clears the currently selected display screen in developer mode. It does not disable dev mode, but only cleans the content.

## Esempi

```
DEV CLEAR
```

Clears the console on the display (useful for having a clean screen).

# DEV DISPLAY

## Sintassi

```
DEV DISPLAY ILI
```

## Descrizione

Select which display to use as the DEV console output (for all print commands: PRINT, LIST, EDIR, messages, etc.). The selection is saved in configuration and remains active on subsequent startups. Note: The chosen display must already be configured (pins/addresses, modes, etc.). Otherwise, an error is reported and the selection does not change.

## Esempi

```
10 DEV DISPLAY ILI
20 DEV CLEAR
30 PRINT "ILI screen active"
```

Select ILI9341

## Note

- You can switch between displays with the same command: DEV DISPLAY ILI.
- If the device is not configured or does not initialize, the command reports an error and the previous display remains.
- Performance: for very long prints activate DEV SERIAL OFF for maximum responsiveness.
- Columns/Rows: may vary depending on the display and its resolution/settings; use your dedicated commands (COLUMNS/ROWS, MAPCOLS/MAPROWS) if present to adapt the layout.

# DEV FONT

## Sintassi

```
DEV FONT <size>
```

## Descrizione

Set the font scaling factor in the DEV console.

## Esempi

```
DEV FONT 1
DEV FONT 2
```

## Note

- Minimum value 1.
- Changing the font clears the display.

# DEV ILI SET

## Sintassi

```
DEV ILI SET <CS> <DC> <RST> <LED> [SCK MOSI MISO] [ROT]
```

## Descrizione

Configure an ILI9341 display by specifying the connection pins. The LED argument indicates the backlight control pin. Optional parameters: SCK MOSI MISO for custom SPI bus, ROT for rotation.

## Esempi

```
DEV ILI SET 17 16 5 32 18 23 19 3
DEV DISPLAY ILI
DEV ON
```

Initialize an ILI9341 with CS=17, DC=16, RST=5, backlight on the GPIO32.

# DEV MAPCOLS

## Sintassi

```
DEV MAPCOLS <n>
```

## Descrizione

Sets the number of logical columns in the developer console. It is used to correctly map text printing on graphic displays.

## Esempi

```
DEV MAPCOLS 40
```

The developer console will use 40 virtual columns.

# DEV MAPROWS

## Sintassi

```
DEV MAPROWS <n>
```

## Descrizione

Sets the number of logical lines of the developer console.

## Esempi

```
DEV MAPROWS 20
```

The console will have 20 virtual rows.

# DEV ON / OFF

## Sintassi

```
DEV ON
DEV OFF
```

## Descrizione

Manages the status of developer mode. DEV ON → activates the developer console and duplicates all Serial.prints also on the selected display. DEV OFF → deactivates the developer console, leaving only the serial port active.

## Esempi

```
DEV DISPLAY ILI
DEV ON
```

Activate the console on ILI and all messages will be visible there too. Automatically restart ESP to initialize the configuration.

```
DEV OFF
```

Deactivate the console on the display, continuing to work only from the serial port.

# DEV PS2 LAYOUT

## Sintassi

```
DEV PS2 LAYOUT <US | IT | DE | FR | ES>
```

## Descrizione

Set the PS/2 keyboard layout, allowing correct key mapping based on country. Supported layouts: US → American (default) IT → Italian DE → German FR → French ES → Spanish

## Esempi

```
DEV PS2 LAYOUT IT
Oppure:
DEV PS2 LAYOUT US
```

## Note

- The layout is saved in prefs.cfg and automatically restored upon reboot.
- Can be edited even with the PS/2 keyboard already running.
- You must have configured the pins (with DEV PS2 SET) to use the keyboard.

# DEV PS2 OFF

## Sintassi

```
DEV PS2 OFF
```

## Descrizione

Disable the PS/2 keyboard in developer mode. When OFF, keystrokes are not read or sent to the DEV console.

## Esempi

```
DEV PS2 OFF
```

## Note

- Stops playing the PS/2 keyboard immediately.
- The pin configuration remains stored, but the keyboard remains inactive until re-enabled with DEV PS2 ON.

# DEV PS2 ON

## Sintassi

```
DEV PS2 ON
```

## Descrizione

Enables the PS/2 keyboard for use in the DEV console. Allows you to type BASIC commands directly from the ILI display in real time.

## Esempi

```
DEV PS2 ON
```

## Note

- Requires prior pin configuration via DEV PS2 SET.
- If the pins have not been configured, the command generates an error.

# DEV PS2 SET

## Sintassi

```
DEV PS2 SET <dataPin> <clkPin>
```

## Descrizione

Configure the pins used by the PS/2 keyboard. After setting the pins, the keyboard can be used to directly type BASIC commands on the DEV console (ILI display). Warning: The actual firmware syntax is DATA → CLK (in this order).

## Esempi

```
DEV PS2 SET 26 27
Effetto
DATA = 26
CLK  = 27
```

The PS/2 keyboard is connected at DATA=26 and CLK=27 and is ready to be enabled with DEV PS2 ON.

## Note

- Pin configuration is saved in prefs.cfg and automatically reloaded at startup.
- Allows the keyboard to work in real time on the DEV console.

# DEV SERIAL ON / OFF

## Sintassi

```
DEV SERIAL ON
DEV SERIAL OFF
```

## Descrizione

Enable or disable mirroring of DEV console output to the Serial (USB) port. With DEV SERIAL ON, everything that normally prints on the display (e.g. PRINT, LIST, status messages) is also replicated on the Serial Monitor. With DEV SERIAL OFF, the output is not replicated on Serial: useful for maximum speed on ILI/OLED during long prints (e.g. LIST, directory).

```
Does not change the baud rate or the serial port settings; it only controls whether or not t
```

The setting is not persistent: after reboot it returns to the default value (usually OFF). If you want to keep it, put the command in your startup script.

## Esempi

```
10 DEV SERIAL ON
20 PRINT "Hello both on screen and USB!"
```

Enable USB logging

```
10 DEV SERIAL OFF
20 LIST
```

Fast LIST (without echo on USB)

```
10 DEV SERIAL OFF
20 PRINT "Fast printing on the display"
30 DEV SERIAL ON
40 PRINT "This line also goes to USB"
50 DEV SERIAL OFF
60 PRINT "Again display only"
```

USB log only during one section

```
10 D=1
20 IF D THEN DEV SERIAL ON
30 PRINT "Debug messages"
40 IF D THEN DEV SERIAL OFF
50 PRINT "Silent output on USB"
```

Conditional debugging

## Note

- Performance: DEV SERIAL OFF is recommended during long printouts (e.g. LIST, file list) to avoid slowdowns due to the USB/Serial buffer.
- Scope: Applies to all supported displays and does not affect keyboard/PS2 input or other peripherals.
- Persistence: the setting is not saved in config.
- Compatibility: does not change Serial.begin(...); uses the baud already set in the firmware. If the Serial Monitor is not open, DEV SERIAL ON does not cause errors, but the output may still accumulate in the USB buffer.

# DEV STATUS

## Sintassi

```
DEV STATUS
```

## Descrizione

Shows the current status of developer mode (active/auto, display selected, font, map, cursor, PS/2).

## Esempi

```
DEV STATUS
```

## Note

- Also shows saved configurations for ILI.

# DHTCALIBRESET

## Sintassi

```
DHTCALIBRESET pin → resetta tutte le calibrazioni associate a quel pin (DHT, DHTT, DHTH).
DHTCALIBRESET ALL → resetta tutte le calibrazioni di tutti i pin.
```

## Descrizione

Clear calibration settings and restore default values:

```
offset = 0
slope = 1
```

It can be used when you want to reset corrections previously set with DHTCALIBSET.

## Esempi

```
10 DHTCALIBSET DHT 2 0.5 -2
20 DHTCALIBRESET 2
30 SDHCALIBSHOW

Expected output:
(no active calibration for pin 2)
```

Reset calibration on pin 2

```
10 DHTCALIBSET DHT 2 1.0
20 DHTCALIBSET DHT 4 -0.5
30 DHTCALIBRESET ALL
40 SDHCALIBSHOW

Expected output:
(no active calibration)
```

Total reset of all calibrations

```
10 DHTCALIBSET DHTT 2 1.0
20 DHTREAD 2 T H
30 PRINT T
40 DHTCALIBRESET 2
50 DHTREAD 2 T H
60 PRINT T

Expected output (example):
T=(temperature+1.0) // before reset
T=(raw_temperature) // after reset
```

After reset reading without correction

## Note

- pin = number of the pin used in the DHTINIT.
- With ALL deletes all recorded calibrations.
- After reset, readings return to the “raw” sensor values.

# DHTCALIBSET

## Sintassi

```
Forma semplice
DHTCALIBSET DHT pin offset [slope] → applica a Temperatura e Umidità
DHTCALIBSET DHTT pin offset [slope] → applica solo alla Temperatura
DHTCALIBSET DHTH pin offset [slope] → applica solo all'Umidità
Forma estesa
DHTCALIBSET DHT pin t_offset h_offset [t_slope] [h_slope]
```

## Descrizione

Set the calibration of DHT readings on the pin. Corrections follow the formula:

```
calibrated_value = raw_value * slope + offset
```

offset: additive correction (°C for T, %RH for H) slope: multiplication factor (default 1.0, must be > 0 and reasonable) Read priority (DHTREAD): DHTT/DHTH (specific) override DHT (general).

## Esempi

```
10 DHTCALIBSET DHT 17 0.5 -2
20 DHTINIT 17 11
30 DHTREAD 17 T H
40 PRINT T; "C  "; H; "%"

Expected output (example):
(Raw_Temperature + 0.5) (Raw_Humidity - 2)
```

Distinct offsets for T and H (long form, 1 line)

```
10 DHTCALIBSET DHT 17 0.3
20 DHTINIT 17 22
30 DHTREAD 17 T H
40 PRINT "T=";T;"  H=";H

Expected output:
T=(raw_T + 0.3) H=(raw_H + 0.3)
```

Same calibration on T and H (simple form)

```
10 DHTCALIBSET DHTT 17 0.0 1.02
20 DHTINIT 17 22
30 DHTREAD 17 T H
40 PRINT "T=";T

Expected output:
T=(raw_T * 1.02)
```

Only Temperature with slope

```
10 DHTCALIBSET DHTH 17 -1.5
20 DHTINIT 17 22
30 DHTREAD 17 T H
40 PRINT H

Expected output:
H=(raw_H - 1.5)
```

Humidity only, offset -1.5 (implicit slope = 1)

```
10 DHTCALIBSET DHT 17 0.5 -2
```

```
Expected output:
DHTCALIBSET: Invalid slope (must be > 0 and reasonable)
```

Slope validation (error if negative or not reasonable)

## Note

- Slope is optional; default 1.0. Must be > 0 (e.g. typical values 0.9…1.1).
- The extended form (DHT pin t_off h_off [t_slope] [h_slope]) is the clearest when you want different corrections on T and H in a single line.
- Calibrations remain active until you change them or perform a reset (e.g. DHTCALIBRESET pin or DHTCALIBRESET ALL).
- Compatible with PRINT, LET, IF because it acts "upstream" of the variables read with DHTREAD.
- In DHTREAD specific calibrations (DHTT:pin, DHTH:pin) have priority over general calibration (DHT:pin).

# DHTCALIBSHOW

## Sintassi

```
DHTCALIBSHOW
```

## Descrizione

Shows the current DHT calibrations (keys and offset/slope parameters) for all configured pins on the screen.

## Esempi

```
10 DHTCALIBSHOW

DHT:2 => offset=0.500 slope=1.000
DHTT:2 => offset=0.000 slope=1.020
DHTH:2 => offset=-1.500 slope=1.000
```

List of active calibrations

```
10 DHTCALIBSET DHT 2 0 1
20 DHTCALIBSHOW

Expected output:
DHT:2 => offset=0.000 slope=1.000
```

After calibration reset

```
10 DHTCALIBSHOW
20 PRINT "FINE"

Expected output:
DHT:... => offset=... slope=...
```

Conditional printing (if needed)

## Note

- It is read-only: it does not change settings.
- Useful for debugging before acquiring measurements.

# DHTINIT (DHT11-DHT22)

## Sintassi

```
DHTINIT pin tipo
```

## Descrizione

Initializes a DHT sensor connected to the indicated pin.

```
type = 11 (DHT11) or 22 (DHT22/AM2302).
```

Must be run before DHTREAD.

## Esempi

```
10 DHTINIT 2 22
20 PRINT "OK"

Expected output:
OK
```

Initialize a DHT22 on pin 2

```
10 LET P = 7
20 LET T = 11
30 DHTINIT P T
40 PRINT "DHT READY"

Expected output:
DHT READY
```

Variables for pin and type

```
10 DHTINIT 2 22
20 DHTINIT 4 22
30 PRINT "MOVED TO PIN 4"

Expected output:
MOVED TO PIN 4
```

Re-initialization on another pin

## Note

- It should be called at least once before reading.
- If you change pin or model, repeat DHTINIT.
- Returns 1/OK (if your platform supports it) or no value.

# DHTREAD

## Sintassi

```
DHTREAD pin varTemp varHum
```

## Descrizione

Reads temperature (°C) and humidity (%) from the DHT initialized on the pin and saves the values in varTemp and varHum.

```
If a variable ends with $, the value is saved as a string; otherwise as a number.
```

Automatically applies any calibrations set with DHTCALIBSET.

## Esempi

```
10 DHTINIT 2 22
20 DHTREAD 2 T H
30 PRINT "T=";T;"C H=";H;"%"

Expected output (example):
T=24.0C H=48.0%
```

Basic reading

```
10 DHTINIT 2 22
20 DHTREAD 2 T$ H$
30 PRINT "T=";T$;"  H=";H$;"%"

Expected output (example):
T=24.0 H=48.0%
```

Use with strings

```
10 DHTINIT 2 22
20 DHTREAD 2 T H
30 IF H > 70 THEN PRINT "HIGH HUMIDITY"

Expected output:
HIGH HUMIDITY
```

Condition on humidity

## Note

- Requires DHTINIT run for that pin.
- In case of read error some implementations produce NaN/sentinel value: handle it with IF.

# DIM

## Sintassi

```
DIM nome_array(n)
DIM nome_array$(n)
DIM FILE   buf$ pos() len() size() contatore
DIM FILESD buf$ pos() len() size() contatore
```

## Descrizione

DIM declares a one-dimensional array (vector), indexed from 0 to n (inclusive). Numeric arrays: without $ Example: DIM A(5) → create A(0)…A(5) (6 elements), initialized to 0. Text array: with $ Example: DIM NAME$(5) → create NAME$(0)…NAME$(5) (6 elements), initialized to "". The assignment of values must be done with LET:

```
LET A(0) = 10
LET NAME$(1) = "LUCA"
```

General rules Valid indices: 0..n (inclusive). Indexes can be constants or numeric variables. Access out of range → error: INDEX OUT OF RANGE. You cannot change the size of an already declared array: use CLEAR ARRAY first.

```
Numeric and text arrays are independent: you can have A() and A$() at the same time.
```

DIM Extensions — File List FILE → loads the list of files from SPIFFS storage. FILESD → loads the list of files from the SD card (error if SD not initialized). buf$ : string variable that will contain all the concatenated names (without separators).

```
pos() : numeric array with the 1-based starting position of each name inside buf$.
len() : numeric array with the length of each name.
size() : numeric array with the size of each file in bytes.
```

counter: numeric variable name of your choice (e.g. NF, NFILES, TOT) in which the number of uploaded files is written.

```
Important: pos(), len(), size() must already be declared with DIM before using DIM FILE….
```

If their capacities differ, the smallest capacity among the three is used. The filled elements are at indices 1..counter (1-based for ease of use), while your arrays still remain allocated 0..n. Possible errors

```
UNDEFINED ARRAY → pos(), len() or size() were not declared.
```

SD NOT INITIALIZED → DIM FILESD … used but SD is not mounted/ready. INDEX OUT OF RANGE → general rule on accessing arrays.

## Esempi

```
10 DIM A(3)
20 LET A(0) = 5
30 LET A(1) = 10
40 LET A(2) = 15
50 FOR I = 0 TO 2
60 PRINT "A("; I; ") = "; A(I)
70 NEXT I

Expected output:
A(0) = 5
A(1) = 10
A(2) = 15
```

Numeric array

```
10 DIM NOME$(2)
20 LET NOME$(0) = "LUCA"
30 LET NOME$(1) = "ANNA"
40 FOR I = 0 TO 1
50 PRINT "NOME("; I; ") = "; NOME$(I)
60 NEXT I

Expected output:
NAME(0) = LUCA
NAME(1) = ANNA
```

Text array

```
10 DIM N(5)
20 FOR I = 0 TO 5
30   LET N(I) = I * I
40 NEXT I
50 PRINT "N(3) ="; N(3)

Expected output:
N(3) = 9
```

Cycle calculation

```
10 DIM MSG$(2)
20 LET MSG$(0) = "CIAO"
30 LET MSG$(1) = MSG$(0) + " MONDO"
40 PRINT MSG$(1)

Expected output:
HELLO WORLD
```

String concatenation

```
10 DIM POS(512)
20 DIM LENN(512)
30 DIM SIZE(512)
40 BUF$=""
50 DIM FILE BUF$ POS() LENN() SIZE() NF
60 IF NF=0 THEN PRINT "No files in SPIFFS"
70 FOR I=1 TO NF
80   N$ = MID$(BUF$, POS(I), LENN(I))
90   PRINT N$; "  "; SIZE(I); " B"
100 NEXT I
```

Full list from SPIFFS

```
10 DIM POS(512)
20 DIM LEN(512)
30 DIM SIZE(512)
40 BUF$=""
50 DIM FILESD BUF$ POS() LEN() SIZE() NFILES
60 IF NFILES=0 THEN PRINT "No files on SD"
70 K = 2
80 N$ = MID$(BUF$, POS(K), LEN(K))
90 PRINT "Files #"; K; ": "; N$; "  "; SIZE(K); " B"
```

SD (already mounted) and printing of a single file

```
10 DIM POS(512): DIM LEN(512): DIM SIZE(512)
20 BUF$=""
```

```
30 DIM FILE BUF$ POS() LEN() SIZE() NF
40 IF NF=0 THEN PRINT "SPIFFS empty"
50 INPUT "Name to search for"; T$
60 FOUND = 0
70 FOR I=1 TO NF
80   N$ = MID$(BUF$, POS(I), LEN(I))
90   IF N$ = T$ THEN FOUND = I : GOTO 120
100 NEXT I
110 PRINT "File not found": END
120 PRINT "Found:"; N$; "  "; SIZE(FOUND); " B"
```

Searching for a file name

## Note

- The counter (last parameter in DIM FILE…) is an arbitrary numerical variable: you decide it (e.g. NF, NFILES, TOT). It is not confidential.
- The classic DIM does not create/update counters: if you want to use one, you manage it yourself (LET K=…) or use the one passed to DIM FILE….
- Positions in pos() are 1-based to simplify use with MID$.
- File names in buf$ are concatenated without separators: always use MID$(buf$, pos(i), len(i)) to read them.

# DIR

## Sintassi

```
DIR
DIR SD
DIR SPIFFS
```

## Descrizione

The DIR command displays the list of files present in the system memory. If run without parameters, displays the contents of SPIFFS internal memory. If the SD argument is specified, it shows the files on the SD card instead. Alternatively, you can explicitly tell SPIFFS to list internal storage (equivalent to DIR with no arguments). This command is useful for quickly viewing the available files, both on SPIFFS and on SD, to load, delete or rename.

## Esempi

```
DIR

Output atteso:
Directory of SPIFFS:
main.bas
prog1.bas
```

List files in SPIFFS internal storage

```
DIR SD

Output atteso:
Directory of SD:
program1.bas
demo.bas
```

List files on the SD card

## Note

- DIR accepts only one optional argument: SD or SPIFFS.
- If no SD card is connected or mounted, the DIR SD command will display an error message (“Unable to open SD root.”).

# DLEVEL(p)

## Sintassi

```
DLEVEL(p)
```

## Descrizione

Always returns the “raw” digital level of the pin, ignoring any software debounce. Use it when you want to know the instant status of the GPIO (useful for detecting “held pressed”, duration of pressure, etc.). Returns:

```
1 if the pin is HIGH (HIGH, ~3.3V)
0 if the pin is LOW (LOW, ~0V)
```

First configure the pin with PINMODE (INPUT + possible PULLUP/PULLDOWN/NOPULL).

## Esempi

```
10 PINMODE 12 INPUT NOPULL
20 V = DLEVEL(12)
30 PRINT "PIN 12 ="; V

Expected output:
0 oppure 1 in base al segnale.
```

Direct reading

```
10 PINMODE 12 INPUT PULLUP
20 IF DLEVEL(12) = 0 THEN PRINT "PREMUTO" ELSE PRINT "RILASCIATO"
30 DELAY 500
40 GOTO 20

Expected output:
continuously prints the real state (0=pressed, 1=released).
```

Button with PULLUP (held state)

```
10 PINMODE 12 INPUT PULLUP DEBOUNCE 60
20 IF DLEVEL(12) = 0 THEN PRINT "HOLD" ELSE PRINT "UP"
30 DELAY 500
40 GOTO 20

Expected output:
DLEVEL shows real-time status; debounce has no effect on DLEVEL.
```

With DEBOUNCE active (demonstration that ignores it)

## Note

- With PULLUP: pressed = 0, released = 1. With PULLDOWN it is the opposite.
- Difference from DREAD:
- DREAD(pin) can become one-shot if you enable DEBOUNCE in PINMODE.
- DLEVEL(pin) is always raw, independent of DEBOUNCE.
- To count single clicks use DREAD with DEBOUNCE; to detect “hold” or pressure times use DLEVEL.

# DO

## Sintassi

```
DO <numero_di_linea>
```

## Descrizione

The DO command allows you to repeatedly execute a single program line at each main cycle, without blocking the general execution of the system. It is useful for cyclic checks on variables, digital inputs, or for repeating simple actions.

## Esempi

```
50 DO 100
100 PRINT "ACTIVE CYCLE"

Expected output:
ACTIVE CYCLE
ACTIVE CYCLE
ACTIVE CYCLE
...(continuously)
```

Print a message cyclically

```
10 PINMODE 12 INPUT PULLUP
20 DO 100
100 IF DREAD(12) = 0 THEN PRINT "PREMUTO"

Expected output:
Prints “PRESSED” every time the button on pin 5 is pressed.
```

Read the state of a button

## Note

- You can use multiple DO commands for different lines
- Does not block the execution of other commands or services such as MQTT
- Useful for simple and repetitive checks
- Valid only on a single line for each DO command

# DO BLOCK

## Sintassi

```
DO BLOCK <inizio> TO <fine>
```

## Descrizione

The DO BLOCK command cycles through a block of program lines, from the initial line to the final line inclusive. All lines included in the block are executed once per cycle, in order. It is useful for grouping multiple statements to be executed cyclically, such as controls, automations, reactions to variables or messages.

## Esempi

```
5 PINMODE 2 OUTPUT NOPULL
10 DO BLOCK 100 TO 110
100 IF TIMEH > 20 THEN DWRITE 2 1
110 IF TIMEH < 21 THEN DWRITE 2 0

Expected output:
The LED turns on after 8pm and turns off before 9pm, automatically.
```

Check if an LED lights up

```
10 DO BLOCK 100 TO 120
100 IF MSG$ = "ACCENDI" THEN PRINT "LIGHT ON"
110 IF MSG$ = "SPEGNI" THEN PRINT "LIGHT OFF"
120 LET MSG$ = ""

Expected output:
Prints “LIGHT ON” or “LIGHT OFF” based on the value of the MSG$ variable.
```

Reaction to a text variable

```
5  PINMODE 2 OUTPUT NOPULL
10 LET S = 0
20 DO BLOCK 100 TO 140
100 IF S = 0 THEN DWRITE 2 1
110 IF S = 1 THEN DWRITE 2 0
120 LET S = S + 1
130 IF S > 1 THEN LET S = 0
140 DELAY 500

Expected output:
the LED on pin 2 turns on/off every half second.
```

Blink LED every 500 ms

## Note

- The rows are all executed every cycle
- May contain IF, LET, DWRITE, WAIT, etc.
- Does not interfere with MQTT or other system operations
- You can use multiple DO BLOCKS in your program

# DREAD(p)

## Sintassi

```
DREAD(pin)
```

## Descrizione

Reads the digital state of the pin. Without DEBOUNCE (PINMODE without the word DEBOUNCE): Returns the raw level of the pin (immediate readout, can repeat as long as you hold it down). With DEBOUNCE (PINMODE with DEBOUNCE [ms]): returns a one-shot pulse when pressed: outputs 0 (with PULLUP) once when you press, then returns to 1 until you release and press again.

```
with PULLDOWN it is the opposite (pressed=1).
```

Returns:

```
1 if the pin is HIGH (HIGH, ~3.3V)
0 if the pin is LOW (LOW, ~0V)
```

First use PINMODE to configure the pin in INPUT (with PULLUP, PULLDOWN or NOPULL). If you always want the raw level (ignoring debounce), use DLEVEL(pin).

## Esempi

```
10 PINMODE 12 INPUT PULLUP
20 V = DREAD(12)
30 PRINT "STATUS OF PIN 12:"; V

Expected output:
0 oppure 1 in base al segnale presente sul pin.
```

Direct reading (without debounce)

```
10 PINMODE 12 INPUT PULLUP DEBOUNCE 60
20 IF DREAD(12) = 0 THEN PRINT "BUTTON PRESSED"
30 GOTO 20

Behavior: Press only once for each full press (press→release), avoiding repetitions while h
```

Button with PULLUP and DEBOUNCE (one-shot)

```
10 PINMODE 12 INPUT PULLUP DEBOUNCE 60
20 IF DLEVEL(12) = 0 THEN PRINT "BUTTON PRESSED" ELSE PRINT "RILASCIATO"
30 DELAY 500
40 GOTO 20

Expected output:
shows the instant state of the pin (0=pressed, 1=released with PULLUP), regardless of debour
```

Continuous “raw” reading with DLEVEL (ignore debounce)

## Note

- With PULLUP: “active-low” circuit → pressed=0, released=1.
- With PULLDOWN it is the opposite (pressed=1).
- DEBOUNCE [ms] in PINMODE filters out bounces and turns DREAD(pin) into a single pressure detector.
- For counters/menus: use DREAD(pin) with DEBOUNCE; for “hold” or hold durations, use DLEVEL(pin).

# DWRITE(p v)

## Sintassi

```
DWRITE(pin valore)
```

## Descrizione

The DWRITE command sets a digital pin of the ESP32 to the state: HIGH (1) → 3.3V voltage LOW (0) → 0V voltage It is used to turn LEDs on or off, activate relays, control signals, etc. The pin must first be configured as OUTPUT using PINMODE.

## Esempi

```
10 PINMODE 2 OUTPUT NOPULL
20 DWRITE 2 1
30 WAIT 1000
40 DWRITE 2 0

Expected output:
The LED lights up for 1 second, then turns off.
```

Turn an LED connected to pin 2 on and off

```
10 PINMODE 2 OUTPUT NOPULL
20 DO BLOCK 20 TO 60
30 DWRITE 2 1
40 WAIT 500
50 DWRITE 2 0
60 WAIT 500

Expected output:
LED connected to GPIO2 flashes at regular intervals.
```

Continuous flashing

```
10 PINMODE 13 OUTPUT NOPULL
20 INPUT A
30 IF A > 100 THEN DWRITE 13 1 ELSE DWRITE 13 0

Expected output:
Pin 13 will be active (HIGH) if A is greater than 100.
```

Conditional control

## Note

- The values 1 and HIGH are equivalent (same for 0 and LOW)
- Virtual or logical output pins can also be controlled in some cases.

# ELSE

## Sintassi

```
IF condizione THEN istruzione1 ELSE istruzione2
```

## Descrizione

The ELSE construct is part of the IF...THEN...ELSE conditional structure. Allows you to execute an alternative statement if the condition is not true. It can be used with single statements on the same line or with GOTO, PRINT, LET, INPUT, etc. BASIC32 does not support multi-line blocks (IF...ENDIF), so the entire logic must be expressed on a single line.

## Esempi

```
10 INPUT A
20 IF A > 0 THEN PRINT "POSITIVO" ELSE PRINT "NEGATIVE OR ZERO"

Expected output (if you enter 5):
POSITIVO
```

Verify a number

```
10 PINMODE 2 OUTPUT NOPULL
20 INPUT V
30 IF V = 1 THEN DWRITE 2 1 ELSE DWRITE 2 0

Expected output:
The GPIO2 pin will be on if V = 1, otherwise off.
```

Choice between two actions

```
10 INPUT T
20 IF T < 20 THEN LET STATO$ = "FREDDO" ELSE LET STATO$ = "CALDO"
30 PRINT "STATE:"; STATO$

Expected output (e.g. input 15):
STATUS: COLD
```

Use with LET for different assignments

```
10 INPUT A
20 IF A < 100 THEN GOTO 100 ELSE GOTO 200
100 PRINT "SMALL NUMBER": END
200 PRINT "BIG NUMBER"

Expected output (e.g. input 50):
SMALL NUMBER
```

With GOTO to jump to different lines

# EXP(x)

## Sintassi

```
EXP(x)
```

## Descrizione

The EXP(x) function returns the value of e raised to x, where e ≈ 2.71828 is the base of the natural logarithms. It is useful for advanced mathematical calculations, exponentials, logistic growth, and scientific operations. The parameter x can be positive, negative or zero.

## Esempi

```
10 PRINT "EXP(1) ="; EXP(1)

Expected output:
EXP(1) = 2.71828
```

Calculate e^1

```
10 X = EXP(2)
20 PRINT "EXP(2) ="; X

Expected output:
EXP(2) = 7,389
```

and raised to the second

```
10 PRINT "EXP(-1) ="; EXP(-1)

Expected output:
EXP(-1) = 0.3679
```

Use EXP with negative numbers

```
10 A = 5
20 PRINT "ORIGINAL VALUE:"; A
30 PRINT "EXP(LOG(A)) ="; EXP(LOG(A))

Expected output:
ORIGINAL VALUE: 5
EXP(LOG(A)) = 5
```

Comparison with powers

# FCLOSE

## Sintassi

```
FCLOSE
```

## Descrizione

Closes the currently open file, freeing resources. You must ALWAYS close a file after use. If no file is open, it produces no error (no-op).

## Esempi

```
10 OPEN "test.txt" W
20 FPRINT "CIAO"
30 FCLOSE
```

Close the file

```
10 OPEN "a.txt" W
20 FPRINT "uno"
30 OPEN "b.txt" W    'a.txt is closed automatically
40 FPRINT "due"
50 FCLOSE
```

Multiple opening

## Note

- It is automatically invoked by OPEN if an open file already exists.
- Always safe to call.

# FNname(...)

## Sintassi

```
FNnome(arg1, arg2, ...)
```

## Descrizione

FNname(...) is used to call a custom function previously defined with DEF FN. The function name must begin with "FN", and the arguments must match those used in the definition in number and order. The function returns a value calculated based on the specified expression. It can be used: in a PRINT in an assignment (LET) in logical conditions (IF)

## Esempi

```
10 DEF FNADD(X,Y) = X + Y
20 PRINT "SUM ="; FNADD(5,3)

Expected output:
SUM = 8
```

Sum function

```
10 DEF FNDOPPIO(X) = X * 2
20 A = FNDOPPIO(4) + 1
30 PRINT "RESULT:"; A

Expected output:
RESULT: 9
```

Use in a more complex expression

```
10 DEF FNSQ(X) = X * X
20 FOR I = 1 TO 5
30 PRINT I; "^2 = "; FNSQ(I)
40 NEXT I

Expected output:
1^2 = 1
2^2 = 4
3^2 = 9
4^2 = 16
5^2 = 25
```

Call in a loop

```
10 DEF FNTRIPLO(X) = X * 3
20 INPUT A
30 IF FNTRIPLO(A) > 20 THEN PRINT "GRANDE" ELSE PRINT "PICCOLO"

Expected output (with input 8):
GRANDE
```

Use in IF condition

# FOR/NEXT

## Sintassi

```
FOR variabile = inizio TO fine [STEP incremento]
```

## Descrizione

The FOR command creates a counter loop that executes one or more instructions until the indicated variable exceeds the final value (positively or negatively, depending on the STEP). Each FOR must be followed by a NEXT, which increments (or decrements) the variable and decides whether to repeat the loop or exit. It is useful for: Repeating blocks of code a defined number of times Scrolling through coordinates, counters, indices or regular sequences Building animations or timed loops

## Esempi

```
10 FOR I = 1 TO 5
20 PRINT "VALUE:"; I
30 NEXT I

Expected output:
VALUE: 1
VALUE: 2
VALUE: 3
VALUE: 4
VALUE: 5
```

Basic cycle from 1 to 5

```
10 FOR N = 10 TO 1 STEP -2
20 PRINT "N: "; N
30 NEXT N

Expected output:
N: 10
N: 8
N: 6
N: 4
N: 2
```

Cycle with negative STEP (decreasing)

```
10 FOR R = 1 TO 2
20   FOR C = 1 TO 3
30     PRINT "LINE:"; R; "COLUMN:"; C
40   NEXT C
50 NEXT R

Expected output:
ROW: 1 COLUMN: 1
ROW: 1 COLUMN: 2
ROW: 1 COLUMN: 3
ROW: 2 COLUMN: 1
ROW: 2 COLUMN: 2
ROW: 2 COLUMN: 3
```

Nested loops (2x3 grid)

## Note

- The variable is created or updated automatically within the loop

• The STEP value can be positive or negative (default: 1)

• Loops can be nested if they use different variables

• The variable names in NEXT must match those in FOR exactly

• NEXT – closes a FOR loop

• RESETFOR – forces cleaning of the for loop and empties all active loops in case of GOTO or errors

# FORMAT

## Sintassi

```
FORMAT SD
FORMAT SPIFFS
```

## Descrizione

FORMAT deletes all files in the root directory of the selected file system. To be safe, if run without arguments, it does nothing and displays a confirmation message. FORMAT SD → delete all files on the SD card FORMAT SPIFFS → delete all files in the internal memory SPIFFS Warning: the files are permanently deleted without the possibility of recovery.

## Esempi

```
FORMAT

Output:
Specify FORMAT SD or FORMAT SPIFFS to confirm formatting.
Warning: all files on the selected memory will be deleted!
```

Error protection

```
FORMAT SD

Output:
FORMAT SD: deleted 12 file(s)
```

Format the SD card

```
FORMAT SPIFFS

Output:
FORMAT SPIFFS: deleted 5 file(s)
```

Format SPIFFS internal memory

## Note

- If the SD is not inserted or not accessible, FORMAT SD will display IO ERROR: SD root not accessible.

# FPRINT

## Sintassi

```
FPRINT <espressione_stringa>
```

## Descrizione

Writes a line to the file opened via OPEN. Automatically adds a newline character to the end of the line. Receives any expression that evaluates to a string: "HELLO" STR$(A)

```
"TEMP=" + STR$(T)
```

Can only be used if the file is opened in W or A mode.

## Esempi

```
10 OPEN "test.txt" W
20 FPRINT "HELLO WORLD"
30 FCLOSE
```

Simple line

```
10 TEMP = 24.6
20 OPEN "sensor.txt" A
30 FPRINT "TEMP=" + STR$(TEMP)
40 FCLOSE
```

Line with numbers

## Note

- In case of file not opened or in invalid mode → error.
- FPRINT does not print to the terminal → writes ONLY to the file.

# FREAD$

## Sintassi

```
FREAD$
```

## Descrizione

Reads a line from the file opened with OPEN. Returns a string (without trailing newline). Returns "" if EOF (end of file) is reached. Can be used: in an assignment directly in PRINT in IF conditions

## Esempi

```
10 OPEN "log.txt" R
20 RIGA$ = FREAD$
30 IF RIGA$ = "" THEN FCLOSE : END
40 PRINT RIGA$
50 GOTO 20
```

Read all the lines

```
10 OPEN SD "config.txt" R
20 PRINT "First line=", FREAD$
30 FCLOSE
```

Single reading

## Note

- Works only in R and A modes.
- Each call reads the next line.
- Do not include the newline (\n), which is automatically removed.

# FREEMEM

## Sintassi

```
PRINT FREEMEM
```

## Descrizione

The FREEMEM command returns the amount of free memory available in bytes for executing the BASIC program. It is useful for monitoring RAM usage and preventing problems related to memory exhaustion.

## Esempi

```
10 PRINT FREEMEM

Expected output:
22480
```

View free memory

```
10 A = FREEMEM
20 PRINT A

Expected output:
22480
```

Save free memory in variable

## Note

- The returned value is in bytes
- Does not require parameters
- Useful for debugging and diagnostics
- Valid only as an expression in PRINT

# FSDEFAULT

## Sintassi

```
FSDEFAULT SD
FSDEFAULT SPIFFS
```

## Descrizione

Sets the default filesystem used by commands that access files. After setting the value, any commands that do not explicitly specify SD or SPIFFS will automatically use the chosen filesystem. The value is stored in the preferences (/prefs.cfg) and is retained even after restarting the ESP32. Effects of the default filesystem on BASIC commands: You can always manually force a filesystem by writing SD or SPIFFS after the command (e.g. DIR SD, SAVE SPIFFS "FILE").

## Esempi

```
10 FSDEFAULT SD
20 SAVE "TEST"

Effect:
The TEST.BAS file is saved on SD even without writing SAVE SD.
```

Set SD as default storage

```
10 FSDEFAULT SPIFFS
20 DIR

Effect:
DIR displays the contents of SPIFFS because it is now the default filesystem.
```

Return to SPIFFS as the default archive

```
10 FSDEFAULT SD
20 SAVE SPIFFS "CONFIG"

Effect:
Even if the default is SD, the CONFIG.BAS file is saved in SPIFFS because it is explicitly s
```

Default SD, but a forced command on SPIFFS

## Note

- The original default value is SPIFFS.
- If the default is SD, but the SD is not mounted, the command produces a warning message.
- The FSDEFAULT command does not require numeric parameters: it accepts only SD or SPIFFS.
- Preferences are automatically saved after the command.

# FUNC / ENDFUNC

## Sintassi

```
FUNC <nome>
FUNC <nome> LOOP
...
ENDFUNC
```

## Descrizione

The FUNC command defines a user function. Functions allow you to create reusable blocks of code.

- If the LOOP keyword is present, the function is executed cyclically in the background (non-blocking) via STARTFUNC.
- All functions must be closed by ENDFUNC.

## Esempi

```
5  FUNC SALUTO
10 PRINT "Hello from Basic!"
20 ENDFUNC
30 CALLFUNC SALUTO

Expected output:
Hello from Basic!
```

Simple function (not looped)

```
5 PINMODE 2 OUTPUT NOPULL
10 FUNC BLINK LOOP
20 DWRITE 2 1
30 DELAY 300
40 DWRITE 2 0
50 DELAY 300
60 ENDFUNC
70 STARTFUNC BLINK

Expected output:
The LED flashes every 300 ms in the background.
```

Cyclic function (loop) for LED flashing

```
10 FUNC SALUTA
20 PRINT "ARGC="; ARGC()
30 PRINT "Text:"; ARG$(1)
40 PRINT "Number:"; ARG(2)
50 ENDFUNC
60 CALLFUNC SALUTA("Ciao", 123)

Expected output:
ARGC=2
Text:Hello
Number:123
```

Function with arguments

```
10 FUNC SOMMASTAMPA
20 PRINT "Result:"; ARG(1) + ARG(2)
30 ENDFUNC
40 A = 10
```

```
50 CALLFUNC SOMMASTAMPA(A+5, 2+3)

Expected output:
Result:20
```

Function with arguments as expressions

## Note

- Each function must start with FUNC name and end with ENDFUNC
- The name must be a single word (no spaces or symbols)
- The code inside is not executed alone: it must be called with CALLFUNC or STARTFUNC
- If defined with LOOP, the function runs continuously and in parallel
- Cyclic functions must be stopped with STOPFUNC
- Can contain any BASIC command (except nested FUNC/ENDFUNC)

# GET

## Sintassi

```
GET
```

## Descrizione

The GET command reads a single character from the keyboard (serial terminal) without waiting for the ENTER key. Returns the ASCII code of the pressed character. If nothing is pressed, it may return -1 or wait (depending on the firmware). It is useful for: reading keys in real time building interactive interfaces reading sequences of commands or manual input

## Esempi

```
10 PRINT "PRESS A KEY:"
20 A = GET
30 PRINT "ASCII CODE:"; A

Expected output (if you press the letter A for example):
PRESS A KEY:
ASCII CODE: 65
```

Press a key and view its ASCII code

```
10 C = GET
20 IF C <> -1 THEN PRINT "KEY:"; C
30 IF C = 81 THEN PRINT "fine" : END
40 GOTO 10

Expected output:
KEY: 72
KEY: 69
KEY: 76
KEY: 76
KEY: 79
KEY: 81
```

Read multiple keys in one cycle

```
10 PINMODE 2 OUTPUT NOPULL
20 PRINT "PRESS 1 FOR ON, 0 FOR OFF"
30 DO BLOCK 40 TO 60
40 C = GET
50 IF C = 49 THEN DWRITE 2 1
60 IF C = 48 THEN DWRITE 2 0

Expected output:
If you press 1, the LED lights up
If you press 0, the LED turns off
```

Perform actions based on the key pressed

## Note

- GET can return -1 if there are no characters in the queue
- ASCII codes of common keys:
- 0 → 48, 1 → 49, A → 65, a → 97

# GOSUB n

## Sintassi

```
GOSUB numero_riga
```

## Descrizione

The GOSUB command allows you to jump to a subroutine (block of code) defined on another line of the program, and execute it. At the end of the subroutine, RETURN is used to return to the line following the one from which GOSUB was called. It is useful for: reusing code structuring the program into logical blocks creating operational functions without DEF FN You can use multiple GOSUBs and nest them, but each GOSUB must have a corresponding RETURN.

## Esempi

```
10 GOSUB 100
20 PRINT "MAIN PROGRAM"
30 END
100 PRINT "SUBROUTINE EXECUTED"
110 RETURN

Expected output:
SUBROUTINE EXECUTED
MAIN PROGRAM
```

Subroutine that prints a line

```
10 FOR I = 1 TO 3
20 GOSUB 100
30 NEXT I
40 END
100 PRINT "EXECUTION NUMBER:"; I
110 RETURN

Expected output:
EXECUTION NUMBER: 1
EXECUTION NUMBER: 2
EXECUTION NUMBER: 3
```

Call the same subroutine multiple times

```
10 INPUT A, B
20 GOSUB 100
30 PRINT "RESULT:"; R
40 END
100 R = A * B
110 RETURN

Expected output (e.g. input 3, 4):
RESULT: 12
```

Subroutine for reusable computation

```
10 GOSUB 100
20 PRINT "THIS WILL NEVER BE EXECUTED"
100 PRINT "FORGOTTEN THE RETURN"
Corrected instead with:
10 GOSUB 100
20 PRINT "THIS WILL BE DISPLAYED"
30 END
```

```
100 PRINT "WITH RETURN"
110 RETURN
```

Common mistake to avoid: If you forget RETURN, the program does not return correctly.

# GOTO n

## Sintassi

```
GOTO numero_riga
```

## Descrizione

The GOTO command transfers program execution to line number n. It is a basic but powerful tool for skipping blocks of code, creating manual loops, or branching program flow based on conditions. However, it is advisable to use it carefully to keep the code readable (avoid the so-called "spaghetti code").

## Esempi

```
10 PRINT "INIZIO"
20 GOTO 40
30 PRINT "YOU CAN'T SEE THIS"
40 PRINT "AFTER THE JUMP"

Expected output:
INIZIO
AFTER THE JUMP
```

Simple jump

```
10 A = 1
20 PRINT A
30 A = A + 1
40 IF A <= 5 THEN GOTO 20

Expected output:
1
2
3
4
5
```

Creating a manual cycle

```
10 PRINT "1. START"
20 PRINT "2. EXIT"
30 INPUT C
40 IF C = 1 THEN GOTO 100
50 IF C = 2 THEN GOTO 200
100 PRINT "START GAME": END
200 PRINT "EXIT FROM THE PROGRAM"

Expected output (if you press 1):
START GAME
```

Text menu management

```
10 PRINT "CONTINUOUS CYCLE"
20 GOTO 10

This cycle is infinite and must be stopped manually.
```

Avoid inadvertent infinite loops

## Note

- The target lines must exist, otherwise the program may crash.
- GOTO can be used inside IF, ELSE, loops or subroutines.

# HTMLOBJ

## Sintassi

```
HTMLOBJ "<riga_html>"
HTMLOBJ espressione_stringa$
```

## Descrizione

The HTMLOBJ command stores a line of HTML code to include in the device-generated web page. Each call to HTMLOBJ appends the string to the internal HTML page. The accumulated lines are sent to the browser when HTMLSTART is executed. The argument can be: a constant string enclosed in quotation marks a string variable (e.g. A$) a string expression (e.g. "Value: " + STR$(T)) The HTML lines remain stored until MEMCLEAN(HTML) is executed or the card is restarted.

## Esempi

```
10 WIFI "SSID" "PASSWORD"
20 HTMLOBJ "<h1>Welcome to Basic32</h1>"
30 HTMLSTART

expected output:
Show a large title on the web page.
```

Simple HTML header

```
10 WIFI "SSID" "PASSWORD"
20 LET T = 23.7
30 LET R$ = "<p>Current temperature:" + STR$(T) + "°C</p>"
40 HTMLOBJ "<h2>Sensore</h2>"
50 HTMLOBJ R$
60 HTMLSTART

expected output:
Inserts the value of a BASIC variable formatted as HTML into the page.
```

Show a BASIC variable

```
10 WIFI "SSID" "PASSWORD"
20 LET T = 22.5
30 LET H = 55.0
40 HTMLOBJ "<h2>Sensor reading</h2>"
50 HTMLOBJ "<table border='1' style='margin:auto;border-collapse:collapse;'>"
60 HTMLOBJ "<tr><th style='padding:6px;'>Size</th><th style='padding:6px;'>Value</th></tr>"
70 LET RIGA$ = "<tr><td style='padding:6px;'>Temperature</td><td style='padding:6px;'>" + ST
80 HTMLOBJ RIGA$
90 LET RIGA$ = "<tr><td style='padding:6px;'>Humidity</td><td style='padding:6px;'>" + STR$(
100 HTMLOBJ RIGA$
110 HTMLOBJ "</table>"
120 HTMLSTART

expected output:
Constructs an HTML table using strings generated by BASIC.
```

HTML table with values

```
10 PINMODE 32 OUTPUT NOPULL
20 WIFI "ssid" "password"
30 HTMLOBJ "<div style='font-family:Arial,sans-serif;background:#202124;color:#fff;text-alig
40 HTMLOBJ "<h2 style='font-size:32px;margin-bottom:40px;'>LED control on board</h2>"
50 HTMLOBJ "<iframe name='exec_hidden' style='display:none;width:0;height:0;border:0;'></ifr
```

```
60 HTMLOBJ "<a href='/exec?cmd=DWRITE%2032%201' target='exec_hidden'>"
70 HTMLOBJ "<button style='font-size:24px;padding:12px 24px;margin:10px;border-radius:10px;b
80 HTMLOBJ "</a>"
90 HTMLOBJ "<a href='/exec?cmd=DWRITE%2032%200' target='exec_hidden'>"
100 HTMLOBJ "<button style='font-size:24px;padding:12px 24px;margin:10px;border-radius:10px;
110 HTMLOBJ "</a>"
120 HTMLOBJ "</div>"
130 HTMLSTART

expected output:
The buttons send commands to /exec but the page always remains the same (the “OK” response i
```

Buttons to control an LED without changing pages

## Note

- Constant strings must be enclosed in double quotes.
- To insert double quotes in HTML it is recommended to use single quotes ' or double the double quotes:
- class=""button""
- HTMLOBJ does not validate HTML: any markup errors will be reflected in the page.
- HTML lines are accumulated between different executions: to start from scratch use:
- MEMCLEAN(HTML)
- HTMLOBJ does not start any web server: to make the page visible you need to call HTMLSTART.

# HTMLSTART

## Sintassi

```
HTMLSTART
```

## Descrizione

The HTMLSTART command starts the integrated web server and makes the HTML page previously constructed via HTMLOBJ accessible via browser. The device must already be connected to Wi-Fi, via: WIFI "ssid" "password" (station mode) WIFIAP "ssid" "password" (Access Point mode, if supported) The generated page can be reached at the IP address reported by the commands: IP → address in Wi-Fi client mode IPAP → address in Access Point mode

## Esempi

```
10 WIFI "ssid" "password"
20 HTMLOBJ "<h2>Basic32 test page</h2>"
30 HTMLOBJ "<p>If you see this message, HTMLSTART is working!</p>"
40 HTMLSTART

expected output:
Start a very simple page to check the functioning of the web server.
```

Test page

```
10 PINMODE 32 OUTPUT NOPULL
20 WIFI "ssid" "password"
30 HTMLOBJ "<div style='font-family:Arial,sans-serif;background:#202124;color:#fff;text-alig
40 HTMLOBJ "<h2 style='font-size:32px;margin-bottom:40px;'>LED control on board</h2>"
50 HTMLOBJ "<iframe name='exec_hidden' style='display:none;width:0;height:0;border:0;'></ifr
60 HTMLOBJ "<a href='/exec?cmd=DWRITE%2032%201' target='exec_hidden'>"
70 HTMLOBJ "<button style='font-size:24px;padding:12px 24px;margin:10px;border-radius:10px;b
80 HTMLOBJ "</a>"
90 HTMLOBJ "<a href='/exec?cmd=DWRITE%2032%200' target='exec_hidden'>"
100 HTMLOBJ "<button style='font-size:24px;padding:12px 24px;margin:10px;border-radius:10px;
110 HTMLOBJ "</a>"
120 HTMLOBJ "</div>"
130 HTMLSTART

expected output:
Allows you to control the LED from your browser. The commands are executed on /exec and the
```

LED control via Web (without changing pages)

## Note

- Must be called after Wi-Fi setup (WIFI / WIFIAP).
- Requires that one or more HTMLOBJ lines have been defined, otherwise the page will be empty or minimal.
- To prevent old pages from mixing with new ones, it is recommended to use MEMCLEAN(HTML) at the beginning of programs that define HTML content.
- BASIC side actions invoked via the web pass through the endpoint:
- /exec?cmd=<BASIC command>
- You can use <iframe> (as in the examples) or JavaScript to execute commands without changing the main page.

# HTTP GET

## Sintassi

```
HTTP GET "url" TO var$
```

## Descrizione

Performs an HTTP GET request (http:// only) and saves the response body in the indicated variable.
If the URL starts with https:// → error: “HTTPS NOT COMPATIBLE”.

```
The special HTTPSTATUS variable contains the HTTP code (200=OK).
```

Full headers can be read with: HTTP HEADERS TO var$.

## Esempi

```
10 WIFI "SSID" "PASSWORD"
20 HTTP GET "http://ip-apicom/json" TO R$
30 IF HTTPSTATUS<>200 THEN PRINT "HTTP error";HTTPSTATUS : GOTO 90
40 HTTP JSON GET "query" FROM R$ TO I$
50 PRINT "Public IP=";I$
90 END
```

Public IP

```
10 WIFI "SSID" "PASSWORD"
20 LET URL$="http://api.openweathermap.org/data/2.5/weather?q=Milano,it&units=metric&appid=L
30 HTTP GET URL$ TO R$
40 IF HTTPSTATUS<>200 THEN PRINT "HTTP error";HTTPSTATUS : GOTO 100
50 HTTP JSON GET "main.temp" FROM R$ TO T$
60 HTTP JSON GET "weather[0].description" FROM R$ TO D$
70 PRINT "Weather forecast:";T$;"°C ";D$
100 END
```

Weather OpenWeatherMap

## Note

- Requires active Wi-Fi (WIFI "ssid" "pwd").
- The Accept-Encoding: identity header is forced → response always in uncompressed text.
- If the response is not JSON, you can print it directly.

# HTTP HEADERS

## Sintassi

```
HTTP HEADERS TO var$
```

## Descrizione

Save the complete HTTP headers of the latest GET/POST response in the indicated variable.

## Esempi

```
10 WIFI "SSID" "PASSWORD"
20 HTTP GET "http://ip-api.com/json" TO R$
30 HTTP HEADERS TO H$
40 PRINT "Status=";HTTPSTATUS
50 PRINT LEFT$(H$,200)
60 PRINT LEFT$(R$,200)
90 END
```

## Note

- Returns the raw headers in a single string.
- Useful for diagnosing redirects, Content-Type, Content-Length.

# HTTP JSON GET

## Sintassi

```
HTTP JSON GET path$ FROM src$ TO var$
```

## Descrizione

Extracts a value from a string containing a JSON response. path$ can include: nested fields → main.temp array indices → weather[0].description If path does not exist → error: JSON Path not found. The function is robust:

```
ignores everything before the first { or [
```

(so it works even if there are headers in the buffer)

## Esempi

```
10 WIFI "SSID" "PASSWORD"
20 LET URL$="http://api.openweathermap.org/data/2.5/weather?q=Milano,it&units=metric&appid=L
30 HTTP GET URL$ TO R$
40 HTTP JSON GET "main.temp" FROM R$ TO T$
50 HTTP JSON GET "weather[0].description" FROM R$ TO D$
60 PRINT "Temp=";T$;"°C ";D$
70 END
```

Weather OpenWeatherMap

```
10 WIFI "SSID" "PASSWORD"
20 HTTP GET "http://httpbin.org/get" TO R$
30 HTTP JSON GET "headers.Host" FROM R$ TO H$
40 PRINT "Server seen as host:";H$
50 END
```

Host from httpbin

## Note

- The function automatically searches for the first { or [ to identify the beginning of the JSON.
- Array indices start at 0.
- Always check HTTPSTATUS=200 before extracting.

# HTTP POST

## Sintassi

```
HTTP POST "url" "contentType" "body" TO var$
```

## Descrizione

Sends an HTTP POST request (http:// only) with the specified body and Content-Type. HTTPS → not supported. The response is saved in var$. HTTP status available in HTTPSTATUS. IMPORTANT If the body contains quotes ", it must be constructed with CHR$(34) or it will be truncated.

## Esempi

```
10 WIFI "SSID" "PASSWORD"
20 LET URL$="http://httpbin.org/post"
30 LET CT$="application/json"
40 LET Q$ = CHR$(34)
50 LET B$ = "{" + Q$ + "hello" + Q$ + ":" + Q$ + "world" + Q$ + "}"
60 HTTP POST URL$ CT$ B$ TO R$
70 HTTP JSON GET "json.hello" FROM R$ TO V$
80 PRINT "Server received:";V$
90 END

Outputs:
Server received:world
```

Echo JSON

```
10 WIFI "SSID" "PASSWORD"
20 LET URL$="http://mioserver.local/api/save"
30 LET CT$="application/x-www-form-urlencoded"
40 LET B$="name=temp&value=23.7"
50 HTTP POST URL$ CT$ B$ TO R$
60 PRINT "Status=";HTTPSTATUS
70 PRINT LEFT$(R$,120)
90 END
```

Send a sensor

## Note

- If the server returns JSON → you can use HTTP JSON GET.
- Avoid bodies that are too large (ESP32 memory limits).
- Works only with HTTP servers (port 80, uncompressed text).

# IF ... THEN [ELSE]

## Sintassi

```
IF condizione THEN istruzione [ELSE istruzione]
```

## Descrizione

```
IF condition THEN statement [ELSE statement]
```

Description:

```
IF ... THEN is the main conditional construct of BASIC32.
```

Evaluates a logical condition and, if true, executes the statement indicated after THEN. If the condition is false and ELSE is specified, execute the alternative statement.

```
Supports operators: =, <>, , >=
```

It is compatible with numbers, strings and functions. Instructions must be on the same line

## Esempi

```
10 INPUT A
20 IF A > 10 THEN PRINT "GREATER THAN 10"

Expected output (if you enter 15):
GREATER THAN 10
```

Simple condition

```
10 INPUT A
20 IF A = 0 THEN PRINT "ZERO" ELSE PRINT "NOT ZERO"

Expected output (if you enter 0):
ZERO
```

Condition with ELSE

```
10 INPUT S
20 IF S = 1 THEN GOTO 100 ELSE GOTO 200
100 PRINT "CHOICE 1": END
200 PRINT "OTHER CHOICE"

Expected output (with input 2):
OTHER CHOICE
```

Use with GOTO

```
10 INPUT A$
20 IF A$ = "CIAO" THEN PRINT "GREETING ACKNOWLEDGED" ELSE PRINT "DIFFERENT STRING"

Expected output (with CIAO input):
GREETING ACKNOWLEDGED
```

Condition on strings

```
10 INPUT X
20 IF X <> 0 THEN PRINT "DIFFERENT FROM ZERO"
```

Negative condition

## Note

- Conditions must evaluate to true/false (logical numeric values)
- Only a single statement can follow THEN and ELSE on the line

# ILI CIRCLE

## Sintassi

```
ILI CIRCLE <x> <y> <r> [r g b] [drawnow]
```

## Descrizione

Draw an empty circle (only the outline). : center of the circle. : radius in pixels. [r g b]: optional color (default white).

```
[drawnow]: optional (1 = draw now, 0 = append).
```

## Esempi

```
10 ILI CIRCLE 60 120 30 255 0 0

Immediately draw a red empty circle of radius 30 centered at (60,120).
```

Immediate red circle

```
10 ILI CIRCLE 80 140 20 255 0 255 0
20 ILI DRAW

The circle appears only after ILI DRAW.
```

Deferred magenta circle

## Note

- For a filled circle, use ILI FILLCIRCLE.
- Useful for buttons, outlines or indicators.

# ILI CLEAR

## Sintassi

```
ILI CLEAR
```

## Descrizione

Clears the screen by filling it with the current background color. The background color can be set with ILI SETBGCOLOR. It does not remove sprites or stored data, but only what is graphically visible.

## Esempi

```
10 ILI SETBGCOLOR 0 0 255
20 ILI CLEAR

Outputs:
the screen is completely filled with blue.
```

Cleaning the blue screen

```
10 ILI RECT 10 10 100 50 255 0 0
20 ILI CLEAR

Outputs:
the red rectangle is erased and the screen returns to the background color.
```

After the drawing

## Note

- Does not remove stored sprites.
- To completely remove sprites from the screen, use ILI SPRITE CLEAR

# ILI DRAW

## Sintassi

```
ILI DRAW
```

## Descrizione

```
Execute all graphics commands that were called with [drawnow] = 0.
```

They are drawn in the order of insertion: PIXEL, LINE, RECT, FILLRECT, CIRCLE, FILLCIRCLE, TEXT. After drawing, the queue is emptied.

## Esempi

```
10 ILI LINE 10 10 100 50 255 0 0
```

Immediate drawing

```
10 ILI RECT 20 20 100 50 255 0 0 0
20 ILI FILLRECT 130 50 60 40 255 255 0 0
30 ILI TEXT 10 200 2 "HELLO BUFFER" 255 255 255 0
40 ILI DRAW
```

Buffered drawing

## Note

- ILI DRAW shows everything that has been enqueued with drawnow=0 at once.
- Useful for avoiding flickering or for complete screen updates (simulated frame buffer).

# ILI FILLCIRCLE

## Sintassi

```
ILI FILLCIRCLE <x> <y> <r> [r g b] [drawnow]
```

## Descrizione

Draws a filled circle of radius centered at (x, y). [r g b]: fill color (default white).

```
[drawnow]: optional (1 = draw now, 0 = append).
```

## Esempi

```
10 ILI FILLCIRCLE 100 100 30 255 0 0

Outputs:
Immediately draw a solid red circle of radius 30 centered at (100,100).
```

Immediate red filled circle

```
10 ILI FILLCIRCLE 140 140 20 0 255 255 0
20 ILI DRAW

Outputs:
The circle is drawn only on line 20.
```

Deferred cyan filled circle

## Note

- If [r g b] is not specified, the color is white.
- The [drawnow] parameter allows you to append the circle to draw it together with other elements.

# ILI FILLRECT

## Sintassi

```
ILI FILLRECT <x> <y> <w> <h> [r g b] [drawnow]
```

## Descrizione

Draws a filled rectangle at the specified coordinates. : coordinates of the upper left corner. : width and height of the rectangle. [r g b]: fill color (default white).

```
[drawnow]: optional (1 = draw now, 0 = append).
```

## Esempi

```
10 ILI FILLRECT 10 10 50 20

Outputs:
Immediately draw a solid white rectangle of 50x20 pixels at (10,10).
```

Immediate white full rectangle

```
10 ILI FILLRECT 40 80 60 30 0 255 0 0
20 ILI DRAW

Outputs:
The green rectangle is drawn only on line 20.
```

Deferred green rectangle

## Note

- Use ILI FILLRECT for backgrounds, buttons or colored areas.
- Combined with [drawnow]=0 you can compose entire screens and display them in a single frame.

# ILI INIT (ILI9341)

## Sintassi

```
ILI INIT cs dc rst rot
ILI INIT cs dc rst rot  [miso] [mosi] [sck]
```

## Descrizione

Initializes the ILI9341 TFT display, specifying the connection pins and screen orientation. Parameters: cs: Chip Select pin dc: Data/Command pin rst: Reset pin rot: screen rotation (allowed values: 0, 1, 2, 3) miso (optional): MISO pin of the SPI bus mosi (optional): MOSI pin of the SPI bus sck (optional): SCK/CLK pin of the SPI bus If the optional SPI pins are not specified, the default SPI configuration already present in the system is used. If they are specified instead, the SPI bus is initialized with the indicated pins. Once initialized, the display is ready to receive graphic commands (lines, rectangles, text, sprites, etc.).

## Esempi

```
10 ILI INIT 17 16 5 0

Result: The screen is initialized with the specified pins and normal (landscape) orientation
Example 2 – Vertical rotation (value 1)
```

Initialization with rotation 0

```
10 ILI INIT 17 16 5 1

Result: The screen is initialized with the specified pins and normal (landscape) orientation
```

Vertical rotation (value 1)

```
10 ILI INIT 17 16 5 0  19 23 18

Result: The display is initialized using the indicated SPI pins (MISO=19, MOSI=23, SCK=18).
```

Initialization with custom SPI pins

## Note

- This command must be run before any other ILI command.
- The rotation can also be changed later by rerunning ILI INIT with a new value.
- After initialization, the screen is automatically cleaned (filled with black).
- You can have a maximum of 256 sprites and 512 elements on the display.

# ILI INIT TOUCH (XPT2046)

## Sintassi

```
ILI INIT TOUCH cs rotation
ILI INIT TOUCH cs rotation  [miso] [mosi] [sck]
ILI INIT TOUCH cs rotation  [miso] [mosi] [sck] [irq]
```

## Descrizione

The ILI INIT TOUCH command initializes the XPT2046 touch controller connected to the ILI9341 display. Parameters: cs: Chip Select pin of the touch controller rotation: defines the orientation of the display (0–3) and is synchronized with the rotation of the TFT miso (optional): MISO pin of the SPI bus mosi (optional): MOSI pin of the SPI bus sck (optional): SCK/CLK pin of the SPI bus irq (optional): IRQ/INT pin of the touch (only if physically connected) Behavior: If the SPI pins are not specified, the configuration is used Default SPI. If miso mosi sck are specified, the SPI is initialized with those pins. If irq is not specified, the system automatically uses IRQ-free mode (SPI polling). If irq is specified, it is used as the touch interrupt line. After initialization, the touch can be used with the commands: ILI TOUCH CALIBRATE ILI TOUCH SPRITE ... ILI TOUCH AREA ... If executed multiple times, the command closes the previous instance of the touch and re-initializes it. It is necessary to calibrate the touch upon first use using ILI TOUCH CALIBRATE.

## Esempi

```
10 ILI INIT TOUCH 5 1
20 PRINT "Touch ready!"

Expected output (serial):
Touch ready!
```

Initialize the touch with CS=5 and rotation 1

```
10 ILI INIT TOUCH 4 3
20 ILI TOUCH CALIBRATE

Expected behavior:
The touch is initialized with CS=4 and rotation 3.
The interactive calibration procedure starts (5 points).
Calibration data is saved automatically.
```

Initialize and then calibrate the touch

```
10 ILI INIT TOUCH 15 1  19 23 18
```

Touch with custom SPI (without IRQ)

```
10 ILI INIT TOUCH 15 1  19 23 18 27
```

Touch with SPI and IRQ

## Note

- rotation must be a number from 0 to 3:
- 0 → standard vertical display
- 1 → display rotated to the right
- 2 → upside down display
- 3 → display rotated to the left
- If touch is not initialized, ILI TOUCH ... commands will return an error.

• Calibration must be done at least once: without it, the coordinate mapping will not be correct.

• If your display does not have the IRQ pin connected, you do not need to specify it.

• You can recall ILI INIT TOUCH multiple times if you change wiring or rotation.

# ILI LED

## Sintassi

```
ILI LED pin value
```

## Descrizione

Turns on or off the backlight of the ILI9341 display via the specified pin.

## Esempi

```
10 ILI LED 32 1   'Turns on the backlight
20 ILI LED 32 0   'Turns off the backlight
```

## Note

- Make sure the specified pin is connected to the display backlight LED.
- value can be 1 (turns on) or 0 (turns off).
- Not all ILI9341 modules require manual LED management.

# ILI LINE

## Sintassi

```
ILI LINE <x1> <y1> <x2> <y2> [r g b] [drawnow]
```

## Descrizione

Draws a line between two points on the TFT screen. : starting point. : end point. [r g b]: optional RGB color (default white).

```
[drawnow]: optional (1 = draw now, 0 = append).
```

## Esempi

```
10 ILI LINE 10 10 100 50

Immediately draw a white line from (10,10) to (100,50).
```

Immediate white line

```
10 ILI LINE 20 60 120 60 255 0 0 0
20 ILI DRAW

The red line is actually drawn only at the ILI DRAW command.
```

Deferred red line

## Note

- The coordinates must be within the limits of the display (typically 320x240).
- The [drawnow] parameter is useful for drawing multiple elements at once.

# ILI PIXEL

## Sintassi

```
ILI PIXEL <x> <y> [r g b] [drawnow]
```

## Descrizione

Draws a single pixel (dot) on the TFT screen at the specified coordinates. : coordinates of the pixel. [r g b]: Optional RGB color (values 0–255). If omitted, the color is white. [drawnow]: optional.

```
1 or absent → draw immediately.
0 → append the pixel and it will be drawn only after ILI DRAW.
```

## Esempi

```
10 ILI PIXEL 50 50 255 255 255

Immediately draw a white pixel at position (50,50).
```

Instant white pixel

```
10 ILI PIXEL 100 100 0 0 255 0
20 ILI DRAW

The blue pixel in (100,100) is only drawn on line 20, when ILI DRAW is performed.
```

Deferred blue pixel

## Note

- Useful for graphical tests or dot drawings.
- If the coordinates are off screen, nothing is drawn.
- The [drawnow] parameter allows you to build a flicker-free screen.

# ILI RECT

## Sintassi

```
ILI RECT <x> <y> <w> <h> [r g b] [drawnow]
```

## Descrizione

Draw an empty rectangle (only the border). : coordinates of the upper left corner. : width and height of the rectangle. [r g b]: border color (default white).

```
[drawnow]: optional (1 = draw now, 0 = append).
```

## Esempi

```
10 ILI RECT 20 20 100 50

Immediately draw an empty 100x50 rectangle at (20,20).
```

Immediate white rectangle

```
10 ILI RECT 30 40 60 30 255 0 0 0
20 ILI DRAW

The red rectangle appears only after running ILI DRAW.
```

Deferred red rectangle

## Note

- Only the edges are drawn. To fill it use ILI FILLRECT.
- The [drawnow] parameter allows you to update multiple graphic objects together.

# ILI SETBGCOLOR

## Sintassi

```
ILI SETBGCOLOR r g b
```

## Descrizione

Sets the default background color used by ILI SPRITE DRAW to fill the screen before drawing individual sprites. r g b: RGB values from 0 to 255

## Esempi

```
10 ILI INIT 17 16 5 1
20 ILI SETBGCOLOR 0 0 255
30 ILI CLEAR

Result: The screen is filled with blue when ILI CLEAR is executed.
```

Blue background

```
10 ILI SETBGCOLOR 0 0 0

Result: the background returns to black.
```

Black background (default)

## Note

- It does not delete anything by itself, but is automatically applied to the next ILI CLEAR.
- Useful for creating animations with different backgrounds or visual resets between frames.
- If you don't use SETBGCOLOR, the background color is black.

# ILI SPEED

## Sintassi

```
ILI SPEED mhz
```

## Descrizione

Set the SPI bus frequency used for communication with the ILI9341 TFT display. The frequency directly affects the speed of graphics updating (rectangles, text, sprites, fills, etc.), but does not change the performance of the BASIC interpreter or internal calculations.

- mhz: SPI frequency in MHz

Typical permitted values: 40, 60, 80 The set value is stored and used: during subsequent display initializations (ILI INIT) immediately, if the display is already initialized

## Esempi

```
10 ILI SPEED 40
20 ILI INIT 17 16 5 0

Result:
The display is initialized using an SPI frequency of 40 MHz, ensuring maximum stability with
```

40 MHz setting (recommended value)

```
10 ILI SPEED 80
20 ILI INIT 17 16 5 0

Result:
SPI communication is increased to 80 MHz, increasing the speed of graphics operations such a
Requires short wiring and compatible display.
```

Increased graphics performance (80 MHz)

```
10 ILI INIT 17 16 5 0
20 ILI SPEED 80
30 ILI FILLRECT 0 0 320 240 0 255 0

Result:
The SPI speed is changed on the fly, without having to reinitialize the display.
```

Changing speed at runtime

## Note

- ILI SPEED does not initialize the display: it must be used in conjunction with ILI INIT.
- If executed before ILI INIT, the value is applied during initialization.
- If executed after ILI INIT, the new speed is applied immediately.
- Too high values can cause:
- graphic artifacts
- corrupt pixels
- screen instability
- If you have problems, reduce the frequency to 40 MHz.

# ILI SPRITE CLEAR

## Sintassi

```
ILI SPRITE CLEAR
```

## Descrizione

Removes all active sprites from memory. After this command, no sprites will be drawn until they are recreated with ILI SPRITE NEW.

## Esempi

```
10 ILI SPRITE CLEAR

Visually clears the screen and frees up sprite memory.
```

Cleans all sprites

## Note

- Useful for completely resetting a scene or initializing a new graphic state.

# ILI SPRITE COLLISION

## Sintassi

```
ILI SPRITE COLLISION id1 TO id2 GOTO line
ILI SPRITE COLLISION id1 TO ANY GOTO line
ILI SPRITE COLLISION id1 TO id2 SET variabile
ILI SPRITE COLLISION id1 TO ANY SET variabile
```

## Descrizione

Checks for rectangular collision (AABB) between sprites created with ILI SPRITE NEW. In case of collision:

```
with GOTO jumps to the indicated line;
```

with SET variable assigns 1 (collision) or 0 (no collision) to a numeric or string variable ($).

## Esempi

```
10 ILI INIT 17 16 5 3
15 ILI LED 32 1
20 ILI SETBGCOLOR 0 0 0
30 ILI CLEAR
40 ILI SPRITE NEW 1 RECT 10 100 10 10 255 0 0
45 ILI SPRITE NEW 2 RECT 120 100 10 10 0 255 0
50 ILI SPRITE DRAW
51 X = 10
52 Y = 100
53 X = X+1
60 ILI SPRITE MOVE 1 X Y
61 DELAY 30
65 ILI SPRITE DRAW 1
66 ILI SPRITE COLLISION 1 TO 2 GOTO 200
70 GOTO 53
200 PRINT "COLLISIONE"
210 END
```

GOTO (rectangular)

```
10 ILI INIT 17 16 5 3
15 ILI LED 32 1
20 ILI SETBGCOLOR 0 0 0
30 ILI CLEAR
40 ILI SPRITE NEW 1 RECT 10 100 10 10 255 0 0
45 ILI SPRITE NEW 2 RECT 120 100 10 10 0 255 0
50 ILI SPRITE DRAW
51 X = 10
52 Y = 100
53 X = X+1
60 ILI SPRITE MOVE 1 X Y
61 DELAY 30
65 ILI SPRITE DRAW 1
66 ILI SPRITE COLLISION 1 TO 2 SET C
67 IF C THEN PRINT "Hit!" ELSE PRINT "Libero"
68  IF C THEN ILI SPRITE DRAW 2
70 GOTO 53
```

SET variable (rectangular)

```
10 ILI INIT 17 16 5 3
```

```
15 ILI LED 32 1
20 ILI SETBGCOLOR 0 0 0
30 ILI CLEAR
40 ILI SPRITE NEW 1 RECT 10 100 10 10 255 0 0
41 ILI SPRITE NEW 2 RECT 120 100 10 10 0 255 0
42 ILI SPRITE NEW 3 RECT 200  40 10 10 0 0 255
50 ILI SPRITE DRAW
51 X = 10
52 Y = 100
53 X = X+1
60 ILI SPRITE MOVE 1 X Y
61 DELAY 30
65 ILI SPRITE DRAW 1
66 ILI SPRITE COLLISION 1 TO ANY SET HIT
67 IF HIT THEN PRINT "1 hit someone!"
68 IF HIT THEN ILI SPRITE DRAW 2
70 GOTO 53
```

ANY with SET (rectangular)

## Note

- IDs must refer to active sprites (ILI SPRITE NEW).
- Variables: if they end with $ → "1"/"0"; otherwise numeric → 1.0/0.0.
- AABB is fast and suitable for most games.
- With many sprites, limit tests per cycle to maintain frame rate.

# ILI SPRITE COLLISIONC

## Sintassi

```
ILI SPRITE COLLISIONC id1 TO id2 GOTO line
ILI SPRITE COLLISIONC id1 TO ANY GOTO line
ILI SPRITE COLLISIONC id1 TO id2 SET variabile
ILI SPRITE COLLISIONC id1 TO ANY SET variabile
```

## Descrizione

Check for circular collision using the circles inscribed in the sprite bounding boxes:

```
center = center of bbox; radius = min(width, height)/2.
```

Useful for round sprites (balls, coins) or for a "softer" detection.

## Esempi

```
10 ILI INIT 17 16 5 3
15 ILI LED 32 1
20 ILI SETBGCOLOR 0 0 0
30 ILI CLEAR
40 ILI SPRITE NEW 1 CIRCLE 10 100 10 10 255 0
45 ILI SPRITE NEW 2 CIRCLE 120 100 10 10 0 255
50 ILI SPRITE DRAW
51 X = 10
52 Y = 100
53 X = X+1
60 ILI SPRITE MOVE 1 X Y
61 DELAY 30
65 ILI SPRITE DRAW 1
66 ILI SPRITE COLLISIONC 1 TO 2 GOTO 200
70 GOTO 53
200 PRINT "CIRCULAR COLLISION"
210 END
```

GOTO (circular)

```
10 ILI INIT 17 16 5 3
15 ILI LED 32 1
20 ILI SETBGCOLOR 0 0 0
30 ILI CLEAR
40 ILI SPRITE NEW 1 CIRCLE 10 100 10 10 255 0
45 ILI SPRITE NEW 2 CIRCLE 120 100 10 10 0 255
50 ILI SPRITE DRAW
51 X = 10
52 Y = 100
53 X = X+1
60 ILI SPRITE MOVE 1 X Y
61 DELAY 30
65 ILI SPRITE DRAW 1
66 ILI SPRITE COLLISIONC 1 TO 2 SET CIRC
67 IF CIRC THEN PRINT "Contact (circles)!" ELSE PRINT "No contact"
68 IF CIRC THEN ILI SPRITE DRAW 2
70 GOTO 53
```

SET variable (circular)

```
10 ILI INIT 17 16 5 3
15 ILI LED 32 1
20 ILI SETBGCOLOR 0 0 0
```

```
30 ILI CLEAR
40 ILI SPRITE NEW 1 CIRCLE 10 100 10 0 255 0
41 ILI SPRITE NEW 2 CIRCLE 120 100 10 255 0 0
42 ILI SPRITE NEW 3 CIRCLE 200  40 10 0 0 255
50 ILI SPRITE DRAW
51 X = 10
52 Y = 100
53 X = X+1
60 ILI SPRITE MOVE 1 X Y
61 DELAY 30
65 ILI SPRITE DRAW 1
66 ILI SPRITE COLLISIONC 1 TO ANY SET HIT
67 IF HIT THEN PRINT "1 touched at least one (circular)"
68 IF HIT THEN ILI SPRITE DRAW 2
70 GOTO 53
```

ANY with SET (circular)

## Note

- Requires active sprites with valid bounding boxes.
- Variables: numeric (1/0) or string ("1"/"0").
- COLLISIONC is slightly more expensive than COLLISION, but useful with rounded sprites.
- Maintain constant cadence: If you do a lot of tests, spread them across multiple frames.

# ILI SPRITE DELETE

## Sintassi

```
ILI SPRITE DELETE id
```

## Descrizione

Delete the sprite with identifier id. The sprite will no longer be drawn by ILI SPRITE DRAW.

## Esempi

```
10 ILI SPRITE DELETE 3

Removes sprite number 3 from memory.
```

## Note

- After deletion, the sprite slot can be reused with ILI SPRITE NEW.
- Does not clear the drawing from the screen until you run ILI SPRITE DRAW.

# ILI SPRITE DRAW

## Sintassi

```
ILI SPRITE DRAW
ILI SPRITE DRAW id
```

## Descrizione

Draw all active and visible sprites on the current screen. Must be called after any changes to sprites (creation, movement, color, content, etc.).

## Esempi

```
10 ILI SPRITE NEW 0 RECT 10 10 40 40 255 0 0
20 ILI SPRITE DRAW

Displays a red rectangle.
```

Draw all the sprites

```
10 ILI SPRITE NEW 0 RECT 10 10 40 40 255 0 0
20 ILI SPRITE DRAW 0

Displays a red rectangle.
```

Draw only the required sprite

## Note

- The ILI SPRITE DRAW command updates the contents of the display based on the current state of the sprites.
- The background color is set with ILI SETBGCOLOR (if specified).
- If not called, modified sprites will not be redrawn.

# ILI SPRITE FLIP

## Sintassi

```
ILI SPRITE FLIP id modo
```

## Descrizione

The ILI SPRITE FLIP command performs a mirroring of the BITMAP image. The mode parameter can be: H → Flip horizontal V → Flip vertical B → Flip full (H + V) The character does not go in quotes. If used on non-BITMAP sprites an error is generated.

## Esempi

```
10 ILI SPRITE FLIP 0 H

Mirror the sprite horizontally
```

Horizontal flip

```
10 ILI SPRITE FLIP 0 V

Mirror the sprite vertically
```

Vertical flip

```
10 ILI SPRITE FLIP 0 B

Apply both flips
```

Total flip

```
10 ILI SPRITE FLIP 0 X

A character other than H, V or B generates an error:
?RUNTIME ERROR
ILI SPRITE FLIP: mode must be H,V or B.
```

Invalid mode

```
10 ILI SPRITE FLIP 3 H

If ID 3 is a Frame:
?RUNTIME ERROR
ILI SPRITE FLIP: sprite is not BITMAP.
```

Non-BITMAP sprites

## Note

- Works only on BITMAP sprites.
- Keeps unchanged: position, size, scale, palette.
- The flip modifies the internal content of the pattern.
- Automatically redraw the sprite.

# ILI SPRITE HIDE

## Sintassi

```
ILI SPRITE HIDE id
```

## Descrizione

Makes the specified sprite invisible. The sprite is no longer drawn until it is reactivated with ILI SPRITE SHOW.

## Esempi

```
10 ILI SPRITE NEW 0 RECT 10 10 40 40 255 0 0
20 ILI SPRITE DRAW
30 DELAY 2000
40 ILI SPRITE HIDE 0
50 ILI SPRITE DRAW

The red rectangle disappears after 2 seconds.
```

Hide sprites

## Note

- The ID must refer to an existing and active sprite.
- The sprite is not deleted: it remains in memory but not visible.

# ILI SPRITE MOVE

## Sintassi

```
ILI SPRITE MOVE id x y
```

## Descrizione

Move the existing sprite to the new coordinates (x, y).

## Esempi

```
10 ILI SPRITE NEW 2 RECT 10 100 30 30 0 255 255
20 ILI SPRITE DRAW
30 DELAY 2000
40 ILI SPRITE MOVE 2 80 100
50 ILI SPRITE DRAW

A light blue rectangle moves to the right after 2 seconds.
```

Move sprite

## Note

- Does not change the size or type of the sprite.
- After moving, you need to call ILI SPRITE DRAW.

# ILI SPRITE NEW

## Sintassi

```
ILI SPRITE NEW id tipo x y parametri [r g b]
ILI SPRITE NEW id RECT x y larghezza altezza [r g b]
ILI SPRITE NEW id FRAME x y larghezza altezza [r g b]
ILI SPRITE NEW id CIRCLE x y raggio [r g b]
ILI SPRITE NEW id LINE x1 y1 x2 y2 [r g b]
ILI SPRITE NEW id CHAR x y "carattere" dimensione [r g b]
ILI SPRITE NEW id NUM x y valore dimensione [r g b]
ILI SPRITE NEW id TEXT x y dimensione "testo" [r g b]
ILI SPRITE NEW id BITMAP x y larghezza altezza pattern_stringa scala
```

## Descrizione

ILI SPRITE NEW id type x y parameters [r g b] id → unique identifier of the sprite type → RECT, FRAME, CIRCLE, LINE, CHAR, NUM, TEXT, BITMAP x y → initial position of the sprite parameters → vary according to the type

```
r g b (optional) → color in RGB format (default = 255 255 255)
```

Sprites are drawn via: ILI SPRITE DRAW

1. FILLED RECTANGLE (RECT) ILI SPRITE NEW id RECT x y width height [r g b] Draws a filled rectangle of the indicated color.

2. FRAME (FRAME) ILI SPRITE NEW id FRAME x y width height [r g b] Draws only the edge of a rectangle.

3. CIRCLE (CIRCLE) ILI SPRITE NEW id CIRCLE x y radius [r g b] Draws a full circle with center at (x,y).

4. LINE (LINE) ILI SPRITE NEW id LINE x1 y1 x2 y2 [r g b] Draws a line from the point (x1,y1) to (x2,y2).

5. SINGLE CHARACTER (CHAR) ILI SPRITE NEW id CHAR x y "character" size [r g b] Draws a single character with a scaled font.

6. NUMBER (NUM) ILI SPRITE NEW id NUM x y size value [r g b] Draws a number with scaled font. Supports variables.

7. TEXT (TEXT) ILI SPRITE NEW id TEXT x y size "text" [r g b] Draws a string of text. Supports concatenations and string variables.

8. BITMAP (BITMAP) ILI SPRITE NEW id BITMAP x y width height pattern_string scale Draws a graphic sprite based on a pattern string, using a color palette defined with: ILI SPRITE PAL index r g b

## Esempi

```
10 ILI SPRITE NEW 0 RECT 20 20 60 40 255 0 0

Draw a red rectangle, 60×40, at (20,20).
```

Draw a red rectangle

```
20 ILI SPRITE NEW 1 FRAME 100 20 60 40 0 255 0

Draw a green frame, 60×40, at (100,20)
```

Draw only the edge of a rectangle

```
30 ILI SPRITE NEW 2 CIRCLE 60 120 30 0 0 255

Draw a blue circle, radius 30, at (60,120)
```

Draw a filled circle with center at (x,y)

```
40 ILI SPRITE NEW 3 LINE 20 180 120 200 255 255 0

Draw a yellow line from (20,180) to (120,200).
```

Draw a line from the point (x1,y1) to (x2,y2)

```
50 ILI SPRITE NEW 4 CHAR 160 40 "A" 3 0 255 255

Draw "A", size 3, cyan color, at (160,40).
```

Draws a single character with scaled font

```
60 N = 123
70 ILI SPRITE NEW 5 NUM 160 80 N 2

Draw 123 in size 2, white color.
```

Draws a number in a scaled font

```
80 A$ = "WORLD"
90 ILI SPRITE NEW 6 TEXT 20 240 2 "HELLO " + A$ 255 0 255

Draw “HELLO WORLD”, size 2, color magenta.
```

Supports concatenations and string variables

```
100 ILI SPRITE PAL 1 255 0 0
110 ILI SPRITE PAL 2 255 255 0
120 P$ = "01110" + "12221" + "12221" + "12221" + "01110"
130 ILI SPRITE NEW 7 BITMAP 40 40 5 5 P$ 4

Draws a 5×5 bitmap scaled ×4.
After creating one or more sprites, to render them on the display:
ILI SPRITE DRAW
```

Draws a graphic sprite based on a pattern string, using a color palette defined with: ILI SPRITE PAL index r g b

## Note

- RGB color is optional (default: white).
- Sprites remain in memory until they are overwritten or deleted.
- The bitmap pattern must contain digits corresponding to the palette indexes.
- Support numeric and string variables.

# ILI SPRITE PAL

## Sintassi

```
ILI SPRITE PAL indice r g b
```

## Descrizione

Set a color in the global BITMAP sprite palette. The palette contains 16 colors (0–15).
0-1-2-3-4-5-6-7-8-9-A-B-C-D-E-F Color 0 is always transparent.

## Esempi

```
10 ILI SPRITE PAL 0 0 0 0      ' Trasparente
20 ILI SPRITE PAL 1 255 0 0    ' Rosso
30 ILI SPRITE PAL 2 0 255 0    ' Verde
40 ILI SPRITE PAL 3 0 0 255    ' Blu
50 ILI SPRITE PAL 4 255 255 0  ' Giallo
```

Definition of a basic palette

## Note

- Used by all BITMAP sprites in the program.
- Each bitmap sprite interprets the characters 1..F as indices of this palette.
- Color 0 is not drawn (transparency).

# ILI SPRITE ROTATE

## Sintassi

```
ILI SPRITE ROTATE id gradi
```

## Descrizione

The ILI SPRITE ROTATE command rotates a BITMAP sprite by a specified number of degrees. The command internally modifies the pixel map and automatically redraws the sprite. If the sprite is not of type BITMAP, an error is generated.

## Esempi

```
10 ILI SPRITE ROTATE 0 90
```

Rotates the BITMAP sprite with ID 0 90 degrees clockwise

```
10 ILI SPRITE ROTATE 5 180
```

Rotate sprite 5 by flipping it completely 180°

```
10 ILI SPRITE ROTATE 2 90

Expected output:
?RUNTIME ERROR
ILI SPRITE ROTATE: sprite is not BITMAP
```

If ID 2 is a RECT, the command fails

## Note

- Works only on sprites created with ILI SPRITE NEW ... BITMAP.
- Keeps position, size and scale unchanged.
- Automatically updates the drawing on the display.

# ILI SPRITE SETBITMAP

## Sintassi

```
ILI SPRITE SETBITMAP id pattern$
```

## Descrizione

Change the graphic pattern of an already created BITMAP sprite. It is used for animations (walking, movement, snake sprites, objects that change shape...).

## Esempi

```
10 P1$ = "01110" + "10001" + "11111" + "10001" + "01110"
20 P2$ = "01110" + "11011" + "11111" + "11011" + "01110"
30 ILI SPRITE PAL 1 255 255 255
40 ILI SPRITE NEW 0 BITMAP 40 40 5 5 P1$ 4
50 FOR I = 1 TO 10
60   ILI SPRITE SETBITMAP 0 P1$
70   ILI SPRITE DRAW 0
80   DELAY 100
90   ILI SPRITE SETBITMAP 0 P2$
100  ILI SPRITE DRAW 0
110  DELAY 100
120 NEXT I
```

Simple animation (2 frames)

## Note

- The sprite maintains: position, scale, size, visibility.
- Only changes the pattern.
- The new pattern must have the same W×H dimension as the original sprite.
- To see the change you need an ILI SPRITE DRAW id.

# ILI SPRITE SETCHAR

## Sintassi

```
ILI SPRITE SETCHAR id "carattere" r g b
```

## Descrizione

Modifies the character displayed by the CHAR sprite.

## Esempi

```
10 ILI SPRITE SETCHAR 4 "B" 255 255 0
20 ILI SPRITE DRAW
```

Change the character in sprite 4 to yellow “B”.

## Note

- Works only on CHAR type sprites.

# ILI SPRITE SETNUM

## Sintassi

```
ILI SPRITE SETNUM id valore r g b
```

## Descrizione

Modifies the number displayed by a sprite of type NUM.

## Esempi

```
10 ILI SPRITE SETNUM 5 2024 255 105 180
20 ILI SPRITE DRAW
```

View number 2024 in pink

## Note

- Works only on NUM sprites.
- The number is treated as text.

# ILI SPRITE SETTEXT

## Sintassi

```
ILI SPRITE SETTEXT <id> <testo/expr> [r g b]
```

## Descrizione

Change the content and (optionally) color of the TEXT sprite. The parameter can now be: a quoted string ("HELLO") a string variable (A$)

```
a concatenated expression ("VAL=" + N + " " + A$)
a string function (STR$(), LEFT$(), etc.)
```

a number or numeric expression (automatically converted to string) [r g b] values are optional (default white).

## Esempi

```
10 ILI SPRITE SETTEXT 6 "Changed text" 0 255 0
20 ILI SPRITE DRAW
```

Change the text in sprite 6 to Changed Green Text

```
10 N = 42
20 ILI SPRITE SETTEXT 1 "VAL=" + N 255 0 0
30 ILI SPRITE DRAW 1
```

Change sprite 1 to show VAL=42 in red

```
10 A$ = "WORLD"
20 ILI SPRITE SETTEXT 2 "HELLO " + A$
30 ILI SPRITE DRAW 2
```

Change sprite 2 showing HELLO WORLD

## Note

- Works only on TEXT type sprites.
- It is no longer mandatory to use double quotes: you can concatenate strings, variables and numbers with +.
- Color [r g b] optional (default white).
- After editing, use ILI SPRITE DRAW or ILI SPRITE DRAW id to redraw the sprite.

# ILI SPRITE SHOW

## Sintassi

```
ILI SPRITE SHOW id
```

## Descrizione

Makes a previously hidden sprite visible again with ILI SPRITE HIDE.

## Esempi

```
10 ILI SPRITE SHOW 1
20 ILI SPRITE DRAW
```

Sprite 1 becomes visible again

## Note

- The position, content and color are not changed.

# ILI TEXT

## Sintassi

```
ILI TEXT <x> <y> <size> <testo/expr> [r g b] [drawnow]
```

## Descrizione

Show text on the display at the specified position, with size and color. The parameter can be:

- a string in quotes ("HELLO")
- a string variable (A$)
- an expression concatenated with + (e.g. "HELLO " + A$ + N)
- a string function (LEFT$(A$,3), STR$(X))
- a number or numeric expression (converted to a string)

[r g b]: text color (default white).

```
[drawnow]: optional (1 = print now, 0 = append for ILI DRAW).
```

## Esempi

```
10 ILI TEXT 10 50 2 "HELLO ILI" 0 255 255

Writes "HELLO ILI" in cyan, size 2, at position (10,50).
```

Immediate fixed text

```
10 A$ = "WORLD"
20 ILI TEXT 20 80 2 "HELLO " + A$

Shows "HELLO WORLD" in white.
```

Text concatenated with string variable

```
10 N = 123
20 ILI TEXT 10 120 2 "VALUE=" + N 255 0 0 0
30 ILI DRAW

Writes "VALUE=123" in red only at the ILI DRAW command.
```

Deferred red text

## Note

- You can concatenate text, numbers and variables.
- [r g b] is optional (0–255).
- [drawnow]=0 appends the text to draw it with other elements.
- The text remains visible until you run ILI CLEAR.

# ILI TOUCH AREA

## Sintassi

```
ILI TOUCH AREA x y w h GOTO line [DOWN|UP|HOLD ms|DEBOUNCE ms]
ILI TOUCH AREA x y w h SET var [DOWN|UP|HOLD ms|DEBOUNCE ms]
```

## Descrizione

Checks whether the touch falls within the specified rectangular area (ILI screen coordinates, after calibration). With the options you can distinguish when the event is triggered: The area accepts negative dimensions (w, h): it is automatically normalized.

## Esempi

```
10 X = 100
20 Y = 180
30 W = 120
40 H = 40
50 ILI FILLRECT X Y W H 0 120 200
60 ILI TOUCH AREA X Y W H GOTO 200 DOWN DEBOUNCE 200
70 GOTO 60
200 PRINT "Tap in the area!"
210 END
```

Area with jump

```
10 ILI TOUCH AREA 20 20 100 60 SET T DOWN
20 IF T THEN PRINT "Tap in the box 20,20,100,60"
30 GOTO 10
```

Area with variable

## Note

- If the touch is not available or not pressed, no jump occurs and with SET the variable receives 0.
- Touch coordinates are mapped based on rotation and calibration (minRawX/maxRawX/minRawY/maxRawY).
- Use DEBOUNCE to avoid unwanted repetitions if you hold down your finger.
- Use HOLD to create “long press” buttons.

# ILI TOUCH CALIBRATE

## Sintassi

```
ILI TOUCH CALIBRATE
ILI TOUCH CALIBRATE FORCE
```

## Descrizione

Launches a touch calibration wizard. The system asks you to tap a few points at the corners of the screen to measure the raw panel values (min/max on X and Y) and align the touch coordinates with the TFT coordinates. Performs calibration in the current display rotation. Updates the internal parameters used for mapping (typically minRawX, maxRawX, minRawY, maxRawY). Requires touch to be already initialized with ILI INIT TOUCH ....

## Esempi

```
10 ILI INIT 17 16 5 2
20 ILI LED 32 1
30 ILI INIT TOUCH 13 2
40 IF TOUCH CALIBRATE = 0 THEN ILI TOUCH CALIBRATE
50 ILI CLEAR
60 ILI TEXT 10 20 2 "Calibration completed" 0 255 0

Result:
the procedure starts; at the end the touches are aligned with the screen coordinates.
```

Simple calibration

```
10 ILI INIT 17 16 5 2
20 ILI LED 32 1
30 ILI INIT TOUCH 13 2
40 IF TOUCH CALIBRATE = 0 THEN ILI TOUCH CALIBRATE
50 ILI CLEAR
60 ILI TEXT 15 0 2 "Touch the square" 255 255 255
70 ILI SPRITE NEW 0 RECT 90 40 50 50 255 0 0
80 ILI SPRITE DRAW
90 ILI TOUCH SPRITE 0 GOTO 200
100 GOTO 90
200 ILI TEXT 25 120 2 "Touch detected!" 0 255 0
210 END
```

Calibration + pointer test

```
10 ILI INIT 17 16 5 2
20 ILI LED 32 1
30 ILI INIT TOUCH 13 2
40 IF TOUCH CALIBRATE = 0 THEN ILI TOUCH CALIBRATE
50 ILI CLEAR
60 ILI TEXT 15 0 2 "Touch a square" 255 255 255
70 ILI SPRITE NEW 0 RECT 90 40 50 50 255 0 0
80 ILI SPRITE NEW 1 RECT 90 100 50 50 0 0 255
90 ILI SPRITE DRAW
100 ILI TOUCH SPRITE 0 GOTO 200
110 ILI TOUCH SPRITE 1 GOTO 220
120 GOTO 100
200 ILI TEXT 30 180 2 "Touched red!" 0 255 0
210 END
220 ILI TEXT 30 180 2 "Touched blue!" 0 255 0
230 END
```

Calibration + double sprite test

```
10 ILI INIT 17 16 5 2
20 ILI LED 32 1
30 ILI INIT TOUCH 13 2
40 ILI TOUCH CALIBRATE FORCE
```

Force calibration even if already present

## Note

- Run ILI TOUCH CALIBRATE after ILI INIT and ILI INIT TOUCH.
- Don't put your finger on the screen before startup: touch only when prompted.
- If you change the display rotation, repeat the calibration.

# ILI TOUCH SPRITE

## Sintassi

```
ILI TOUCH SPRITE id GOTO line [DOWN|UP|HOLD ms|DEBOUNCE ms]
ILI TOUCH SPRITE id SET var [DOWN|UP|HOLD ms|DEBOUNCE ms]
ILI TOUCH SPRITE ANY SET var [DOWN|UP|HOLD ms|DEBOUNCE ms]
```

## Descrizione

Detects tapping on a sprite's bounding box. With the options you can distinguish when the event is triggered: It works with all types of sprites (RECT, FRAME, CIRCLE, LINE, TEXT, CHAR, NUM, PIXELS...).

## Esempi

```
10 ILI SPRITE NEW 5 RECT 40 40 50 30 0 200 0
20 ILI SPRITE DRAW
30 ILI TOUCH SPRITE 5 SET HIT DOWN DEBOUNCE 150
40 IF HIT THEN PRINT "You have touched sprite 5"
50 GOTO 30
```

Single sprite with variable 1/0

```
10 ILI TOUCH SPRITE ANY SET S DOWN
20 IF S > 0 THEN PRINT "Sprite touched:"; S
30 GOTO 10
```

Any sprite → variable = ID

```
10 ILI SPRITE NEW 3 RECT 100 100 80 40 255 0 0
20 ILI SPRITE DRAW
30 ILI TOUCH SPRITE 3 GOTO 200 DOWN
40 GOTO 30
200 PRINT "Touched sprite 3!"
210 END
```

Sprite with jump

## Note

- Sprites must be active and visible, otherwise the touch is ignored.
- DOWN, UP, HOLD, DEBOUNCE also work in combination (e.g. DOWN DEBOUNCE 200).
- The test uses the bounding box (x,y,w,h).
- For truly circular tests you can still use COLLISIONC if necessary.
- If touch is not available or not pressed, no jump occurs and the SET variables are 0.

# INCLUDE

## Sintassi

```
INCLUDE "nomefile.bas"
INCLUDE SD "nomefile.bas"
INCLUDE SPIFFS "nomefile.bas"
```

## Descrizione

The INCLUDE command inserts the contents of a .bas file present on SPIFFS or SD into the BASIC program. The included file is: read and compiled line by line, automatically renumbered without conflicts, added to the program in memory, made available for execution (functions, code, etc.). Included content is not shown in the LIST command, which displays only the INCLUDE line. When the INCLUDE line is deleted, all code loaded from that file is automatically removed. This command is useful for: dividing the program into modules creating separate game levels including sprites, routines or external functions keeping the main code cleaner

## Esempi

```
10 PRINT "INIZIO"
20 INCLUDE "basic.module"
30 CALLFUNC TEST
40 END
RUN

module.bas file
10 FUNC TEST
20   PRINT "works!"
30 ENDFUNC

Outputs:
INIZIO
works!
```

Include a module from SPIFFS

## Note

- The listing only shows INCLUDES, not the contents of the file.
- Functions defined in the included files can be called normally.
- Includes can be nested up to 8 levels.
- Changing the line number of the INCLUDE correctly updates all loaded lines.
- Use NEW to completely clear the program.

# INITRTC

## Sintassi

```
INITRTC <indirizzo_hex>
```

## Descrizione

ATTENTION: Before using INITRTC it is mandatory to call the WIRE command to specify the SDA and SCL pins of the I2C bus.

The INITRTC command initializes the RTC (Real Time Clock) module connected to the I2C bus. Accepts the I2C address of the RTC module as a parameter, in decimal or hexadecimal form (0x68). Check that the device responds to that address. Start the RTC library and, if the module is configured correctly, synchronize the system time with that of the RTC. It is used to manage: READING TIME (TIMEH, TIMEM, TIMES) READING DATE (DATEY, DATEM, DATED) and other commands that depend on real time. Practical examplesThe INITRTC command initializes the RTC (Real Time Clock) module connected to the I2C bus. Accepts the I2C address of the RTC module as a parameter, in decimal or hexadecimal form (0x68). Check that the device responds to that address. Start the RTC library and, if the module is configured correctly, synchronize the system time with that of the RTC. It is used to manage: READING TIME (TIMEH, TIMEM, TIMES) READING DATE (DATEY, DATEM, DATED) and other commands that depend on real time.

## Esempi

```
10 WIRE 21 22
20 INITRTC 0x68
30 PRINT TIMEH; ":"; TIMEM; ":"; TIMES

Expected output:
14:03:52
```

Initialize the RTC and print the time

```
10 WIRE 21 22
20 INITRTC 0x68
30 PRINT "RTC initialized successfully!"
```

Check the presence of the RTC

## Note

- WIRE is required before INITRTC → if it was not called, an error is generated.
- The typical address of the DS1307/DS3231 RTC is 0x68.
- The module must be correctly connected to the SDA and SCL pins indicated by WIRE.
- The RTC must be powered.
- Works with DS1307, DS3231 and compatible modules.

# INITSD

## Sintassi

```
INITSD <cs> <mosi> <miso> <sck> [0|1] [<cs_alt1> <cs_alt2> ...]
```

## Descrizione

The INITSD command initializes the SD card by specifying the pins to use for the SPI interface. When the [0|1] argument is specified, the command saves to /prefs.cfg:

[[[P]]]the pins used for the SD (CS, MOSI, MISO, SCK); choosing auto-mount (SD_AUTO);[[[/P]]]

any additional CS pins (EXTRA_CS).

[[[P]]]At startup, if SD_AUTO=1, the firmware: sets HIGH all pins listed in EXTRA_CS; automatically mounts the SD with the saved pins;[[[/P]]]

shows the free space available.

## Esempi

```
10 INITSD 13 23 19 18
20 PRINT SDFREE

Expected output:
Shows free space on the SD if initialized correctly.
```

Initialize an SD with custom pins

```
10 INITSD 13 23 19 18 1
20 PRINT SDFREE

Expected output:
The SD is initialized, settings are saved in /prefs.cfg,
and the card will be mounted automatically on subsequent startups.
```

Initialize an SD and save preferences for automatic startup

```
10 INITSD 13 23 19 18 1 17 4
20 PRINT SDFREE

Explanation:
13 23 19 18 → pin SD (CS, MOSI, MISO, SCK)
1 → abilita auto-mount e salva le preferenze
17 4 → porta HIGH i pin CS del display ILI (17) e del touch (4)
Expected output:
Shows free space on the SD and saves preferences in /prefs.cfg.
```

Initialize an SD with other SPI devices connected

## Note

- You must run INITSD before using commands that access SD files (e.g. LOAD, DIR SD, SDFREE), unless auto-mount is enabled.
- The SD card must be formatted FAT32.
- The correct pin assignment depends on the hardware wiring.
- After initialization the SD is ready for reading and writing.
- To check the status of the SD and the available space: PRINT SDFREE

• The /prefs.cfg configuration file is used at startup to manage automatic initialization.

# INPUT

## Sintassi

```
INPUT variabile
```

## Descrizione

The INPUT command allows you to request a value from the keyboard (via serial terminal). It can be used to receive both: numeric values (e.g. A,

```
It is not necessary to specify the type: BASIC automatically distinguishes based on the vari
```

## Esempi

```
10 INPUT A
20 PRINT "YOU HAVE ENTERED:"; A

Expected output (if you enter 42):
? 42
YOU ENTERED: 42
```

Request for an integer

```
10 INPUT NOME$
20 PRINT "CIAO "; NOME$

Expected output (if you enter MARCO):
? MARK
HELLO MARCO
```

Request for a string

```
10 INPUT A
20 INPUT B
30 PRINT "SUM:"; A + B

Expected output (e.g. inputs 5 and 7):
? 5
? 7
SUM: 12
```

Multiple INPUT (with two lines)

```
10 INPUT X
20 IF X < 0 THEN PRINT "NEGATIVE VALUE" ELSE PRINT "OK"

Check if the entered value is positive
```

Simple validation

```
10 PRINT "ENTER YOUR NAME:"
20 INPUT N$
30 PRINT "BENVENUTO "; N$

Expected output:
ENTER YOUR NAME:
? LUCA
WELCOME LUCA
```

With personalized message

## Note

• The symbol ? is automatically shown as an input prompt

# INT(x)

## Sintassi

```
INT(x)
```

## Descrizione

The INT(x) function returns the integer part of a number x, truncates decimals, and rounds toward zero. It is useful for converting a decimal number to an integer in a controlled way, for example to handle loops, array indices, custom rounding. INT(4.7) → 4 INT(-2.3) → -2 Don't confuse this with rounding: INT doesn't round up or down — it just cuts the decimals.

## Esempi

```
10 A = 5.89
20 PRINT "INT(5.89) ="; INT(A)

Expected output:
INT(5.89) = 5
```

Integer part of a positive number

```
10 PRINT "INT(-3.99) ="; INT(-3.99)

Expected output:
INT(-3.99) = -3
```

Integer part of a negative number

```
10 DIM A(10)
20 X = 5.7
30 A(INT(X)) = 99
40 PRINT A(5)

Expected output:
99
```

Use to access array indexes. Remove the decimal from a division and use the result as an index.

```
10 FOR I = 1 TO 10
20 PRINT INT(RND(10))
30 NEXT I

Expected output:
(esempio)
7
2
9
0
5
...
```

Use in a loop for integer random numbers. Generate 10 random numbers from 0 to 9:

# IP

## Sintassi

```
IP
```

## Descrizione

The IP command prints the current IP address of the device if it is connected to a Wi-Fi network via the WIFI command. It is used to easily view the address with which the device can be reached in the local network.

## Esempi

```
10 WIFI "ssid" "password"
20 WAIT 3000
30 PRINT IP

Expected output:
192.168.1.42
```

Wi-Fi connection and IP printing

```
5 WIFI "ssid" "password"
10 DELAY 5000
20 PRINT "MY IP IS:"
30 IP

Expected output:
MY IP IS:
192.168.1.50
```

Wait and then view IP

## Note

- Shows the IP address obtained via DHCP
- Only works after successful WIFI

# IPAP

## Sintassi

```
IPAP
```

## Descrizione

The IPAP command prints the device's local IP address when it is in Access Point mode (created via the AP command). This is useful for knowing where to access the device when it has created its own Wi-Fi network.

## Esempi

```
10 WIFIAP "Basic32AP" "mypassword"
20 DELAY 2000
30 PRINT IPAP

Expected output:
192.168.4.1
```

Start an Access Point and see the IP

## Note

- Show the device's IP address as a hotspot
- The default IP is usually 192.168.4.1
- Works only after WIFIAP

# IR HALT

## Sintassi

```
IR HALT
```

## Descrizione

Empties/releases the “current” IR code stored (internal buffer). It serves to prevent the same code from being reused/read multiple times.

## Esempi

```
10 IR INIT 15
20 IR WAIT C$ 5000
30 IF C$<>"" THEN PRINT "Letto=";C$
40 IR HALT
```

Read and then “clean”

```
10 IR INIT 15
20 IR WAIT C$ 0
30 PRINT "Codice=";C$
40 IR HALT
50 GOTO 20
```

Use in loop (avoid repetitions)

## Note

- It is not mandatory, but recommended in programs that read often.
- It is the conceptual equivalent of RFID HALT: it "frees" the state and prepares for new clean readings.

# IR INIT

## Sintassi

```
IR INIT rxPin
IR INIT rxPin txPin
IR INIT rxPin txPin khz
```

## Descrizione

Initialize the infrared system.

```
With rxPin you enable reception; by adding txPin you also enable sending.
```

khz is the carrier frequency for sending (typical 38 kHz). Parameters rxPin → IR receiver pin (sensor data output) txPin → IR LED pin (optional, required to send) khz → frequency in kHz (optional, default 38)

## Esempi

```
10 IR INIT 15
20 PRINT "IR RX ready"
```

Reception only

```
10 IR INIT 15 4 38
20 PRINT "IR RX/TX ready"
```

Receive + send (38kHz)

## Note

- It should be called only once at startup (or when you change pin/frequency).
- If you don't do IR INIT, IR WAIT can't receive and IR SEND can't send.
- To send it is mandatory to have also initialized txPin.

# IR SEND

## Sintassi

```
IR SEND code$
IR SEND code$ reps
```

## Descrizione

Send an IR code$ (string) via the transmitter (IR LED). reps repeats sending several times (default 1). Parameters code$ → code string (e.g. NEC:...:... or RAW38:...) reps → number of repetitions (optional)

## Esempi

```
10 IR INIT 15 4 38
20 IR SEND "NEC:20DF10EF:32"
```

Submit a known NEC code

```
10 IR INIT 15 4 38
20 PRINT "Press a button..."
30 IR WAIT C$ 5000
40 IF C$="" THEN PRINT "Timeout": END
50 IR SEND C$ 1
60 IR HALT
```

Send the code you just received

```
10 IR INIT 15 4 38
20 IR SEND "RAW38:9000,4500,560,560,..." 3
```

Repeat 3 times

## Note

- Requires IR INIT with txPin, otherwise cannot send.
- RAW... is the most “universal” form (sending always possible).
- If code$ is decoded (e.g. NEC:), sending depends on the protocols supported in the firmware (basic often NEC; others can be added).

# IR WAIT

## Sintassi

```
IR WAIT var$
IR WAIT var$ timeout_ms
IR WAIT var$ timeout_ms GOTO line
```

## Descrizione

Waits for an IR signal and saves the result in var$ as a string code$ (serialized code). If a signal arrives: var$ will contain a code like NEC:20DF10EF:32 (protocol recognized) or RAW38:9000,4500,560,... (fallback RAW if not recognized). If the timeout expires:

```
without GOTO → var$="" and the program continues
```

with GOTO line → jump to the indicated line (without error) Parameters var$ → destination string variable (e.g. C$) timeout_ms → timeout in ms (optional)

```
GOTO line → jump on timeout (optional)
```

## Esempi

```
10 IR INIT 15
20 PRINT "Press a button..."
30 IR WAIT C$
40 PRINT "Codice=";C$
```

Wait for a key and print the code

```
10 IR INIT 15
20 IR WAIT C$ 3000
30 IF C$="" THEN PRINT "Timeout" ELSE PRINT "Ricevuto=";C$
```

3s timeout handled in IF

```
10 IR INIT 15
20 PRINT "Press a key (2s)..."
30 IR WAIT C$ 2000 GOTO 100
40 PRINT "Ricevuto=";C$
50 END
100 PRINT "No IR signal"
```

Timeout + skip

## Note

- It is a blocking command: waits until received or times out.
- Many remote controls send “repeat” if you hold down: to avoid double readings use IR HALT after managing the code.
- If you get RAW... it's normal with unrecognized remote controls/protocols: the RAW can still be saved and resent.

# IRDB ADD

## Sintassi

```
IRDB ADD "name" code$
IRDB ADD "name" PRESENT
IRDB ADD "name" PRESENT timeout_ms
IRDB ADD "name" PRESENT timeout_ms GOTO line
```

## Descrizione

Adds or updates a record in the DB in RAM. Direct form: associate "name" → code$ PRESENT form: “learns” a key by reading from the IR receiver and saves the code under that name Parameters "name" → name of the command (e.g. "TV_POWER", "AC_ON") code$ → code string (decoded or RAW) timeout_ms → maximum time to learn (optional)

```
GOTO line → jump if timeout expires (optional)
```

## Esempi

```
10 IRDB INIT "ir.json"
20 IRDB ADD "TV_POWER" "NEC:20DF10EF:32"
30 IRDB SAVE
```

Direct entry

```
10 IR INIT 15
20 IRDB INIT "ir.json"
30 PRINT "Press POWER (5s)..."
40 IRDB ADD "TV_POWER" PRESENT 5000
50 IRDB SAVE
```

Learning with timeout

```
10 IR INIT 15
20 IRDB INIT "ir.json"
30 IRDB ADD "TV_POWER" PRESENT 3000 GOTO 120
40 PRINT "Saved!"
50 IRDB SAVE
60 END
120 PRINT "Timeout: no command saved"
130 END
```

Jump learning on timeout

## Note

- To use PRESENT you must have done IR INIT rxPin.
- The protocol can be RAW... (that's ok: it's often the most reliable).
- Remember: ADD modifies RAM → use IRDB SAVE to save to file.

# IRDB CLEAR

## Sintassi

```
IRDB CLEAR
```

## Descrizione

Empties the database into RAM (does not modify the file until you do IRDB SAVE).

## Esempi

```
10 IRDB INIT "ir.json"
20 IRDB CLEAR
30 IRDB SAVE
```

Clear and save

```
10 IRDB INIT "ir.json"
20 IRDB LOAD
30 IRDB CLEAR
40 PRINT "DB in RAM emptied (file intact)"
```

Empty temporarily (without saving)

## Note

- For permanent deletion: CLEAR + SAVE.
- If you CLEAR and then LOAD, you return to the contents of the file.

# IRDB CODE

## Sintassi

```
IRDB CODE "name" var$
```

## Descrizione

Reads the code associated with "name" from the DB in RAM and puts it in var$.

```
If does not exist, var$="".
```

## Esempi

```
10 IRDB INIT "ir.json"
20 IRDB LOAD
30 IRDB CODE "TV_POWER" C$
40 PRINT "Codice=";C$
```

Read and print

```
10 IR INIT 15 4 38
20 IRDB INIT "ir.json"
30 IRDB LOAD
40 IRDB CODE "TV_POWER" C$
50 IF C$<>"" THEN IR SEND C$
```

Read and send via IR SEND

## Note

- IRDB CODE does not send: IR SEND or IRDB SEND is needed.
- It is useful for “doing logic” on code (printing, comparing, etc.).

# IRDB DELETEFILE

## Sintassi

```
IRDB DELETEFILE
```

## Descrizione

Delete the JSON file from storage (SPIFFS/SD). The database in RAM is not automatically modified.

## Esempi

```
10 IRDB INIT "ir.json"
20 IRDB DELETEFILE
30 PRINT "IRDB file deleted"
```

Clear archive

```
10 IRDB INIT "ir.json"
20 IRDB DELETEFILE
30 IRDB CLEAR
40 IRDB ADD "TV_POWER" "NEC:20DF10EF:32"
50 IRDB SAVE
```

Erase and recreate

## Note

- It is useful when you want to start from scratch and the file is corrupted or you no longer need it.
- If the storage is SD, check that the SD is present.

# IRDB INIT

## Sintassi

```
IRDB INIT "file.json"
IRDB INIT SD "file.json"
```

## Descrizione

Select the JSON file and storage for the IR database. without SD → use SPIFFS with SD → use SD card The DB is kept in RAM during execution, and saved/reloaded from file. Parameters "file.json" → file name (e.g. "ir.json") SD → optional, use SD instead of SPIFFS

## Esempi

```
10 IRDB INIT "ir.json"
20 PRINT "IRDB ready on SPIFFS"
```

DB on SPIFFS

```
10 IRDB INIT SD "ir.json"
20 PRINT "IRDB ready on SD"
```

DB on SD

## Note

- Must be called before IRDB LOAD/SAVE/...
- The chosen storage remains valid until you do a new IRDB INIT.

# IRDB LIST

## Sintassi

```
IRDB LIST
```

## Descrizione

Prints via serial port the list of all commands saved in RAM: name and associated code.

## Esempi

```
10 IRDB INIT "ir.json"
20 IRDB LOAD
30 IRDB LIST
```

List after LOAD

```
Esempio 2 – Lista dopo ADD:
10 IRDB INIT "ir.json"
20 IRDB ADD "TV_VOLUP" "RAW38:9000,4500,..."
30 IRDB LIST
```

List after ADD

## Note

- Useful for debugging and verifying that the DB contains the correct names.
- The output depends on your serial number (style “NAME=… CODE=…”).

# IRDB LOAD

## Sintassi

```
IRDB LOAD
```

## Descrizione

Load database from JSON file (SPIFFS/SD) into RAM. It completely replaces the current content in RAM.

## Esempi

```
10 IRDB INIT "ir.json"
20 IRDB LOAD
30 IRDB LIST
```

Load and list

```
10 IRDB INIT "ir.json"
20 IRDB LOAD
30 PRINT "Records loaded"
```

Load and check

## Note

- If the file does not exist, the DB remains empty (normal behavior).
- If the JSON is corrupt, it may give runtime error (depends on error handling in the firmware).

# IRDB REMOVE

## Sintassi

```
IRDB REMOVE "name"
```

## Descrizione

Removes the record with the name "name" from the DB in RAM.

## Esempi

```
10 IRDB INIT "ir.json"
20 IRDB LOAD
30 IRDB REMOVE "TV_POWER"
40 IRDB SAVE
```

Remove and save

```
10 IRDB INIT "ir.json"
20 IRDB LOAD
30 IRDB LIST
40 IRDB REMOVE "TV_VOLUP"
50 IRDB SAVE
```

Remove after list

## Note

- If the name does not exist, nothing happens (typical “safe” behavior).
- Remember SAVE if you want the removal to be permanent.

# IRDB SAVE

## Sintassi

```
IRDB SAVE
```

## Descrizione

Save the current database in RAM to the JSON file on the selected storage.

## Esempi

```
10 IRDB INIT "ir.json"
20 IRDB ADD "TV_POWER" "NEC:20DF10EF:32"
30 IRDB SAVE
```

After a change

```
10 IRDB INIT "ir.json"
20 IRDB LOAD
30 '... changes ...
40 IRDB SAVE
```

Periodic saving

## Note

- ADD/REMOVE/CLEAR only modifies the RAM: SAVE is needed to make it permanent.
- On SD make sure the SD is initialized/mounted (as per your system setup).

# IRDB SEND

## Sintassi

```
IRDB SEND "name"
IRDB SEND "name" reps
```

## Descrizione

Retrieves the code associated with "name" from the DB in RAM and sends it. reps repeats sending (default 1).

## Esempi

```
10 IR INIT 15 4 38
20 IRDB INIT "ir.json"
30 IRDB LOAD
40 IRDB SEND "TV_POWER"
```

Send saved command

```
10 IR INIT 15 4 38
20 IRDB INIT "ir.json"
30 IRDB LOAD
40 IRDB SEND "TV_POWER" 1
50 WAIT 500
60 IRDB SEND "TV_VOLUP" 2
```

Macro: Send two commands

## Note

- Requires IR INIT with txPin.
- If the record does not exist, it does not send anything (typical behavior).
- If the code is RAW..., sending is always possible; if it is PROTO:..., it depends on the supported protocols.

# KEYPOLL

## Sintassi

```
KEYPOLL ON
KEYPOLL OFF
```

## Descrizione

KEYPOLL enables or disables keyboard polling during program execution. It acts simultaneously on: PS/2 keyboard I²C keyboard CardKB When polling is disabled: the program does not detect ESC keys it does not stop RUN any functions such as CARDKB do not return characters It is useful for maximizing the speed of games or intensive calculations. Behavior KEYPOLL ON Reactivates keyboard polling. Used to return to a menu or accept input. KEYPOLL OFF Disables keyboard reading. Improves performance by reducing CPU load. Interaction with RUN FAST

```
RUN FAST starts automatically with KEYPOLL OFF.
```

The program can reactivate input with KEYPOLL ON.

## Esempi

```
10 KEYPOLL OFF        'maximum speed
20 FOR I = 1 TO 20000
30   'game logic
40 NEXT I
50 END
```

Fast game without keyboard

```
10 KEYPOLL OFF          'fast game
20 GOSUB 500
30 KEYPOLL ON           'I wake up the keyboard
40 GOSUB 900            ' menu
50 END
```

Menu with inputs

```
10 REM Ciclo massimo veloce
20 FOR I = 1 TO 30000: NEXT
30 REM Ora richiedo input → riattivo polling
40 KEYPOLL ON
50 PRINT "Press a button..."
60 K = CARDKB
70 IF K = 0 THEN GOTO 60
80 PRINT "You pressed:", CHR$(K)
```

RUN FAST + selective enable

## Note

- KEYPOLL does not affect BREAKPIN: it remains functional.
- Disabling polling can increase RUN speed by up to 20–30%.
- PS/2 and CardKB are both handled by this command.
- KEYPOLL OFF does not affect functions that read files or sensors.

# KPAVAILABLE

## Sintassi

```
KPAVAILABLE
```

## Descrizione

Returns the number of keys currently in the buffer (0 if empty). Can be used in PRINT, IF, LET.

## Esempi

```
10 IF KPAVAILABLE > 0 THEN KPREAD K$
```

IF

```
10 PRINT KPAVAILABLE
```

PRINT

```
10 LET N = KPAVAILABLE
20 PRINT "BUFFER=";N
```

LET

## Note

- Useful for non-blocking polling.
- Integer value ≥ 0.

# KPFLUSH

## Sintassi

```
KPFLUSH
```

## Descrizione

Empties the key buffer (discards pending events). Useful before reading "clean" input.

## Esempi

```
10 KPFLUSH
20 PRINT "Enter PIN and press #"
30 KPWAIT K$
40 IF K$="#" THEN PRINT "FINE" : END
50 PRINT K$;
60 GOTO 30
```

Cleaning before PIN

```
10 PRINT "Screen 1"
20 KPWAIT _
30 KPFLUSH
40 PRINT "Screen 2"
```

Reset buffer between screens

## Note

- Does not require parameters.
- Does not change the keypad configuration.

# KPINIT (Matrix keypad)

## Sintassi

```
KPINIT nrows ncols r1 r2 ... rn c1 c2 ... cm
```

## Descrizione

Initializes the matrix keypad by specifying the number of rows and columns, then the list of row pins (OUTPUT) and column pins (INPUT_PULLUP), in that order.

## Esempi

```
10 KPINIT 2 2 13 12 14 27
20 PRINT "KEYPAD READY"
```

KEYPAD 2X2

```
10 KPINIT 4 3 2 3 4 5   6 7 8
```

KEYPAD 4X4

```
10 KPINIT 4 4 2 3 4 5   6 7 8 9
```

KEYPAD 4X4

## Note

- Must be called before KPMAP/KPREAD/KPWAIT.
- Rows are OUTPUT (scan), columns INPUT_PULLUP.
- The total number of pins indicated must be nrows + ncols.

# KPMAP

## Sintassi

```
KPMAP "stringaMappa"
```

## Descrizione

Set the keymap in line-by-line order. The length of the string must be nrows * ncols. If not set, position codes such as "R1C2" will be returned.

## Esempi

```
10 KPMAP "ABCD"
```

KEYPAD 2×2 → "ABCD"

```
10 KPMAP "123A456B789C*0#D"
```

KEYPAD 4X4

```
10 KPMAP "123456789*0#"
```

KEYPAD 3X4 numeric

## Note

- The map must have exactly nrows*ncols characters.
- The order is: row 1 (columns 1..n), row 2, …

# KPMODE

## Sintassi

```
KPMODE debounce_ms repeat_delay_ms repeat_rate_ms
```

## Descrizione

```
Configure debounce times, delay before auto-repeat, and auto-repeat cadence (if a key is hel
```

## Esempi

```
10 KPMODE 30 500 150
```

Recommended values

```
10 KPMODE 30 0 0
```

No auto-repeat

```
10 KPMODE 60 500 150
```

More aggressive debounce

## Note

- Works after KPINIT.
- Milliseconds are non-negative integers.

# KPREAD

## Sintassi

```
KPREAD var$
```

## Descrizione

Non-blocking reading: puts the next key in the buffer into the string variable var$ (empty string "" if there is nothing). Compatible with PRINT, IF, LET.

## Esempi

```
10 KPREAD K$
20 IF K$<>"" THEN PRINT K$
30 GOTO 10
```

Continuous poll

```
10 KPINIT 2 2 13 12 14 27
20 KPMAP "ABCD"
30 KPREAD T$
40 IF T$="A" THEN PRINT "LED ON"
```

Use with map

## Note

- var$ must end with $ (string variable).
- Does not lock: if there is no key, returns "".

# KPWAIT

## Sintassi

```
KPWAIT var$
```

## Descrizione

Read blocking: waits until a valid key is pressed and stores it in var$.

## Esempi

```
10 PRINT "Press a button..."
20 KPWAIT K$
30 PRINT "You pressed:";K$
```

Wait and print

```
10 KPWAIT K$
20 IF K$="A" THEN PRINT "LED ON"
30 IF K$="B" THEN PRINT "LED OFF"
40 IF K$="C" THEN PRINT "START"
50 IF K$="D" THEN PRINT "STOP"
60 GOTO 10
```

4-key menu

```
10 KPINIT 4 4 2 3 4 5 6 7 8 9
20 KPMAP "123A456B789C*0#D"
30 KPWAIT K$
40 PRINT "KEY=";K$
```

With 4×4 map

## Note

- Requires KPINIT (and optionally KPMAP).
- Lock until a key with debounce applied arrives.

# LCD BACKLIGHT (solo I²C)

## Sintassi

```
LCD BACKLIGHT ON
LCD BACKLIGHT OFF
```

## Descrizione

Turns the backlight on/off (supported on I²C modules).

## Esempi

```
10 LCD ROW 0 "Backlight OFF in 1s"
20 DELAY 1000
30 LCD BACKLIGHT OFF
40 DELAY 1500
50 LCD BACKLIGHT ON
```

Turns the backlight on/off

## Note

- Not available on parallel; if called it may give error/be ignored (depends on your implementation).

# LCD CLEAR

## Sintassi

```
LCD CLEAR
```

## Descrizione

Clears the entire screen and clears the display memory.

## Esempi

```
10 LCD CLEAR
```

Clear the entire screen

## Note

- After CLEAR the cursor typically returns to (0,0) (depends on the library; in any case you can use LCD HOME).

# LCD CURSOR

## Sintassi

```
LCD CURSOR ON|OFF [BLINK ON|OFF]
```

## Descrizione

Show/hide the cursor and enable/disable blinking.

## Esempi

```
10 LCD ROW 0 "Cursor ON, no blink"
20 LCD CURSOR ON BLINK OFF
30 DELAY 1500
40 LCD ROW 1 "Now BLINK ON"
50 LCD CURSOR ON BLINK ON
60 DELAY 1500
70 LCD CURSOR OFF
```

Show/hide the cursor and enable/disable flashing

## Note

- Map to cursor()/noCursor() and blink()/noBlink() of the library.

# LCD HOME

## Sintassi

```
LCD HOME
```

## Descrizione

Returns the cursor to the (0,0) position without erasing the contents.

## Esempi

```
10 LCD HOME
```

Return the cursor to the (0,0) position

# LCD I2C INIT (16X2 – 16X4)

## Sintassi

```
LCD I2C INIT <addr> [cols rows] [sda scl]
```

## Descrizione

Initializes an I²C LCD (PCF8574). accepts hex without prefix (27, 3F) or with 0x (0x27). Optional: dimensions (cols rows) and SDA SCL pins (for ESP32).

## Esempi

```
10 LCD I2C INIT 27 16 2
20 LCD CLEAR
30 LCD ROW 0 "HELLO I2C"
```

Initializes an LCD and writes HELLO I2C

## Note

- Valid address range: 03..77 (hex).
- On some boards the default I²C pins are already fine; specify SDA SCL only if necessary.

# LCD PAR INIT (16X2 – 16X4)

## Sintassi

```
LCD PAR INIT <rs> <en> <d4> <d5> <d6> <d7> [cols rows]
```

## Descrizione

Initializes an HD44780 compatible parallel LCD. If you don't specify cols rows, use 16 2.

## Esempi

```
10 LCD PAR INIT 12 11 5 4 3 2 16 2
20 LCD CLEAR
30 LCD ROW 0 "HELLO PARALLEL"
```

Initializes a parallel LCD and writes HELLO PARALLEL

## Note

- After init you can use all LCD print/control commands.
- For other formats, set cols rows (e.g. 16 4).

# LCD PRINT

## Sintassi

```
LCD PRINT <testo/expr>
```

## Descrizione

Print from your current location. Accepts strings, variables, and expressions with +.

## Esempi

```
10 A$ = "WORLD"
20 N = 42
30 LCD PRINT "HELLO " + A$
40 LCD PRINT "  N=" + N
```

Print from your current location

## Note

- Does not wrap between lines: to print on another line use LCD SETCUR or LCD ROW.

# LCD ROW

## Sintassi

```
LCD ROW <row> <testo/expr>
```

## Descrizione

Writes the entire line starting from column 0. The text is filled with spaces up to cols and possibly truncated if too long.

## Esempi

```
10 T = 23
20 H = 55
30 LCD ROW 0 "TEMP=" + T + "C"
40 LCD ROW 1 "HUM=" + H + "%"
```

writes TEMP and HUM with the values of the variables

## Note

- Ideal for “clean” updates of an information line.

# LCD SCROLL

## Sintassi

```
LCD SCROLL LEFT [n]
LCD SCROLL RIGHT [n]
```

## Descrizione

Scroll the entire display left or right by n steps (default 1).

## Esempi

```
10 LCD ROW 0 "Scrolling LEFT >>>"
20 DELAY 1000
30 LCD SCROLL LEFT 5
40 DELAY 1000
50 LCD ROW 1 "<<< Scrolling RIGHT"
60 LCD SCROLL RIGHT 5
```

scroll the text

# LCD SETCUR

## Sintassi

```
LCD SETCUR <col> <row>
```

## Descrizione

Place the cursor in column and row .

## Esempi

```
10 LCD SETCUR 2 1
20 LCD PRINT "Qui"
```

Writes Here in column 2 and row 1

## Note

- Out-of-range values are limited to [0..cols-1] and [0..rows-1].

# LEFT$(A$, N)

## Sintassi

```
LEFT$(stringa$, N)
```

## Descrizione

The LEFT$ function returns a substring containing the first N characters of the string A$. If N is greater than the length of the string, the entire string is returned. Useful for: Parsing or cutting strings Input parsing Checking prefixes or codes

## Esempi

```
10 A$ = "BASIC32"
20 PRINT LEFT$(A$, 4)

Expected output:
BASI
```

First 4 characters

```
10 COM$ = "LOAD file.bas"
20 IF LEFT$(COM$, 4) = "LOAD" THEN PRINT "LOAD COMMAND"

Expected output:
LOAD COMMAND
```

Use in IF to recognize a command

```
10 T$ = "ESP"
20 PRINT LEFT$(T$, 10)

Expected output:
ESP
```

Value of N greater than length

```
10 A$ = "TEST"
20 PRINT LEFT$(A$, 0)

Expected output:
(empty string)
```

Substring with N = 0

```
10 RIGA$ = "12345: PRINT 'HELLO'"
20 CODICE$ = LEFT$(RIGA$, 5)
30 PRINT "CODE ="; CODICE$

Expected output:
CODE = 12345
```

Reading of numeric code from the beginning of the line

## Note

- N must be ≥ 0
- Only works with string variables ($)
- Can be combined with RIGHT$, MID$, LEN, ASC, CHR$, etc.

# LEN(A$)

## Sintassi

```
LEN(stringa$)
```

## Descrizione

The LEN function returns the length (in characters) of a string. Counts every character, including spaces, symbols, numbers, letters, etc. It is useful for: Testing user input Checking whether a string is empty Working with substrings

## Esempi

```
10 A$ = "CIAO"
20 PRINT "LENGTH:"; LEN(A$)

Expected output:
LENGTH: 4
```

Length of a string

```
10 T$ = "HELLO WORLD"
20 PRINT LEN(T$)

Expected output:
10
```

String with spaces

```
10 V$ = ""
20 PRINT LEN(V$)

Expected output:
0
```

Empty string

```
10 INPUT "INSERT TEXT:"; T$
20 IF LEN(T$) = 0 THEN PRINT "NO DATA ENTERED"

Expected output:
NO DATA ENTERED
```

Use in IF condition

```
10 MSG$ = "BASIC32"
20 PRINT "NUMBER OF CHARACTERS:"; LEN(MSG$)

Expected output:
NUMBER OF CHARACTERS: 7
```

Character counter

## Note

- LEN only works with string variables ($)
- Does not return errors if the string is empty: returns 0
- Can be combined with LEFT$, RIGHT$, MID$, CHR$, ASC

# LET

## Sintassi

```
LET variabile = espressione
oppure semplicemente:
variabile = espressione
```

## Descrizione

The LET command is used to assign a value to a variable, whether numeric or string ($). It is optional: you can omit it and write the assignment directly (as in most modern BASICs). It can assign: numeric constants strings results of expressions or functions conditional expressions

## Esempi

```
10 LET A = 5
20 PRINT A

Expected output:
5
```

Simple numerical assignment

```
10 B = 10 * 2
20 PRINT B

Expected output:
20
```

Use without LET (short form)

```
10 LET NOME$ = "LUCA"
20 PRINT "CIAO "; NOME$

Expected output:
HELLO LUCA
```

String assignment

```
10 X = SQR(49)
20 PRINT "ROOT:"; X

Expected output:
ROOT: 7
```

With functions

```
10 A = 3
20 IF A < 5 THEN LET RISULTATO = 1 ELSE LET RISULTATO = 0
30 PRINT RISULTATO

Expected output:
1
```

Conditional assignment

## Note

- Arrays cannot be assigned directly (A(1) = ...) except before a DIM
- You can use LET to improve readability, although it is not required

# LIST

## Sintassi

```
LIST
LIST start
LIST start-end
LIST -end
LIST start-
LIST "nomefile.bas"
LIST "nomefile.bas" start
LIST "nomefile.bas" start-end
LIST "nomefile.bas" -end
LIST "nomefile.bas" start-
```

## Descrizione

The LIST command displays: 1. The program currently in memory If a file name is not specified, LIST displays the BASIC listing present in RAM, ordered by line number. Lines generated by INCLUDE are not shown. 2. The contents of a .bas file without loading it If a file name is specified in quotes, the command displays the contents of the file from SPIFFS (or SD if extended), without modifying the current program. In both cases it is possible to limit the output with ranges of lines.

## Esempi

```
LIST
```

View the entire program in memory

```
LIST 200
```

View from line 200 onwards

```
LIST 200-300
```

View lines 200 to 300

```
LIST -300
```

View from the beginning up to line 300

```
LIST 500-
```

View from line 500 to the end

```
LIST "basic.module"

Output di esempio:
10 FUNC TEST
20 PRINT "ok"
30 ENDFUNC
```

View the contents of a file

```
LIST "basic.module" 200-
LIST "basic.module" -300
LIST "basic.module" 100-200
```

View a file by applying a line range

## Note

- LIST without quotes shows the program in RAM.

- LIST "file.bas" displays the contents of the file without loading it.
- Lines from INCLUDE are not shown in the program listing.
- To delete the entire program use NEW.
- The lines are modified by rewriting the same line number.

# LISTVARS

## Sintassi

```
LISTVARS "file"
LISTVARS SPIFFS "file"
LISTVARS SD "file"
```

## Descrizione

LISTVARS prints on the screen all the key = value pairs present in the JSON file.

It is useful for: debugging checking file contents quick checking after SAVEVAR and DELVAR

## Esempi

```
10 SAVEVAR "data.json" "TEMP" 22.5
20 SAVEVAR "data.json" "STATO" "OK"
30 LISTVARS "data.json"

Expected output:
Listing variables in /dati.json:
TEMP = 22.5
STATUS = "OK"
```

View saved variables

## Note

- If the file does not exist, the list is empty (or an empty object is created internally).
- Strings are shown in quotes.
- The order of the keys may not always be the same (depends on the JSON).

# LOAD

## Sintassi

```
LOAD "nomefile"
LOAD variabile$
LOAD SD "nomefile"
LOAD SD variabile$
LOAD SPIFFS "nomefile"
LOAD SPIFFS variabile$
```

## Descrizione

The LOAD command loads a .bas file into program memory, completely overwriting the current listing. If no storage is specified, LOAD reads the file from SPIFFS internal storage. By indicating SD, the file is loaded from the SD card. You can also indicate SPIFFS explicitly, but it is not mandatory. The file name can be specified directly in quotes or via a string variable (e.g. F$). Loading completely replaces the BASIC program currently in memory. Behavior in a BASIC Listing When LOAD is executed within a BASIC program (i.e. with a line number), the behavior changes: The command: Immediately stops the running program. Load the indicated new BASIC file. Start the new program automatically, as if it were run with RUN "filename". In practice, LOAD within a listing acts like a "CHAIN & RUN", replacing the program in memory and launching it immediately.

## Esempi

```
LOAD "main.bas"
```

Load a file from internal memory

```
LOAD SD "menu.bas"
```

Load a file from SD card

```
10 LET F$ = "config.bas"
20 LOAD F$
```

Load from string variable

```
10 LET F$ = "game.bas"
20 LOAD SD F$
```

Load from SD with variable

```
10 PRINT "Starting loader..."
20 LOAD "main.bas"
```

Load and run another program from a listing

## Note

- If the SD card is missing or not mounted, LOAD SD will return an error.
- If the file does not exist, an error message (“File not found” or “Loading failed”) is shown.
- You can use string variables to dynamically specify the file name.
- If LOAD is executed from a BASIC listing, there is no need to add RUN after: execution starts automatically.

# LOADVAR

## Sintassi

```
LOADVAR "file" "chiave" variabile
LOADVAR SPIFFS "file" "chiave" variabile
LOADVAR SD "file" "chiave" variabile
Dove variabile può essere:
Numerica: A, S, TEMP, SPEED
Stringa: A$, U$, NOME$, USER$
```

## Descrizione

LOADVAR reads a key from a JSON file and puts the value into a BASIC variable. If the destination variable ends with $ → it is treated as a string Otherwise → it is treated as a number

## Esempi

```
10 LOADVAR SPIFFS "prefs.json" "SPEED" S
20 LOADVAR SPIFFS "prefs.json" "USER" U$
30 PRINT S
40 PRINT U$

Expected output:
150
Anna
```

Load settings from SPIFFS

```
10 LOADVAR SD "data.json" "TEMP" T
20 PRINT "TEMP="; T
```

Load from SD

## Note

- If the key does not exist in the file → runtime error (key not found).
- If you load a number into a string variable (A$) the value is converted to text.
- If you load a string into a numeric variable (A) the runtime tries to convert it to a number (if not convertible, it often becomes 0).

# LOG(x)

## Sintassi

```
LOG(x)
```

## Descrizione

The LOG(x) function returns the natural logarithm (in base e) of a positive number x. The base e (approximately 2.71828) is the base of natural logarithms commonly used in mathematics and physics.

[[[P]]]LOG(1) = 0 LOG(e) = 1[[[/P]]]

LOG(x) is defined only for x > 0. If x is zero or negative, the result is invalid and may generate an error or undefined value.

## Esempi

```
10 PRINT "LOG(1) ="; LOG(1)

Expected output:
LOG(1) = 0
```

Natural logarithm of 1

```
10 PRINT "LOG(E) ="; LOG(2.71828)

Expected output:
LOG(E) = 1
```

Logarithm of e (approximately 2.71828)

```
10 X = 10
20 PRINT "LOG(10) ="; LOG(X)

Expected output:
LOG(10) = 2.30258
```

Logarithm of a larger value

```
10 INPUT A
20 PRINT "f(A) ="; LOG(A) * A

Expected output (e.g. input 5):
f(A) = 8.047
```

Use in combined formula

## Note

- To calculate base 10 logarithms, you can use:
- LOG10(X) = LOG(X) / LOG(10)

# MAC

## Sintassi

```
MAC      ' funzione numerica
MAC$     ' funzione stringa
```

## Descrizione

The MAC command returns the MAC address of the ESP32 associated with the Wi-Fi interface. MAC → returns the MAC as a number (48 bits in decimal) MAC$ → returns the MAC as a string in the format AA:BB:CC:DD:EE:FF When you assign a string variable, MAC is automatically treated as MAC$:

A$ = MAC ' is equivalent to A$ = MAC$

As with the IP command, it is recommended to use MAC after the WIFI command, so the Wi-Fi is initialized correctly. If Wi-Fi has not been initialized, the value may be 0 / 00:00:00:00:00:00.

## Esempi

```
10 WIFI "ssid" "password"
20 WAIT 3000
30 PRINT MAC$

Expected output (example):
AA:BB:CC:DD:EE:FF
```

Wi-Fi connection and MAC printing in string format

```
10 WIFI "ssid" "password"
20 WAIT 3000
30 M = MAC
40 M$ = MAC$
50 PRINT "NUMERIC MAC ="; M
60 PRINT "MAC STRING ="; M$

Expected output (example):
MAC NUMERIC = 187723572702975
MAC STRING = AA:BB:CC:DD:EE:FF
```

Numeric and string assignment

```
10 WIFI "ssid" "password"
20 WAIT 3000
30 A$ = MAC
40 PRINT "MY MAC IS:"; A$

Expected output:
MY MAC IS: AA:BB:CC:DD:EE:FF
(due to the shortcut, A$ = MAC is interpreted as A$ = MAC$)
```

Assignment with shortcut

## Note

- Requires the Wi-Fi subsystem to be initialized (e.g. via WIFI "ssid" "password"), otherwise it may return:
- MAC = 0
- MAC$ = "00:00:00:00:00:00"

• MAC is useful in numeric form for identifiers, hashing, comparisons.

• MAC$ is ideal for logging, debugging, serial and file printing.

• Compatible with PRINT, LET, IF, numeric and string variables.

# MEMCLEAN

## Sintassi

```
MEMCLEAN <tipo>
Dove <tipo> può essere:
STRING, VARIABLE, ARRAY, HTML, GRAPHICS, STACK, ALL
```

## Descrizione

The MEMCLEAN command frees specific portions of memory by removing data stored at runtime. It is useful for optimizing the use of RAM, preventing slowdowns or freezes during cyclic or animated executions. It can also be called several times during the program.

## Esempi

```
10 MEMCLEAN STRING

Expected output:
All string variables ($) are cleared from memory.
```

Clean up string variables

```
10 MEMCLEAN VARIABLE

Expected output:
All numeric variables are cleared (no longer defined).
```

Clean up all numeric variables

```
10 MEMCLEAN ARRAY

Expected output:
All allocated arrays are freed.
```

Clean defined arrays

```
10 MEMCLEAN HTML

Expected output:
Any stored or buffered HTML pages are deleted.
```

Remove HTML content from memory

```
10 MEMCLEAN GRAPHICS

Expected output:
All sprites are disabled.
Graphic maps (OLEDDATA, ILIDATA) are deleted.
The OLED/ILI display is cleaned.
```

Clean graphics and sprites

```
10 MEMCLEAN STACK

Expected output:
The GOSUB jump stack is emptied. Useful if you suspect overflow or broken recursions.
```

Clean up the GOSUB stack

```
10 MEMCLEAN ALL

Expected output:
```

```
All data in memory is cleared: strings, variables, arrays, HTML, graphics and stacks.
```

Complete cleanup of all user memory

## Note

- Parameters are keywords without quotes: STRING, VARIABLE, ARRAY, HTML, GRAPHICS, STACK, ALL
- The call is case-insensitive (memclean(graphics) is valid)
- Can be used to recover RAM before or after animated loops
- Useful for long or dynamic programs with graphics resources or GOSUB stacks

# MID$(A$, start, len)

## Sintassi

```
MID$(stringa$, inizio, lunghezza)
```

## Descrizione

The MID$ function extracts a substring from A$ starting from the start position (1 = first character), at most character length.

If start exceeds the length of the string, the result is empty. If start + len exceeds the length of the string, only the available part is extracted.

## Esempi

```
10 A$ = "BASIC32"
20 PRINT MID$(A$, 3, 3)

Expected output:
SIC
```

Extract "SIC" from "BASIC32"

```
10 F$ = "SETUP.BAS"
20 PRINT MID$(F$, 6, 4)

Expected output:
.BAS
```

Extract the extension from a file

```
10 S$ = "HELLO"
20 PRINT MID$(S$, 2, 1)

Expected output:
E
```

Extract a single letter

```
10 X$ = "TEST"
20 PRINT MID$(X$, 10, 2)

Expected output:
(empty string)
```

Index beyond length

```
10 T$ = "BENVENUTO"
20 INIZIO = 4
30 LUN = 5
40 PRINT MID$(T$, INIZIO, LUN)

Expected output:
VENUT
```

Use with variables

## Note

- The index starts from 1, not from 0
- The specified length may exceed the end of the string: the output will still be valid

• Can be combined with LEFT$, RIGHT$, LEN, ASC, CHR$, etc.

# MILLIS

## Sintassi

```
MILLIS n
```

## Descrizione

The MILLIS command only suspends the execution of the BASIC program for n milliseconds, but does not block the microcontroller. While waiting, the ESP32 continues to perform all other system operations: WiFi / MQTT web server

```
keyboard or serial input
```

BASIC functions in the background (FUNC ... LOOP) periodic tasks or other internal activities It is therefore a cooperative delay: the BASIC program “stops” momentarily, but the system continues to function normally. It is useful for: Creating animations or flashes without freezing the system Managing time delays while other functions continue to work Synchronizing multiple BASIC routines running in parallel Replacing DELAY more efficiently Difference between DELAY and MILLIS

## Esempi

```
10 PRINT "CIAO"
20 MILLIS 1000
30 PRINT "MONDO"

Expected output:
CIAO
(1 second pause)
```

Non-blocking pause between two messages

```
10 DWRITE 2 1        'Turns on LED
20 MILLIS 500
30 DWRITE 2 0        'Turns off LED
40 MILLIS 500
50 GOTO 10

Expected output:
The LED on GPIO2 flashes every half second,
while the rest of the system (WiFi, BASIC functions in the background, etc.) continues to wo
```

Flashing without blocking the system

```
10 FOR I = 5 TO 1 STEP -1
20 PRINT I
30 MILLIS 1000
40 NEXT I
50 PRINT "GO!"

Expected output:
5
4
3
2
1
GO!
The countdown takes place in real time, but the system does not remain blocked.
```

Countdown with other active activities

```
10 FUNC BLINK
20 DWRITE 2 1
30 MILLIS 250
40 DWRITE 2 0
50 MILLIS 250
60 GOTO 20
70 ENDFUNC
80 FUNC CLOCK
90 PRINT "Millis:", SYS.MILLIS
100 MILLIS 1000
110 GOTO 90
120 ENDFUNC
130 STARTFUNC BLINK
140 STARTFUNC CLOCK
150 END

Effect:
The LED flashes every 250 ms
The time (in milliseconds) is printed every second
Both functions run in parallel, without blocking anything
```

Two processes in parallel (cooperative multitasking)

```
10 PRINT "CARICAMENTO"
20 FOR I=1 TO 10
30 PRINT ".";
40 MILLIS 300
50 NEXT I
60 PRINT "COMPLETED!"

Expected output:
CARICAMENTO.........
COMPLETED!
During the total 3 seconds of waiting, the ESP32 remains operational.
```

Cooperative loading simulation

## Note

- MILLIS measures time in milliseconds (1000 = 1 second)
- Does not block input, network or other BASIC functions
- Can be used inside functions (FUNC) or loops
- When used in the interactive console (outside RUN), it behaves like a traditional DELAY
- It is ideal for multitasking programs or those with active servers

# MQTTAUTOPOLL

## Sintassi

```
MQTTAUTOPOLL ON <intervallo_ms>
MQTTAUTOPOLL OFF
```

## Descrizione

The MQTTAUTOPOLL command enables or disables automatic polling of the MQTT client. When active, the device periodically checks if new MQTT messages have arrived. It is necessary to receive messages in real time without blocking the execution of the program.

## Esempi

```
10 MQTTCONNECT "192.168.1.49" 1883 "" ""
20 MQTTSUB "casa/comandi"
30 MQTTAUTOPOLL ON, 100

Expected output:
The device checks every 100 milliseconds if there are new messages on the home/commands topi
```

Enable polling every 100 ms

```
10 MQTTAUTOPOLL OFF

Expected output:
The device stops automatically checking MQTT messages.
```

Disable automatic polling

## Note

- Polling is essential for automatically receiving messages
- The <ms_interval> parameter is the interval in milliseconds between each check
- Use OFF to disable polling completely
- It should be activated only after MQTTCONNECT and MQTTSUB

# MQTTCONNECT

## Sintassi

```
MQTTCONNECT "broker" porta "user" "password"
```

## Descrizione

The MQTTCONNECT command allows you to connect the device to an MQTT broker (such as Mosquitto or a cloud broker) by specifying the address, port and any access credentials. Once connected, the device can publish and receive MQTT messages.

## Esempi

```
10 WIFI "ssid" "password"
20 MQTTCONNECT "192.168.1.100" 1883 "" ""

Expected output:
Connection established with local MQTT broker at address 192.168.1.100.
```

Connection without authentication

```
10 WIFI "ssid" "password"
20 MQTTCONNECT "mqtt.myserver.com" 1883 "utente1" "passw1"

Expected output:
Connect to the mqtt.mioserver.com broker on port 1883 using your username and password.
```

Connecting to a credentialed broker

## Note

- The standard MQTT port is 1883
- If the broker does not require authentication, use "" for user and password
- Requires an active Wi-Fi connection (WIFI)
- Must be followed by MQTTSUB, MQTTPUBLISH, etc.
- If the connection fails, it is reported in the serial monitor
- Not compatible with MQTT over TLS (port 8883)

# MQTTPUB

## Sintassi

```
MQTTPUB "<topic>" "<messaggio>"
```

## Descrizione

The MQTTPUB command publishes a message to the specified MQTT topic. It allows you to send commands or notifications to external systems, such as other home automation devices or servers.

## Esempi

```
10 MQTTPUB "casa/luci" "ON"

Expected output:
The "ON" message is published on the home/lights topic.
```

Post a message to a topic

```
10 LET M$ = "Temperature OK"
20 MQTTPUB "sistema/stato" M$

Expected output:
Publish the contents of the M$ variable to the system/status topic.
```

Send the contents of a variable

## Note

- Both parameters must be strings
- You must have run MQTTCONNECT first
- Can be used in response to received events or commands
- Useful for communicating with Home Assistant, Node-RED, remote ESP32, etc.

# MQTTSUB

## Sintassi

```
MQTTSUB "<topic>"
```

## Descrizione

The MQTTSUB command subscribes the device to a specific MQTT topic. After subscribing, every message received from that topic will be automatically saved in the MSG$ variable. It can be managed in real time via DO or controlled manually.

## Esempi

```
10 WIFI "ssid" "password"
20 MQTTCONNECT "192.168.1.49" 1883 "" ""
30 MQTTSUB "sistema/stato"
40 MQTTAUTOPOLL ON 100
50 DO 100
100 IF MSG$ <> "" THEN PRINT MSG$
110 LET MSG$ = ""

Expected output:
The MSG$ variable takes the value received from the MQTT mice.
```

Subscribe to a topic and print messages

```
10 PINMODE 2, OUTPUT NOPULL
20 WIFI "ssid" "password"
30 MQTTCONNECT "192.168.1.49" 1883 "" ""
40 MQTTSUB "casa/comandi"
50 MQTTAUTOPOLL ON 100
60 DO 100
70 DO 110
100 IF MSG$ = "ON" THEN DWRITE 2 1
110 IF MSG$ = "OFF" THEN DWRITE 2 0

Expected output:
The LED on pin 2 turns on or off based on the messages received (ON / OFF) on the home/comma
```

Turn an LED on and off by MQTT

## Note

- The topic must be enclosed in quotation marks
- MQTT connection required (MQTTCONNECT)
- MSG$ contains the most recent received message
- Use MQTTAUTOPOLL for continuous reception
- Can be combined with DO or IF

# NEW

## Sintassi

```
NEW
```

## Descrizione

The NEW command deletes the entire program currently in memory, freeing up space to write a new listing. After execution, the program memory is empty, but the variables remain defined until a new RUN or CLR. It does not ask for confirmation: as soon as it is executed, it deletes all the code present.

## Esempi

```
NEW

Output atteso:
Il listato in memoria viene cancellato. Nessuna riga viene mostrata con LIST.
```

Reset the current program

## Note

- NEW does not delete saved files, only the contents of RAM.

# NOTONE

## Sintassi

```
NOTONE pin
```

## Descrizione

Stops the ongoing tone on the specified pin, useful if you want to stop a sound emitted with TONE.

## Esempi

```
10 TONE 26 1000 10000
20 WAIT 2000
30 NOTONE 26
```

Starts a sound for 10 seconds, but stops after 2.

## Note

- If the TONE already has a short duration, NOTONE is not necessary.
- Useful for creating sound effects controlled by external events or logical conditions.

# NOWCLR

## Sintassi

```
NOWCLR
```

## Descrizione

NOWCLR clears the contents of the receive variable set by: NOWINIT recVar$ So, if you did: NOWINIT RX$ then NOWCLR clears only RX$ (or the name you used in NOWINIT). It does not modify the message queue used by NOWPOLL, it only acts on the "historical" variable.

## Esempi

```
10 NOWINIT RX$
20 PRINT "ESP-NOW RECEIVER"
30 PRINT "Local MAC:"; MAC$
40 GOTO 100
100 IF RX$ <> "" THEN PRINT "Received:"; RX$
110 IF RX$ <> "" THEN NOWCLR
120 GOTO 100

Result:
RX$ contains the last message received
after printing it is deleted with NOWCLR, so it is not processed twice
```

Simple reception without queuing

## Note

- NOWCLR is useful in recVar$-based simple mode.
- If you use NOWPOLL, it is not necessary to call NOWCLR: the queue is managed by NOWPOLL and the extracted message is already “consumed”.
- If NOWINIT has never been called, NOWCLR does nothing.

# NOWINIT (ESP-NOW)

## Sintassi

```
NOWINIT recVar$
recVar$ → variabile stringa che conterrà l’ultimo messaggio ricevuto via ESP-NOW
(usata anche da NOWCLR)
```

## Descrizione

NOWINIT initializes ESP-NOW support in your interpreter: sets the Wi-Fi to WIFI_STA mode initializes the ESP-NOW protocol registers the string variable recVar$ as a “historical” receive buffer clears the internal message queue used by NOWPOLL Must be run before using: NOWSEND NOWCLR NOWPOLL If initialization fails, generates a runtime error: ESP-NOW initialization error.

## Esempi

```
10 NOWINIT RX$
20 PRINT "ESP-NOW RECEIVER"
30 PRINT "Local MAC:"; MAC$
```

Initializing a receiver

```
10 NOWINIT TX$
20 PRINT "ESP-NOW TRANSMITTER"
30 PRINT "Local MAC:"; MAC$
```

Initializing a transmitter

## Note

- It must be called once at the beginning of the program (or after NEW).
- Reinitializing with NOWINIT resets the internal NOWPOLL queue.
- recVar$ is updated with the last message received, but the robust management of multiple messages is delegated to NOWPOLL.

# NOWPOLL

## Sintassi

```
NOWPOLL msgVar$ numVar
msgVar$ → variabile stringa in cui viene copiato un messaggio estratto dalla coda
numVar → variabile numerica che indica quanti messaggi restano in coda dopo il prelievo
```

## Descrizione

NOWPOLL reads the internal queue of messages received via ESP-NOW. The queue: is managed by the firmware (you cannot see it directly) has maximum capacity (e.g. 8 messages) is filled by the reception callback every time a message arrives At each call: if the queue contains at least one message: msgVar$ receives the oldest message (FIFO) that message is removed from the queue numVar receives the number of messages still present in the queue if the queue is empty: msgVar$ is set to "" numVar is set to 0 NOWPOLL is the recommended method for reliable reception, especially if the transmitter sends quickly or in bursts.

## Esempi

```
10 NOWINIT RX$
20 PRINT "ESP-NOW RECEIVER"
30 PRINT "MAC:"; MAC$
40 GOTO 100
100 NOWPOLL M$, N
110 IF M$ <> "" THEN PRINT M$, N
120 GOTO 100
With the transmitter:
10 NOWINIT TX$
20 PRINT "ESP-NOW TRANSMITTER"
30 PRINT "MAC:"; MAC$
40 DEST$ = "14:33:5C:0E:9A:B8"
50 N = 0
60 GOTO 100
100 N = N + 1
110 TX$ = "MSG " + STR$(N)
120 NOWSEND DEST$ TX$
130 WAIT 200
140 GOTO 100

You will see on the receiver:
MSG 1 0
MSG 2 0
MSG 3 0
...
Because the receiver is very fast and clears the queue as messages arrive, so they don't acc
```

Simple receiver with tail (works with the TX you already have)

```
10 NOWINIT TX$
20 DEST$ = "14:33:5C:0E:9A:B8"
30 N = 0
40 GOTO 100
100 N = N + 1
110 TX$ = "MSG " + STR$(N)
120 NOWSEND DEST$ TX$
130 WAIT 10
140 GOTO 100
Slowest Receiver:
10 NOWINIT RX$
```

```
20 GOTO 100
100 NOWPOLL M$, N
110 IF M$ <> "" THEN PRINT M$, N
120 WAIT 500
130 GOTO 100

In this case you will be able to see, for example:
MSG 10 3
MSG 11 2
MSG 12 1
MSG 13 0
Here:
M$ is the extracted message
N tells how many messages still remain in the queue after reading M$.
```

Test to see N > 0 (tail filling)

## Note

- NOWPOLL works only after NOWINIT recVar$.
- numVar is not the maximum queue size, but the number of messages currently queued after the POP.
- If you don't care about counting, you can do:
- NOWPOLL M$, N
- IF M$ <> "" THEN PRINT M$
- and ignore N.
- The queue is emptied by:
- NOWINIT (reset)
- the NOWPOLL POP (one at a time)
- NEW (general program reset)

# NOWSEND

## Sintassi

```
NOWSEND "MAC" msg$
NOWSEND macVar$ msg$
"MAC" → stringa con MAC del destinatario nel formato "XX:XX:XX:XX:XX:XX"
macVar$ → variabile stringa che contiene il MAC nello stesso formato
msg$ → stringa da inviare, può essere:
stringa costante "CIAO"
variabile stringa (M$, TX$, MSG$, …)
```

## Descrizione

NOWSEND sends a string message to another device via ESP-NOW. Requires NOWINIT to have been called first. The MAC can be: written directly as a string: "24:62:AB:F2:4D:CC"

```
put into a string variable: DEST$ = "24:62:AB:F2:4D:CC"
```

The message can be constant or variable. If the MAC is invalid, a syntax error is generated: ?SYNTAX ERROR Invalid MAC in NOWSEND If sending fails at runtime: NOWSEND peer error. or NOWSEND send failed.

## Esempi

```
10 NOWINIT TX$
20 PRINT "ESP-NOW TRANSMITTER"
30 PRINT "MAC:"; MAC$
40 DEST$ = "14:33:5C:0E:9A:B8"
50 N = 0
60 GOTO 100
100 N = N + 1
110 TX$ = "MSG " + STR$(N)
120 NOWSEND DEST$ TX$
130 WAIT 200
140 GOTO 100

The device sends MSG 1, MSG 2, MSG 3, … every 200 ms.
DEST$ contains the receiver's MAC.
```

Continuous transmitter (the one you used)

```
10 NOWINIT TX$
20 DEST$ = "24:62:AB:F2:4D:CC"
30 M$ = "HELLO ESP-NOW!"
40 NOWSEND DEST$ M$
```

Single sending with message from variable

## Note

- NOWINIT must have been run before NOWSEND.
- It is good practice to always insert a small WAIT between two NOWSENDs to avoid saturating the radio.
- If the recipient has not initialized ESP-NOW, the message may be lost without notification.

# NRF AVAILABLE

## Sintassi

```
NRF AVAILABLE var
```

## Descrizione

Set var to 1 if there is at least one receiving payload (FIFO RX not empty), otherwise 0. Do not consume the message.

## Esempi

```
10 NRF AVAILABLE A
20 IF A=1 THEN PRINT "There's a message!"
10 NRF START RX
20 NRF AVAILABLE A
30 IF A=0 THEN WAIT 50 : GOTO 20
40 NRF READ M$
50 PRINT "RX: ";M$
```

Set var A to 1 if there is at least one receiving payload

## Note

- Use NRF READ to actually read the payload.
- In continuous RX, it is best to poll with a small WAIT (20–50 ms).

# NRF CONFIG

## Sintassi

```
NRF CONFIG CHANNEL ch
NRF CONFIG DATARATE 250K|1M|2M
NRF CONFIG POWER   MIN|LOW|HIGH|MAX
NRF CONFIG AUTACK  0|1
NRF CONFIG DYNPL   0|1
NRF CONFIG CRCLEN  0|8|16
```

## Descrizione

Modify radio configuration after NRF INIT. CHANNEL (0..125) → frequency 2.4 GHz + ch*1 MHz.

```
DATARATE → speed (250 Kbps = more sensitive/range).
```

POWER → PA level (MAX with PA/LNA modules only if well powered). AUTACK → Auto-Ack/Auto-Retry (enabled by default). DYNPL → dynamic payload (recommended ON). CRCLEN → CRC off/8/16 bit.

## Esempi

```
10 NRF CONFIG DATARATE 250K
20 NRF CONFIG POWER MAX
30 NRF CONFIG CHANNEL 76
```

set configurations

## Note

- After the change, the library immediately applies the parameters.
- CRCLEN 0 disables the CRC (not recommended in noisy environments).

# NRF FLUSH

## Sintassi

```
NRF FLUSH
```

## Descrizione

Empties the module's RX and TX queues (deletes any pending payloads).

## Esempi

```
10 NRF FLUSH
20 PRINT "Clean queues"
```

Clears the RX and TX queues

## Note

- Useful after errors or TX/RX role changes.

# NRF INIT (nRF24L01)

## Sintassi

```
NRF INIT ce csn [channel] [250K|1M|2M] [MIN|LOW|HIGH|MAX] [addrWidth]
```

## Descrizione

Initialize the nRF24L01. ce, csn: GPIO for CE and CSN (SS/CS of the nRF). channel: 0..125 (default 76 → 2.476 GHz). datarate: default 1M. power: default LOW. addrWidth: 3..5 (default 5).

## Esempi

```
10 NRF INIT 4 5
```

Initialize the nRF24L01

```
10 NRF INIT 4 5 76 250K HIGH 5
```

Initialize the nRF24L01

## Note

- Place a 10–47 µF capacitor near VCC of the module (essential with PA/LNA).
- CSN must be high when using other SPI devices (ILI9341, RC522, SD…).

# NRF POWERDOWN

## Sintassi

```
NRF POWERDOWN
```

## Descrizione

Puts the module in standby with very low consumption.

## Esempi

```
10 NRF POWERDOWN
20 WAIT 1000
30 NRF POWERUP
```

Puts the module on standby and then re-enables it

## Note

- After POWERDOWN, call NRF POWERUP before transmitting or listening.

# NRF POWERUP

## Sintassi

```
NRF POWERUP
```

## Descrizione

Returns the module to the active state from low-power.

## Esempi

```
10 NRF POWERUP
20 NRF START RX
```

Return the form to the active state

## Note

- After POWERUP it takes a few ms before TX/RX are ready.

# NRF READ

## Sintassi

```
NRF READ var$ [GOTO line]
```

## Descrizione

If there is a payload, reads it in var$ (max 32 bytes). If there is nothing: with GOTO line → jump to line.

```
without GOTO → set var$="" and continue.
```

## Esempi

```
10 NRF START RX
20 NRF READ M$ 100
30 PRINT "RX: ";M$
40 GOTO 20
100 WAIT 50 : GOTO 20
10 NRF START RX
20 NRF AVAILABLE A
30 IF A=1 THEN NRF READ M$
40 IF M$<>"" THEN PRINT "RX: ";M$
50 WAIT 50 : GOTO 20
```

reads and writes values

## Note

- With DYNPL active, reads exactly the length received.
- If you receive binary data, you may find it unprintable over serial.

# NRF SEND

## Sintassi

```
NRF SEND data$ [GOTO line]
```

## Descrizione

Send data$ (max 32 bytes). If sending fails (no ACK) and GOTO line is present, it jumps.

## Esempi

```
10 NRF SEND "HELLO" 200
20 PRINT "Inviato"
30 GOTO 10
200 PRINT "Sending error"
10 LET MSG$ = "PINGS:" + STR$(TIMER)
20 NRF SEND MSG$
```

Send date$

## Note

- Before sending, make sure you have set NRF SET TXADDR and that the peer has opened an identical RXADDR, with consistent AUTACK (usually 1).
- If you are listening (START RX), the library temporarily switches to TX and then returns to RX.

# NRF SET RXADDR

## Sintassi

```
NRF SET RXADDR pipe "addr"
```

## Descrizione

Opens receive pipe 0..5 at address "addr" (3–5 ASCII characters or 0x... in hexadecimal). If at least one pipe is open, listening starts.

## Esempi

```
10 NRF SET RXADDR 0 "NODE1"
```

Opens receive pipe 0..5 on address "addr"

```
10 NRF SET RXADDR 1 "0xAABBCCDDEE"
```

Opens receive pipe 0..5 on address "addr"

## Note

- nRF addresses are little-endian internally; using 5-character strings is practical and clear (e.g. "NODE1").
- Pipe 0 and 1 can have full address; pipe 2..5 shares the first 4 bytes with pipe1 in the RF24 library (handled internally).

# NRF SET TXADDR

## Sintassi

```
NRF SET TXADDR "addr"
```

## Descrizione

Set the broadcast address (3–5 char or 0x...).

## Esempi

```
10 NRF SET TXADDR "NODE1"
```

Set the broadcast address

```
10 NRF SET TXADDR "0xE7E7E7E7E7"
```

Set the broadcast address

## Note

- To send with ACK, the receiver must have an active identical RXADDR.
- For star topologies: have everyone broadcast to the hub (same TXADDR) and open multiple RXADDRs on the hub for nodes.

# NRF START RX

## Sintassi

```
NRF START RX
```

## Descrizione

Enter listening mode. You will receive payloads on open pipes.

## Esempi

```
10 NRF SET RXADDR 0 "NODE1"
20 NRF START RX
30 NRF READ M$ 100
40 IF M$<>"" THEN PRINT "RX: ";M$
50 GOTO 30
```

Enter listening mode and print M$

## Note

- You can open up to 6 pipes at the same time (0..5).
- If you need to send while listening, NRF SEND suspends RX as needed.

# NRF STOP RX

## Sintassi

```
NRF STOP RX
```

## Descrizione

Exits listening mode (returns ready for TX).

## Esempi

```
10 NRF STOP RX
20 NRF SEND "HELLO"
```

Exits listening mode

## Note

- Not required before every NRF SEND (it is handled), but useful for neat scripts.

# OLED CAMERA CLAMP

## Sintassi

```
OLED CAMERA CLAMP <minX> <minY> <maxX> <maxY>
```

## Descrizione

Set camera limits: after this command, each CAMERA SET (and also the follow) will be blocked within the defined area. minX/minY: minimum limit maxX/maxY: maximum limit

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'Tile 1 full
40 P$=""
50 FOR I=1 TO 256
60   P$=P$+"1"
70 NEXT I
80 OLED TILE NEW 1 16 16 P$ 1
90 OLED TILEMAP NEW 0 20 20 16 16 1 0
100 'I draw a border to understand where I am
110 FOR C=0 TO 19
120   OLED TILEMAP SET 0 C 0 1
130   OLED TILEMAP SET 0 C 19 1
140 NEXT C
150 FOR R=0 TO 19
160   OLED TILEMAP SET 0 0 R 1
170   OLED TILEMAP SET 0 19 R 1
180 NEXT R
190 'World in pixels: 20*16 = 320. Viewport: 128x64.
200 'Valid camera: X 0..192, Y 0..256
210 OLED CAMERA CLAMP 0 0 192 256
220 CX=0: CY=0
230 CX=CX+5
240 CY=CY+3
250 OLED CAMERA SET CX CY
260 OLED TILEMAP DRAW 0 0 0 CAMERA CX CY 1
270 GOTO 230
```

Clamp on a 20x20 map (camera never pops out)

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 P$=""
40 FOR I=1 TO 256
50   P$=P$+"1"
60 NEXT I
70 OLED TILE NEW 1 16 16 P$ 1
80 OLED TILEMAP NEW 0 20 20 16 16 1 0
90 OLED TILEMAP SET 0 19 19 1
100 OLED CAMERA CLAMP 0 0 192 256
110 'I try absurd values: they will be corrected by the clamp
120 CX=999
130 CY=999
140 OLED CAMERA SET CX CY
150 OLED TILEMAP DRAW 0 0 0 CAMERA CX CY 1
```

I try to put the camera "outside": it still gets clamped

### Note

- The clamp remains active until you change it (there is no “disable” in these commands).
- Boundaries are in pixel world.

# OLED CAMERA FOLLOW SPRITE

## Sintassi

```
OLED CAMERA FOLLOW SPRITE <id> [deadW deadH]
```

## Descrizione

Activate “follow”: the camera tries to follow a sprite. : id of the sprite [deadW deadH] (optional): “dead zone”. As long as the sprite moves within this area, the camera does not move.

```
Important note: the follow is applied when the runtime does cameraFollowUpdate() (in your co
```

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'I create a large simple map
40 P$=""
50 FOR I=1 TO 256
60   P$=P$+"1"
70 NEXT I
80 OLED TILE NEW 1 16 16 P$ 1
90 OLED TILEMAP NEW 0 20 20 16 16 1 0
100 FOR C=0 TO 19
110   OLED TILEMAP SET 0 C 10 1
120 NEXT C
130 'Follow sprite 0
140 OLED CAMERA FOLLOW SPRITE 0
150 'In the drawing loop using the global camera (here I pass CX,CY as variables)
160 CX=0: CY=0
170 '(the sprite should move to another part of your program)
180 OLED TILEMAP DRAW 0 0 0 CAMERA CX CY 1
190 GOTO 180

(This example requires that you have already created a sprite elsewhere in the manual; I'm j
```

Follow without dead zone (camera always behind the sprite)

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 P$=""
40 FOR I=1 TO 256
50   P$=P$+"1"
60 NEXT I
70 OLED TILE NEW 1 16 16 P$ 1
80 OLED TILEMAP NEW 0 20 20 16 16 1 0
90 FOR C=0 TO 19
100  OLED TILEMAP SET 0 C 10 1
110 NEXT C
120 'dead zone 32x16 (in pixels): the camera does not "flicker" on small movements
130 OLED CAMERA FOLLOW SPRITE 0 32 16
140 CX=0: CY=0
150 OLED TILEMAP DRAW 0 0 0 CAMERA CX CY 1
160 GOTO 150
```

Follow with dead zone (more stable camera)

## Note

- If the sprite id is invalid, the runtime generates an error.

• For “clean” use, combine FOLLOW with CAMERA CLAMP on large maps.

# OLED CAMERA GET

## Sintassi

```
OLED CAMERA GET SET <xVar> <yVar>
```

## Descrizione

Reads the current position of the global camera and writes it into two numeric variables.

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'Large tile and map
40 P$=""
50 FOR I=1 TO 256
60   P$=P$+"1"
70 NEXT I
80 OLED TILE NEW 1 16 16 P$ 1
90 OLED TILEMAP NEW 0 20 20 16 16 1 0
100 OLED TILEMAP SET 0 0 0 1
110 OLED TILEMAP SET 0 19 19 1
120 CX=0: CY=0
130 CX=CX+3
140 CY=CY+2
150 OLED CAMERA SET CX CY
160 OLED CAMERA GET SET GX GY
170 OLED TILEMAP DRAW 0 0 0 CAMERA GX GY 1
180 GOTO 130
```

I move the camera, then read and use it to draw

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'Tile 1 full marker
40 P$=""
50 FOR I=1 TO 256
60   P$=P$+"1"
70 NEXT I
80 OLED TILE NEW 1 16 16 P$ 1
90 OLED TILEMAP NEW 0 20 20 16 16 1 0
100 CX=0: CY=0
110 OLD_C=-1: OLD_R=-1
120 CX=CX+4
130 IF CX>192 THEN CX=0
140 OLED CAMERA SET CX CY
150 OLED CAMERA GET SET GX GY
160 C=INT(GX/16)
170 R=INT(GY/16)
180 IF OLD_C>=0 THEN OLED TILEMAP SET 0 OLD_C OLD_R 0
190 OLED TILEMAP SET 0 C R 1
200 OLD_C=C: OLD_R=R
210 OLED TILEMAP DRAW 0 0 0 CAMERA GX GY 1
220 GOTO 120

(Here I transform the room into a cell with INT(cam/16).)
```

I show the “camera position” by placing a marker in the corresponding cell

### Note

- GET is useful if you use FOLLOW and want to know “where” the camera really is.
- Variables must be numeric.

# OLED CAMERA SET

## Sintassi

```
OLED CAMERA SET <x> <y>
```

## Descrizione

Set the global camera position in pixel world. The camera is used as an offset for commands that exploit it (e.g. TILEMAP DRAW ... CAMERA camX camY and automatic follow).

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'Tile 1 full
40 P$=""
50 FOR I=1 TO 256
60   P$=P$+"1"
70 NEXT I
80 OLED TILE NEW 1 16 16 P$ 1
90 'Tile 2 chessboard
100 S$=""
110 FOR Y=0 TO 15
120  FOR X=0 TO 15
130    A=X+Y
140    B=A-(INT(A/2)*2)
150    IF B=0 THEN S$=S$+"1"
160    IF B<>0 THEN S$=S$+"0"
170  NEXT X
180 NEXT Y
190 OLED TILE NEW 2 16 16 S$ 1
200 OLED TILEMAP NEW 0 20 20 16 16 1 0
210 FOR R=0 TO 19
220  FOR C=0 TO 19
230    A=C+R
240    B=A-(INT(A/2)*2)
250    IF B=0 THEN OLED TILEMAP SET 0 C R 1
260    IF B<>0 THEN OLED TILEMAP SET 0 C R 2
270  NEXT C
280 NEXT R
290 CX=0: CY=0
300 CX=CX+2
310 IF CX>192 THEN CX=0
320 OLED CAMERA SET CX CY
330 OLED TILEMAP DRAW 0 0 0 CAMERA CX CY 1
340 GOTO 300
```

Manual scrolling on 20×20 map

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 P$=""
40 FOR I=1 TO 256
50   P$=P$+"1"
60 NEXT I
70 OLED TILE NEW 1 16 16 P$ 1
80 OLED TILEMAP NEW 0 20 20 16 16 1 0
90 'two blocks in two distant areas
100 OLED TILEMAP SET 0 1 1 1
```

```
110 OLED TILEMAP SET 0 15 15 1
120 MODE=0
130 IF MODE=0 THEN CX=0
140 IF MODE=0 THEN CY=0
150 IF MODE<>0 THEN CX=160
160 IF MODE<>0 THEN CY=160
170 OLED CAMERA SET CX CY
180 OLED TILEMAP DRAW 0 0 0 CAMERA CX CY 1
190 MODE=1-MODE
200 FOR K=1 TO 25000
210 NEXT K
220 GOTO 130
```

“Camera jump”: jump to two different positions (teleportation effect)

## Note

- CAMERA SET applies any clamp (if you have set CAMERA CLAMP).
- In many examples you can also directly pass CX CY to TILEMAP DRAW without using CAMERA SET: I show it here because it is part of the “camera” section.

# OLED CIRCLE

## Sintassi

```
OLED CIRCLE <x> <y> <raggio> [colore] [0|1]
```

## Descrizione

Draws an unfilled circle centered at , with radius . Optional parameters: [color]: WHITE (default) or BLACK [0|1]: update immediately (1 or absent) or defer (0)

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED CIRCLE 64 32 10 WHITE
```

Circle in the center of the 128x64 screen

```
10 OLED INIT 128 64 3C
20 FOR R = 2 TO 28
30 OLED CLEAR
40 OLED CIRCLE 64 32 R WHITE
50 WAIT 50
60 NEXT R
```

Growing circle animation

## Note

- If you want to erase a previous circle, you can redraw it in BLACK.
- A full circle is available with the OLED FILLCIRCLE command.

# OLED CLEAR

## Sintassi

```
OLED CLEAR
```

## Descrizione

Completely erases the visual content of the OLED display. All pixels are set to black. It does not remove the sprites (which remain in memory), but resets what is shown on the display at that moment.

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
```

Screen cleaning

```
10 OLED INIT 128 64 3C
20 FOR R = 1 TO 16
30 OLED CLEAR
40 OLED CIRCLE 64 16 R
50 WAIT 100
60 NEXT R
```

Cleaning at each animation cycle

## Note

- It is advisable to use it before drawing new elements to avoid unwanted overlaps.
- It does not affect the content of the sprites: if you want to see them again after an OLED CLEAR, you must redraw them with OLED SPRITE DRAW (possibly buffered) and then OLED DRAW.

# OLED DRAW

## Sintassi

```
OLED DRAW
```

## Descrizione

Refreshes the panel with everything that has been drawn in the buffer but not yet shown. It is needed when you use commands with updating disabled (i.e. those where you put 0 as the last parameter): in that case the drawing remains in the internal buffer but is not yet sent to the physical display. OLED DRAW sends everything to the screen in one go (no flicker).

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED LINE 0 0 127 63 WHITE 0
40 OLED LINE 0 63 127 0 WHITE 0
50 OLED DRAW
```

Buffered line

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED TEXT 10 10 1 "READY" WHITE 0
40 OLED RECT 0 0 128 64 WHITE 0
50 OLED DRAW
```

Text, graphics and then a single update

## Note

- If you always use commands without trailing 0 (therefore immediate update), you don't need OLED DRAW.

# OLED DRAWRECT

## Sintassi

```
OLED DRAWRECT x y w h
```

## Descrizione

Updates only a rectangular portion of the OLED display, copying the content already present in the internal graphics buffer onto the screen. The command does not draw anything in the buffer: it only makes changes already made via graphics commands or sprites visible. It is designed to create fluid and fast animations using the dirty rectangle technique, avoiding updating the entire display.

## Esempi

```
10 OLED FILLRECT 0 0 128 64 WHITE 0
20 OLED DRAWRECT 0 0 128 64

Displays a white rectangle on the screen.
```

Refresh the entire display

```
10 OLED FILLRECT 40 20 20 10 WHITE 0
20 OLED DRAWRECT 40 20 20 10

Updates only the indicated portion of the display.
```

Updates only one area of the screen

```
10 OLED INIT 128 64 3C
20 REM ===== INIZIALIZZAZIONE BUFFER =====
30 OLED FILLRECT 0 0 128 64 BLACK 0
40 OLED LINE 0 32 127 32 WHITE 0
50 OLED DRAWRECT 0 0 128 64
60 REM ===== PARAMETRI QUADRATO =====
70 W = 10 : H = 10
80 X = 0 : Y = 28 : V = 2
90 REM ===== LOOP =====
100 OX = X
110 X = X + V
120 IF X <= 0 THEN V = -V
130 IF X >= (128-W) THEN V = -V
140 REM calcolo dirty rectangle
150 XD = OX : IF X < OX THEN XD = X
160 WD = X - OX : IF WD < 0 THEN WD = -WD
170 WD = WD + W
180 REM ripristina sfondo SOLO nella zona sporca
190 OLED FILLRECT XD Y WD H BLACK 0
200 REM ridisegna il pezzo di linea sotto (con coordinate GIUSTE)
210 X2 = XD + WD - 1
220 OLED LINE XD 32 X2 32 WHITE 0
230 REM disegna quadrato nella nuova posizione (buffer)
240 OLED FILLRECT X Y W H WHITE 0
250 REM aggiorna SOLO la zona sporca
260 OLED DRAWRECT XD Y WD H
270 WAIT 20
280 GOTO 100

During movement:
the rectangle passes over the line
the line is correctly redrawn
```

```
no trail and no flicker
```

Dirty rectangle with sprite passing over a line In this example: a horizontal line forms the background a rectangular sprite moves above the line the line is not deleted, because the dirty area is redrawn correctly in the buffer before the DRAWRECT

## Note

- OLED DRAWRECT does not erase or draw objects: it only updates the display.
- It is the program's responsibility to restore the correct contents of the buffer before refreshing.
- In the presence of sprites, the correct flow is:
- clean the dirty area in the buffer
- redraw the background in the area
- redraw the sprite
- call OLED DRAWRECT
- Using OLED DRAWRECT on small areas significantly improves performance compared to OLED DRAW.

# OLED FILLCIRCLE

## Sintassi

```
OLED FILLCIRCLE <x> <y> <raggio> [colore] [0|1]
```

## Descrizione

Draws a solid (filled) circle centered at , with radius . This command colors all pixels inside the circle. Optional parameters: [color]: WHITE (default) or BLACK WHITE fills the circle in white BLACK fills in black and therefore can be used to erase a circular area [0|1]: refresh the screen immediately or not

```
1 (or absent): Draw and update the display immediately
0: draw only in buffer; to view you must then call OLED DRAW
```

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED FILLCIRCLE 64 32 8 WHITE

Result: solid white circle with radius 8 in the center of the display.
```

Solid white dot in the center of the screen

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED FILLRECT 0 0 128 64 WHITE 0
40 OLED FILLCIRCLE 30 30 10 BLACK 0
50 OLED DRAW
```

“Black bubble” without immediate update

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED FILLCIRCLE 64 32 12 WHITE 0
40 OLED FILLCIRCLE 64 32 8 BLACK 0
50 OLED FILLCIRCLE 64 32 4 WHITE 0
60 OLED DRAW
```

“Target” effect (full concentric circles)

## Note

- It is different from OLED CIRCLE, which only draws the edge of the circle.
- With BLACK you can “pierce” or erase rounded areas within already drawn graphics.
- With update 0 you can prepare complex graphics and then do OLED DRAW to show them in one go (without flicker).

# OLED FILLRECT

## Sintassi

```
OLED FILLRECT <x> <y> <larghezza> <altezza> [colore] [0|1]
```

## Descrizione

Draws a filled rectangle at the given coordinates. It is used to fill areas, make panels, or erase areas. Optional parameters: [color]: WHITE or BLACK (default: WHITE) [0|1]: update immediately (1 or absent) or after (0)

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED FILLRECT 0 56 128 8 WHITE
```

White bar

```
10 OLED FILLRECT 0 56 128 8 BLACK 0
20 OLED DRAW
```

Delete a zone but don't update yet

## Note

- By drawing BLACK you cover and erase what was underneath.
- Very useful for black status bars, frames, text boxes, etc.
- Used with update=0 it is perfect for building an entire invisible screen and then making it visible all together with OLED DRAW.

# OLED INIT (DISPLAY SSD1306-SSD1309)

## Sintassi

```
OLED INIT <larghezza> <altezza> <indirizzo_hex> [sda scl]
```

## Descrizione

Initialize the SSD1306 OLED display via I2C. After this command the display is prepared, deleted and made ready for use. Parameters: : width in pixels (typical: 128) : height in pixels (typical: 64) : I2C address of the display (usually 3C) you can write it as 3C or 0x3C [sda scl] (optional): SDA and SCL pins for the I2C

```
ESP32 default: SDA=21, SCL=22
```

This command must be called before using any other OLED commands.

To view SSD1309 I2C on the display you need to connect the reset pin to a GPIO of the ESP32 and set the GPIO to HIGH

## Esempi

```
10 OLED INIT 128 64 3C 21 22
```

Initialization by forcing the I2C pins

```
10 OLED INIT 128 64 3C 21 22
```

Initialization by forcing the I2C pins

```
10  PINMODE 16 OUTPUT NOPULL■
20  DWRITE 16 1■
30  OLED INIT 128 64 3C■
40  OLED TEXT 20 8 2 "HELLO" WHITE
```

Initializing OLED SSD1309

## Note

- After initialization, the firmware enables OLED, then all other controls become active.
- The most common address for SSD1306 is 0x3C.
- The display is cleaned automatically.

# OLED INVERT ON / OFF

## Sintassi

```
OLED INVERT ON
OLED INVERT OFF
```

## Descrizione

Enable or disable the inverted mode of the OLED display: ON: white background, black text OFF: black background, white text (standard)

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED INVERT ON
30 OLED TEXT 10 10 1 "INVERTITO"
40 DELAY 1000
50 OLED INVERT OFF
```

inverts the color of the display

## Note

- Does not clear the display contents.
- It is useful for temporary effects or visual focus on an area.

# OLED LINE

## Sintassi

```
OLED LINE <x1> <y1> <x2> <y2> [colore] [0|1]
```

## Descrizione

Draw a line from point , to point ,. Optional parameters: [color]: WHITE (default) or BLACK [0|1]: update now (1/default) or postpone (0)

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED LINE 0 0 127 63
```

Diagonal on the screen

```
10 OLED INIT 128 64 3C
20 OLED LINE 0 32 127 32 WHITE 0
30 OLED DRAW
```

Buffered white horizontal line

## Note

- With BLACK you can erase a previous white line in the same position.
- You can draw multiple lines with update 0 and then call OLED DRAW.

# OLED PIXEL

## Sintassi

```
OLED PIXEL <x> <y> [colore] [0|1]
```

## Descrizione

Draws a single pixel in , coordinates. Optional parameters: [color]: WHITE or BLACK (default: WHITE) [0|1]: refresh the screen immediately?

```
1 (or absent): update immediately
```

0: does not update yet (OLED DRAW is needed later)

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED PIXEL 10 10
```

White pixel with instant update

```
10 OLED INIT 128 64 3C
20 OLED PIXEL 20 20 WHITE 0
30 OLED DRAW
```

White pixel without updating immediately

## Note

- With BLACK you can “turn off” a pixel.
- If <x> or <y> are off-screen, the command doesn't draw anything.

# OLED RECT

## Sintassi

```
OLED RECT <x> <y> <larghezza> <altezza> [colore] [0|1]
```

## Descrizione

Draws an empty rectangle (border only) with top left corner at ,, width , and height . Optional parameters: [color]: WHITE (default) or BLACK [0|1]: immediate or delayed update

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED RECT 0 0 128 64
```

Edge of the screen

```
10 OLED RECT 0 0 128 64 BLACK 1
```

Black rectangle (to erase a previous border)

## Note

- OLED RECT draws only the outline. For a filled rectangle use OLED FILLRECT.
- You can combine multiple ... 0 and then OLED DRAW.

# OLED SPRITE CLEAR

## Sintassi

```
OLED SPRITE CLEAR
```

## Descrizione

Removes all active sprites from memory. After this command, no sprites will be drawn until they are recreated with OLED SPRITE NEW.

## Esempi

```
10 OLED SPRITE CLEAR
```

Cleans all sprites

## Note

- The command removes sprites, but does not automatically update the display.
- If you also want to clean the screen, use OLED CLEAR and/or redraw the scene and then do OLED DRAW.

# OLED SPRITE COLLISION

## Sintassi

```
OLED SPRITE COLLISION id1 TO id2 GOTO line
OLED SPRITE COLLISION id1 TO ANY GOTO line
OLED SPRITE COLLISION id1 TO id2 SET variabile
OLED SPRITE COLLISION id1 TO ANY SET variabile
```

## Descrizione

Checks for rectangular collision (AABB) between sprites drawn on the OLED. In case of collision:

```
with GOTO jumps to the indicated line;
```

with SET variable assigns 1 (collision) or 0 (no collision) to a variable (numeric or string $).

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED SPRITE NEW 1 RECT 10 30 8 8
40 OLED SPRITE NEW 2 RECT 80 30 8 8
50 OLED SPRITE DRAW
51 X = 10
52 Y = 30
53 X = X+1
60 OLED SPRITE MOVE 1 X Y
65 OLED SPRITE DRAW
66 OLED SPRITE COLLISION 1 TO 2 GOTO 200
70 GOTO 53
200 PRINT "COLLISIONE"
210 END
```

GOTO (rectangular)

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED SPRITE NEW 1 RECT 10 30 8 8
40 OLED SPRITE NEW 2 RECT 80 30 8 8
50 OLED SPRITE DRAW
51 X = 10
52 Y = 30
53 X = X+1
60 OLED SPRITE MOVE 1 X Y
65 OLED SPRITE DRAW
66 OLED SPRITE COLLISION 1 TO 2 SET C
67 IF C THEN PRINT "Hit!" ELSE PRINT "Libero"
70 GOTO 53
```

SET variable (rectangular)

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED SPRITE NEW 1 RECT 10 30 8 8
31 OLED SPRITE NEW 2 RECT 80 30 8 8
32 OLED SPRITE NEW 3 RECT 100 10 8 8
50 OLED SPRITE DRAW
51 X = 10
52 Y = 30
53 X = X+1
60 OLED SPRITE MOVE 1 X Y
```

```
65 OLED SPRITE DRAW
66 OLED SPRITE COLLISION 1 TO ANY SET HIT
67 IF HIT THEN PRINT "1 hit someone!"
70 GOTO 53
```

ANY with SET (rectangular)

## Note

- IDs must refer to active sprites (OLED SPRITE NEW).
- If the variable ends with $, it receives "1"/"0"; otherwise 1.0/0.0.
- AABB is very fast; on 128x64 the impact is minimal.
- For stable performance with many sprites, limit tests per cycle.

# OLED SPRITE COLLISIONC

## Sintassi

```
OLED SPRITE COLLISIONC id1 TO id2 GOTO line
OLED SPRITE COLLISIONC id1 TO ANY GOTO line
OLED SPRITE COLLISIONC id1 TO id2 SET variabile
OLED SPRITE COLLISIONC id1 TO ANY SET variabile
```

## Descrizione

Check for circular collision using the circles inscribed in the sprite bounding boxes:

```
center = center bbox, radius = min(width, height)/2.
```

Useful for round sprites or for a "softer" detection.

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED SPRITE NEW 1 CIRCLE 10 30 8
40 OLED SPRITE NEW 2 CIRCLE 80 30 8
50 OLED SPRITE DRAW
51 X = 10
52 Y = 30
53 X = X+1
60 OLED SPRITE MOVE 1 X Y
65 OLED SPRITE DRAW
66 OLED SPRITE COLLISIONC 1 TO 2 GOTO 200
70 GOTO 53
200 PRINT "CIRCULAR COLLISION"
210 END
```

GOTO (circular)

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED SPRITE NEW 1 CIRCLE 10 30 8
40 OLED SPRITE NEW 2 CIRCLE 80 30 8
50 OLED SPRITE DRAW
51 X = 10
52 Y = 30
53 X = X+1
60 OLED SPRITE MOVE 1 X Y
65 OLED SPRITE DRAW
66 OLED SPRITE COLLISIONC 1 TO 2 SET CIRC
67 IF CIRC THEN PRINT "Contact (circles)!" ELSE PRINT "No contact"
70 GOTO 53
```

SET variable (circular)

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED SPRITE NEW 1 CIRCLE 10 30 8
31 OLED SPRITE NEW 2 CIRCLE 80 30 8
32 OLED SPRITE NEW 3 CIRCLE 100 10 8
50 OLED SPRITE DRAW
51 X = 10
52 Y = 30
53 X = X+1
60 OLED SPRITE MOVE 1 X Y
```

```
65 OLED SPRITE DRAW
66 OLED SPRITE COLLISIONC 1 TO ANY SET HIT
67 IF HIT THEN PRINT "1 touched at least one (circular)"
70 GOTO 53
```

ANY with SET (circular)

## Note

- Active sprites and valid bbox required.
- String variables (VAR$) → "1"/"0"; numeric (VAR) → 1/0.
- COLLISIONC is slightly more expensive than COLLISION, but on OLED the impact is negligible.
- Maintain the order of the branches in the parser: first TO ANY SET, then SET, finally GOTO.

# OLED SPRITE DELETE

## Sintassi

```
OLED SPRITE DELETE <id>
```

## Descrizione

Disables and removes the sprite with identifier .

## Esempi

```
10 OLED INIT 128 32 3C
20 OLED SPRITE NEW 0 RECT 10 10 40 10
30 OLED SPRITE SHOW 0
40 OLED SPRITE DRAW
50 DELAY 1000
60 OLED SPRITE DELETE 0
70 OLED SPRITE DRAW
```

removes the sprite with id 0

## Note

- The sprite is deleted and will not be drawn again until it is recreated.
- The id must be valid (0 to MAX_SPRITES-1).

# OLED SPRITE DRAW

## Sintassi

```
OLED SPRITE DRAW
OLED SPRITE DRAW id
OLED SPRITE DRAW ALL 0
OLED SPRITE DRAW id 0
```

## Descrizione

Draw OLED sprites in the internal framebuffer or directly on the display, depending on the update mode. The command can: draw all the active and visible sprites or just one specific sprite with immediate updating (default) or without immediately updating the display (buffered mode). It must be called after any modification to the sprites (creation, movement, change of content, bitmap, text, etc.). Update mode: To make the buffered content visible it is necessary to call: OLED DRAW or, to update only one area: OLED DRAWRECT x y w h

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED SPRITE NEW 0 RECT 10 10 40 40
30 OLED SPRITE SHOW 0
40 OLED SPRITE DRAW
```

Immediate drawing (classic behavior)

```
10 OLED INIT 128 64 3C
20 OLED SPRITE NEW 0 RECT 10 10 40 40
30 OLED SPRITE SHOW 0
40 OLED SPRITE DRAW ALL 0
50 OLED DRAW
```

Buffered drawing (no flicker)

```
10 OLED INIT 128 64 3C
20 OLED SPRITE NEW 0 RECT 0 28 10 10
30 OLED SPRITE SHOW 0
40 X = 0
50 V = 2
60 X = X + V
70 IF X <= 0 THEN V = -V
80 IF X >= 118 THEN V = -V
90 OLED SPRITE MOVE 0 X 28
100 OLED FILLRECT 0 0 128 64 BLACK 0
110 OLED SPRITE DRAW ALL 0
120 OLED DRAW
130 WAIT 20
140 GOTO 60
```

Smooth animation without flicker

```
10 OLED SPRITE DRAW ALL 0
20 OLED DRAWRECT 40 20 20 20
```

Advanced use with OLED DRAWRECT (dirty-rect)

## Note

- If used without 0, OLED SPRITE DRAW immediately updates the display (may cause flicker if called multiple times in the same frame).

• If used with 0, drawing occurs only in the buffer; to view it you need OLED DRAW or OLED DRAWRECT.

• For smooth animations: always use OLED SPRITE DRAW ALL 0 and then only one OLED DRAW (or OLED DRAWRECT) per frame.

• The sprite drawing order depends on the ID: highest ID = top.

# OLED SPRITE DRAWWORLD

## Sintassi

```
OLED SPRITE DRAWWORLD ALL CAMERA <camX> <camY> [drawNow]
OLED SPRITE DRAWWORLD <id>  CAMERA <camX> <camY> [drawNow]
```

## Descrizione

Draw sprites "in world coordinates", i.e. applying the camera offset on the fly: If you use ALL, draw all the sprites used. If you pass , it just draws a sprite. optional drawNow:

```
1 = update display immediately
0 = draw in the buffer (useful if you then do something else before refreshing)
```

Note: This command temporarily moves the -camX/-camY sprites for drawing only and then puts them back.

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 CX=0: CY=0
40 CX=CX+2
50 IF CX>192 THEN CX=0
60 OLED SPRITE DRAWWORLD 0 CAMERA CX CY 1
70 GOTO 40

(Assumes sprite 0 has already been created and placed in your program.)
```

Draw a sprite with a flowing camera

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 CX=0: CY=0
40 CX=CX+1
50 CY=CY+1
60 IF CX>192 THEN CX=0
70 IF CY>256 THEN CY=0
80 OLED SPRITE DRAWWORLD ALL CAMERA CX CY 1
90 GOTO 40
```

Draw all sprites in world (ALL)

## Note

- If there are no “used” sprites, you won't see anything (this is not an error: there is simply nothing to draw).
- In a typical game: first TILEMAP DRAW ..., then SPRITE DRAWWORLD ... with the same camera.

# OLED SPRITE HIDE

## Sintassi

```
OLED SPRITE HIDE <id>
```

## Descrizione

Temporarily hides the sprite with id without deleting it.

## Esempi

```
10 OLED INIT 128 32 3C
20 OLED SPRITE NEW 0 RECT 10 10 40 10
30 OLED SPRITE SHOW 0
40 OLED SPRITE DRAW
50 DELAY 1000
60 OLED SPRITE HIDE 0
70 OLED SPRITE DRAW
```

Hide sprite 0

## Note

- The sprite still exists but is not drawn.

# OLED SPRITE MOVE

## Sintassi

```
OLED SPRITE MOVE <id> <x> <y>
```

## Descrizione

Move the sprite identified by to position , .

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED SPRITE NEW 0 FRAME 0 10 40 10
30 OLED SPRITE SHOW 0
40 FOR X = 0 TO 80 STEP 10
50 OLED SPRITE MOVE 0 X 10
60 OLED SPRITE DRAW
70 DELAY 100
80 NEXT X
```

Sprite 0 will be moved

## Note

- To make the move effective, call OLED SPRITE DRAW.

# OLED SPRITE NEW

## Sintassi

```
OLED SPRITE NEW id RECT x y larghezza altezza
OLED SPRITE NEW id FRAME x y larghezza altezza
OLED SPRITE NEW id CIRCLE x y raggio
OLED SPRITE NEW id LINE x y dx dy
OLED SPRITE NEW id TEXT x y size "testo"
OLED SPRITE NEW id CHAR x y "carattere" size
OLED SPRITE NEW id NUM x y numero size
OLED SPRITE NEW id BITMAP x y larghezza altezza pattern scala
```

## Descrizione

The OLED SPRITE NEW command allows you to create independent graphic objects (sprites) to display on a monochrome OLED display. A sprite is a graphic element that can be positioned on the screen via X and Y coordinates, identified by a unique ID and defined by a specific type. Once created, the sprite can be activated, displayed individually or rendered together with other sprites. Main features With OLED SPRITE NEW it is possible to: Create geometric shapes (filled rectangles, frames, filled circles, lines) Display scalable text Show enlarged single characters Display whole or real numbers Draw customized monochrome images via bitmap Position each element anywhere on the display Manage multiple graphic objects simultaneously via numerical identifiers Operational characteristics Each sprite is independent of the others. Sprites do not automatically appear upon creation. Viewing occurs via separate activation and rendering commands. The objects are monochromatic. You can use numeric and string variables in parameters. Bitmaps can be scaled to enlarge the original image.

## Esempi

```
10 OLED SPRITE NEW 0 RECT 10 10 30 8
```

Draw a filled rectangle

```
10 OLED SPRITE NEW 1 FRAME 50 8 30 8
```

Draw the outline of a rectangle

```
10 OLED SPRITE NEW 2 CIRCLE 90 16 6
```

Draw a filled circle with center at (x,y)

```
10 OLED SPRITE NEW 3 LINE 0 0 30 16
```

Draw a line from point (x,y) to point (x+dx, y+dy)

```
10 OLED SPRITE NEW 4 TEXT 0 24 1 "HELLO"
```

Shows a text string in scalable font

```
10 OLED SPRITE NEW 5 CHAR 90 24 "A" 2
```

Show a single character

```
10 OLED SPRITE NEW 6 NUM 110 0 2024 1
```

Show a number

```
10 OLED INIT 128 64 3C
20 OLED SPRITE NEW 0 RECT 10 10 30 8
30 OLED SPRITE NEW 1 FRAME 50 8 30 8
40 OLED SPRITE NEW 2 CIRCLE 90 16 6
50 OLED SPRITE NEW 3 LINE 0 0 30 16
```

```
60 OLED SPRITE NEW 4 TEXT 0 24 1 "HELLO"
70 OLED SPRITE NEW 5 CHAR 90 24 "A" 2
80 OLED SPRITE NEW 6 NUM 110 0 2024 1
90 P$ = "0110" + "1001" + "1001" + "0110"
100 OLED SPRITE NEW 7 BITMAP 40 40 4 4 P$ 4
110 FOR I = 0 TO 7
120   OLED SPRITE SHOW I
130 NEXT I
140 OLED SPRITE DRAW
150 END
```

Draws a small monochrome image based on a pattern string (0/1 or digits/pixel). Example patterns P$ = "0110" + "1001" + "1001" + "0110" (4×4 pixels)

## Note

- OLED sprites are monochrome.
- OLED SPRITE SHOW id enables individual sprites.
- OLED SPRITE DRAW renders all active and visible sprites.
- Numeric and string variables are supported.
- Bitmaps use concatenated patterns without spaces.

# OLED SPRITE SETBITMAP

## Sintassi

```
OLED SPRITE SETBITMAP id pattern$
```

## Descrizione

Change the graphic pattern of an already created BITMAP sprite. It is used for animations (walking, movement, snake sprites, objects that change shape...).

## Esempi

```
10 P1$ = "01110" + "10001" + "11111" + "10001" + "01110"
20 P2$ = "01110" + "11011" + "11111" + "11011" + "01110"
30 OLED SPRITE NEW 0 BITMAP 40 40 5 5 P1$ 4
40 FOR I = 1 TO 10
50   OLED SPRITE SETBITMAP 0 P1$
60   OLED SPRITE DRAW 0
70   DELAY 100
80   OLED SPRITE SETBITMAP 0 P2$
90  OLED SPRITE DRAW 0
100  DELAY 100
110 NEXT I
```

Simple animation (2 frames)

## Note

- The sprite maintains: position, scale, size, visibility.
- Only changes the pattern.
- The new pattern must have the same W×H dimension as the original sprite.
- To see the change you need an OLED SPRITE DRAW id.

# OLED SPRITE SETCHAR

## Sintassi

```
OLED SPRITE SETCHAR <id> "X"
```

## Descrizione

Modifies the character shown by a sprite of type CHAR.

## Esempi

```
10 OLED INIT 128 32 3C
20 OLED SPRITE NEW 2 CHAR 100 0 "A" 1
30 OLED SPRITE SHOW 2
40 OLED SPRITE DRAW
50 DELAY 1000
60 OLED SPRITE SETCHAR 2 "B"
70 OLED SPRITE DRAW
```

Change character from A to B

# OLED SPRITE SETNUM

## Sintassi

```
OLED SPRITE SETNUM <id> <numero>
```

## Descrizione

Updates the number displayed by a sprite of type NUM.

## Esempi

```
10 OLED INIT 128 32 3C
20 OLED SPRITE NEW 1 NUM 60 0 123 1
30 OLED SPRITE SHOW 1
40 OLED SPRITE DRAW
50 DELAY 1000
60 OLED SPRITE SETNUM 1 2024
70 OLED SPRITE DRAW
```

Change number from 123 to 2024

# OLED SPRITE SETTEXT

## Sintassi

```
OLED SPRITE SETTEXT <id> "nuovo testo"
```

## Descrizione

Modifies the textual content of the TEXT sprite.

## Esempi

```
10 OLED INIT 128 32 3C
20 OLED SPRITE NEW 0 TEXT 0 0 1 "HELLO"
30 OLED SPRITE SHOW 0
40 OLED SPRITE DRAW
50 DELAY 1000
60 OLED SPRITE SETTEXT 0 "WORLD"
70 OLED SPRITE DRAW
```

change the text from HELLO to WORLD

# OLED SPRITE SHOW

## Sintassi

```
OLED SPRITE SHOW <id>
```

## Descrizione

Makes a previously hidden or created sprite visible.

## Esempi

```
10 OLED INIT 128 32 3C
20 OLED SPRITE NEW 0 CIRCLE 20 16 6
30 OLED SPRITE SHOW 0
40 OLED SPRITE DRAW
```

Makes sprite 0 visible

# OLED TEXT

## Sintassi

```
OLED TEXT <x> <y> <dimensione> <testo/expr> [colore] [0|1]
```

## Descrizione

Writes text on the display at position , with font size . The parameter can be: a string in quotes OLED TEXT 0 0 1 "HELLO" a string variable OLED TEXT 0 16 1 A$ a concatenation with +

```
OLED TEXT 0 0 1 "TEMP=" + T + "C"
```

a string function of BASIC OLED TEXT 0 0 1 LEFT$(A$,3) a number or numeric expression (is converted to a string automatically) Final optional parameters: [color]: WHITE or BLACK (default: WHITE) [0|1]: update 1 (or absent), or buffer 0

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED TEXT 20 8 2 "HELLO" WHITE
```

Large text

```
10 T = 22
20 MSG$ = "READY"
30 OLED TEXT 0 0 1 "TEMP=" + T + "C" WHITE 0
40 OLED TEXT 0 16 1 MSG$ WHITE 0
50 OLED DRAW
```

Text constructed with variables and delayed updating

```
10 OLED TEXT 0 32 1 "LOADING..." WHITE
20 WAIT 1000
30 OLED FILLRECT 0 32 80 10 BLACK
40 OLED TEXT 0 32 1 "DONE" WHITE
```

Write and erase

## Note

- size affects character width and height (1 = normal, 2 = enlarged).
- With update=0 you can write multiple lines of text to the buffer and display them all at once with OLED DRAW.

# OLED TILE ANIM NEW

## Sintassi

```
OLED TILE ANIM NEW <animId> <frames> <tileId...> MS <msPerFrame>
```

## Descrizione

Create a tile animation. An animation is a list of tiles that automatically change over time.

```
 = total number of animation frames (max 8)
 = list of tiles used as frames (you can pass fewer: the last one is repeated)
MS  = duration of each frame in milliseconds
```

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'Tile 1: full
40 A$=""
50 FOR I=1 TO 256
60   A$=A$+"1"
70 NEXT I
80 OLED TILE NEW 1 16 16 A$ 1
90 'Tile 2: chessboard
100 B$=""
110 FOR Y=0 TO 15
120   FOR X=0 TO 15
130     S=X+Y
140     T=S-(INT(S/2)*2)
150     IF T=0 THEN B$=B$+"1"
160     IF T<>0 THEN B$=B$+"0"
170   NEXT X
180 NEXT Y
190 OLED TILE NEW 2 16 16 B$ 1
200 'Anim 0: 2 frames, changes every 200ms
210 OLED TILE ANIM NEW 0 2 1 2 MS 200
220 OLED TILEMAP NEW 0 8 4 16 16 1 0
230 OLED TILEMAP ANIMFILL 0 0
240 ' loop
250 OLED TILE ANIM TICK 16
260 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
270 GOTO 250
```

2 frames (full ↔ board) across the entire map

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'Tile 1: full
40 T1$="": FOR I=1 TO 256: T1$=T1$+"1": NEXT I
50 '(I avoid ":" -> I rewrite safely)
60 T1$=""
70 FOR I=1 TO 256
80   T1$=T1$+"1"
90 NEXT I
100 OLED TILE NEW 1 16 16 T1$ 1
110 'Tile 2: diagonal
120 T2$=""
130 FOR Y=0 TO 15
140  FOR X=0 TO 15
150    IF X=Y THEN T2$=T2$+"1"
```

```
160    IF X<>Y THEN T2$=T2$+"0"
170  NEXT X
180 NEXT Y
190 OLED TILE NEW 2 16 16 T2$ 1
200 'Tile 3: anti-diagonal
210 T3$=""
220 FOR Y=0 TO 15
230  FOR X=0 TO 15
240    IF X=15-Y THEN T3$=T3$+"1"
250    IF X<>15-Y THEN T3$=T3$+"0"
260  NEXT X
270 NEXT Y
280 OLED TILE NEW 3 16 16 T3$ 1
290 'Tile 4: empty (all 0)
300 T4$=""
310 FOR I=1 TO 256
320   T4$=T4$+"0"
330 NEXT I
340 OLED TILE NEW 4 16 16 T4$ 1
350 OLED TILE ANIM NEW 1 4 1 2 3 4 MS 120
360 OLED TILEMAP NEW 0 8 4 16 16 1 0
370 OLED TILEMAP SET 0 3 1 1
380 OLED TILEMAP ANIM 0 3 1 1
390 OLED TILE ANIM TICK 16
400 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
410 GOTO 390
```

4 “blink” frames in a single cell

## Note

- Max frames supported: 8.
- If <msPerFrame> is 0 it is forced to 1.
- The animation “exists” even if you don't use it in a tilemap yet.

# OLED TILE ANIM TICK

## Sintassi

```
OLED TILE ANIM TICK <msDelta>
```

## Descrizione

Advances all tile animations by the indicated time in milliseconds. It must be called in the loop to make them move.

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 A$="": FOR I=1 TO 256: A$=A$+"1": NEXT I
40 'I'll rewrite without ":" to be safe
50 A$=""
60 FOR I=1 TO 256
70   A$=A$+"1"
80 NEXT I
90 OLED TILE NEW 1 16 16 A$ 1
100 B$=""
110 FOR I=1 TO 256
120   B$=B$+"0"
130 NEXT I
140 OLED TILE NEW 2 16 16 B$ 1
150 OLED TILE ANIM NEW 0 2 1 2 MS 500
160 OLED TILEMAP NEW 0 8 4 16 16 1 0
170 OLED TILEMAP ANIMFILL 0 0
180 OLED TILE ANIM TICK 100
190 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
200 GOTO 180
```

Slower moving animation (large tick)

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 P$=""
40 FOR I=1 TO 256
50   P$=P$+"1"
60 NEXT I
70 OLED TILE NEW 1 16 16 P$ 1
80 Q$=""
90 FOR Y=0 TO 15
100  FOR X=0 TO 15
110    S=X+Y
120    T=S-(INT(S/2)*2)
130    IF T=0 THEN Q$=Q$+"1"
140    IF T<>0 THEN Q$=Q$+"0"
150  NEXT X
160 NEXT Y
170 OLED TILE NEW 2 16 16 Q$ 1
180 OLED TILE ANIM NEW 0 2 1 2 MS 120
190 OLED TILEMAP NEW 0 8 4 16 16 1 0
200 OLED TILEMAP ANIMFILL 0 0
210 OLED TILE ANIM TICK 16
220 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
230 GOTO 210
```

Smoother animation (small tick)

## Note

- msDelta is “how much time has passed” between one frame and another.
- If you call TICK but don't redraw, you don't see the change.

# OLED TILE CLEAR

## Sintassi

```
OLED TILE CLEAR
```

## Descrizione

Delete all loaded tiles (0..63). It's a complete reset of the tile library.

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 A$=""
40 FOR I=1 TO 256
50   A$=A$+"1"
60 NEXT I
70 OLED TILE NEW 1 16 16 A$ 1
80 B$=""
90 FOR Y=0 TO 15
100  FOR X=0 TO 15
110    IF X=Y THEN B$=B$+"1"
120    IF X<>Y THEN B$=B$+"0"
130  NEXT X
140 NEXT Y
150 OLED TILE NEW 2 16 16 B$ 1
160 OLED TILEMAP NEW 0 8 4 16 16 1 0
170 FOR R=0 TO 3
180   FOR C=0 TO 7
190     IF R=0 THEN OLED TILEMAP SET 0 C R 1
200     IF R<>0 THEN OLED TILEMAP SET 0 C R 2
210   NEXT C
220 NEXT R
230 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
240 FOR K=1 TO 30000
250 NEXT K
260 OLED TILE CLEAR
270 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
```

Show mosaic, then CLEAR and redraw (effect visible)

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 P$=""
40 FOR I=1 TO 256
50   P$=P$+"1"
60 NEXT I
70 OLED TILE NEW 1 16 16 P$ 1
80 OLED TILEMAP NEW 0 8 4 16 16 1 0
90 OLED TILEMAP SET 0 3 1 1
100 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
110 FOR K=1 TO 20000
120 NEXT K
130 OLED TILE CLEAR
140 X$=""
150 FOR Y=0 TO 15
160  FOR X=0 TO 15
170    B=0
180    IF X=Y THEN B=1
```

```
190    IF X=15-Y THEN B=1
200    IF B=1 THEN X$=X$+"1"
210    IF B<>1 THEN X$=X$+"0"
220  NEXT X
230 NEXT Y
240 OLED TILE NEW 1 16 16 X$ 1
250 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
```

CLEAR as reset and reload a different tile

## Note

- OLED TILE CLEAR does not clear tilemaps: it only clears tiles (graphics/flags/hitboxes).

# OLED TILE FLAGS

## Sintassi

```
OLED TILE FLAGS <tileId> <mask>
```

## Descrizione

Assigns the tile a mask of “logical” flags used by the engine for collisions and triggers. Flags (bitmask):

```
1 SOLID
2 PLATFORMS
4 LADDER
8 WATER
16 DOORS
32 HAZARD
64 SLOW
128 TRIGGER
```

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'Tile 1 full SOLID
40 P$=""
50 FOR I=1 TO 256
60   P$=P$+"1"
70 NEXT I
80 OLED TILE NEW 1 16 16 P$ 1
90 OLED TILE FLAGS 1 1
100 'Tile 2 WATER chessboard
110 Q$=""
120 FOR Y=0 TO 15
130   FOR X=0 TO 15
140     A=X+Y
150     B=A-(INT(A/2)*2)
160     IF B=0 THEN Q$=Q$+"1"
170     IF B<>0 THEN Q$=Q$+"0"
180   NEXT X
190 NEXT Y
200 OLED TILE NEW 2 16 16 Q$ 1
210 OLED TILE FLAGS 2 8
220 OLED TILEMAP NEW 0 8 4 16 16 1 0
230 OLED TILEMAP SET 0 2 1 1
240 OLED TILEMAP SET 0 5 2 2
250 OLED TILEMAP GET 0 2 1 SET TID FLAGS FL
260 OLED TILEMAP GET 0 5 2 SET TID2 FLAGS FL2
270 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1

(Here you see two different tiles on the screen; and in FL / FL2 you find 1 and 8.)
```

“Reading” flags with TILEMAP GET FLAGS (shows difference 1 vs 2)

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'Tile 1 full (door)
40 P$=""
50 FOR I=1 TO 256
60   P$=P$+"1"
70 NEXT I
```

```
80 OLED TILE NEW 1 16 16 P$ 1
90 OLED TILE FLAGS 1 16
100 'Tile 2 diagonal (player)
110 PL$=""
120 FOR Y=0 TO 15
130   FOR X=0 TO 15
140     IF X=Y THEN PL$=PL$+"1"
150     IF X<>Y THEN PL$=PL$+"0"
160   NEXT X
170 NEXT Y
180 OLED TILE NEW 2 16 16 PL$ 1
190 OLED TILEMAP NEW 0 8 4 16 16 1 0
200 OLED TILEMAP SET 0 6 1 1
210 PX=0: PY=16
220 OLD_C=-1: OLD_R=-1
230 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 0
240 IF OLD_C>=0 THEN OLED TILEMAP SET 0 OLD_C OLD_R 0
250 PX=PX+1
260 IF PX>112 THEN PX=0
270 OLED TILEMAP TRIGGER 0 RECT PX PY 16 16 FLAGS 16 SET HIT TID COL ROW
280 C=INT(PX/16)
290 R=INT(PY/16)
300 OLED TILEMAP SET 0 C R 2
310 OLD_C=C: OLD_R=R
320 IF HIT<>0 THEN OLED TILEMAP SET 0 6 1 2
330 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
340 GOTO 240
```

DOOR “door” trigger (16): when you tap, changes tile

## Note

- Flags are useful when you use commands like MOVE, COLLIDE, TRIGGER, HITINFO, RAYCAST.
- The mask can be a sum: e.g. WATER + SLOW = 8 + 64 = 72.

# OLED TILE HITBOX

## Sintassi

```
OLED TILE HITBOX <tileId> <ox> <oy> <w> <h>
```

## Descrizione

Set a custom hitbox for that tile. The hitbox is used for collisions (for example in MOVE and COLLIDE).

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'Full platform tile
40 T$=""
50 FOR I=1 TO 256
60   T$=T$+"1"
70 NEXT I
80 OLED TILE NEW 1 16 16 T$ 1
90 OLED TILE FLAGS 1 2
100 OLED TILE HITBOX 1 0 12 16 4
110 'Diagonal tile player
120 P$=""
130 FOR Y=0 TO 15
140   FOR X=0 TO 15
150     IF X=Y THEN P$=P$+"1"
160     IF X<>Y THEN P$=P$+"0"
170   NEXT X
180 NEXT Y
190 OLED TILE NEW 2 16 16 P$ 1
200 OLED TILEMAP NEW 0 20 20 16 16 1 0
210 FOR C=1 TO 18
220   OLED TILEMAP SET 0 C 10 1
230 NEXT C
240 PX=64: PY=0: VX=0: VY=0
250 OLD_C=-1: OLD_R=-1
260 VY=VY+1
270 OLED TILEMAP MOVE 0 RECT PX PY 16 16 BY VX VY FLAGS 2 SET PX PY HX HY GR
280 IF GR<>0 THEN VY=0
290 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 0
300 IF OLD_C>=0 THEN OLED TILEMAP SET 0 OLD_C OLD_R 0
310 C=INT(PX/16)
320 R=INT(PY/16)
330 OLED TILEMAP SET 0 C R 2
340 OLD_C=C: OLD_R=R
350 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
360 GOTO 260
```

Thin platformer (hitbox only at the bottom): the “player” falls and stops

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'SOLID full wall with narrow hitbox
40 W$=""
50 FOR I=1 TO 256
60   W$=W$+"1"
70 NEXT I
80 OLED TILE NEW 1 16 16 W$ 1
```

```
90 OLED TILE FLAGS 1 1
100 OLED TILE HITBOX 1 4 0 8 16
110 'Full player
120 P$=""
130 FOR I=1 TO 256
140   P$=P$+"1"
150 NEXT I
160 OLED TILE NEW 2 16 16 P$ 1
170 OLED TILEMAP NEW 0 20 20 16 16 1 0
180 FOR R=0 TO 19
190   OLED TILEMAP SET 0 10 R 1
200 NEXT R
210 PX=0: PY=32: VX=2: VY=0
220 OLD_C=-1: OLD_R=-1
230 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 0
240 IF OLD_C>=0 THEN OLED TILEMAP SET 0 OLD_C OLD_R 0
250 OLED TILEMAP MOVE 0 RECT PX PY 16 16 BY VX VY FLAGS 1 SET PX PY HX HY GR
260 IF HX<>0 THEN VX=0-VX
270 C=INT(PX/16)
280 R=INT(PY/16)
290 OLED TILEMAP SET 0 C R 2
300 OLD_C=C: OLD_R=R
310 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
320 GOTO 240
```

Smaller hitbox on the wall: the bounce occurs "further in"

## Note

- The hitbox is relative to the tile (offset + size).
- By default the hitbox coincides with the entire tile.

# OLED TILE NEW

## Sintassi

```
OLED TILE NEW <id> <w> <h> <pattern$> [scale]
```

## Descrizione

Creates (or replaces) a graphic tile. The tile is a monochrome image defined by a string of 0s and 1s.

```
1 = pixel on
0 = pixel off
```

The pattern$ string should be long w*h (for 16×16 → 256 characters).

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'TILE 1: full
40 P1$=""
50 FOR I=1 TO 256
60   P1$=P1$+"1"
70 NEXT I
80 OLED TILE NEW 1 16 16 P1$ 1
90 'TILE 2: chessboard
100 P2$=""
110 FOR Y=0 TO 15
120   FOR X=0 TO 15
130     A=X+Y
140     B=A-(INT(A/2)*2)
150     IF B=0 THEN P2$=P2$+"1"
160     IF B<>0 THEN P2$=P2$+"0"
170   NEXT X
180 NEXT Y
190 OLED TILE NEW 2 16 16 P2$ 1
200 'TILE 3: diagonal
210 P3$=""
220 FOR Y=0 TO 15
230   FOR X=0 TO 15
240     IF X=Y THEN P3$=P3$+"1"
250     IF X<>Y THEN P3$=P3$+"0"
260   NEXT X
270 NEXT Y
280 OLED TILE NEW 3 16 16 P3$ 1
290 OLED TILEMAP NEW 0 8 4 16 16 1 0
300 FOR R=0 TO 3
310   FOR C=0 TO 7
320     A=C+R
330     V=A-(INT(A/3)*3)
340     V=V+1
350     OLED TILEMAP SET 0 C R V
360   NEXT C
370 NEXT R
380 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
```

3 different tiles + 8×4 mosaic (fills the screen)

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 PX$=""
```

```
40 FOR Y=0 TO 15
50   FOR X=0 TO 15
60     B=0
70     IF X=Y THEN B=1
80     IF X=15-Y THEN B=1
90     IF B=1 THEN PX$=PX$+"1"
100    IF B<>1 THEN PX$=PX$+"0"
110   NEXT X
120 NEXT Y
130 OLED TILE NEW 1 16 16 PX$ 1
140 OLED TILEMAP NEW 0 8 4 16 16 1 0
150 FOR R=0 TO 3
160   FOR C=0 TO 7
170     OLED TILEMAP SET 0 C R 1
180   NEXT C
190 NEXT R
200 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
```

Tile “X” (two diagonals) and screen fill

## Note

- If you recreate the same <id>, the previous tile is replaced.
- If pattern$ is shorter than w*h, missing pixels become 0.
- scales if you don't specify it it's 1 (if you put 0 it's forced to 1).

# OLED TILEMAP ANIM

## Sintassi

```
OLED TILEMAP ANIM <mapId> <col> <row> <animId>
```

## Descrizione

Enable (or disable) an animation on a single cell. If animId does not exist or is invalid, animation on that cell is disabled. Important: the animation advances only if you also call OLED TILE ANIM TICK in the loop.

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'Tile 1 full
40 P$=""
50 FOR I=1 TO 256
60   P$=P$+"1"
70 NEXT I
80 OLED TILE NEW 1 16 16 P$ 1
90 'Tile 2 empty
100 Z$=""
110 FOR I=1 TO 256
120   Z$=Z$+"0"
130 NEXT I
140 OLED TILE NEW 2 16 16 Z$ 1
150 OLED TILE ANIM NEW 0 2 1 2 MS 250
160 OLED TILEMAP NEW 0 8 4 16 16 1 0
170 OLED TILEMAP SET 0 3 1 1
180 OLED TILEMAP ANIM 0 3 1 0
190 OLED TILE ANIM TICK 16
200 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
210 GOTO 190
```

A single animated cell (flashing light bulb)

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'Tile 1 chessboard
40 S$=""
50 FOR Y=0 TO 15
60  FOR X=0 TO 15
70    A=X+Y
80    B=A-(INT(A/2)*2)
90    IF B=0 THEN S$=S$+"1"
100   IF B<>0 THEN S$=S$+"0"
110  NEXT X
120 NEXT Y
130 OLED TILE NEW 1 16 16 S$ 1
140 'Tile 2 diagonal
150 D$=""
160 FOR Y=0 TO 15
170  FOR X=0 TO 15
180    IF X=Y THEN D$=D$+"1"
190    IF X<>Y THEN D$=D$+"0"
200  NEXT X
210 NEXT Y
220 OLED TILE NEW 2 16 16 D$ 1
```

```
230 OLED TILE ANIM NEW 0 2 1 2 MS 160
240 OLED TILEMAP NEW 0 8 4 16 16 1 0
250 'animate a 6x2 rectangle (col 1..6, row 1..2)
260 OLED TILEMAP ANIMFILL 0 RECT 1 1 6 2 0
270 'and also animates an isolated cell above
280 OLED TILEMAP ANIM 0 7 0 0
290 OLED TILE ANIM TICK 16
300 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
310 GOTO 290
```

Use ANIMFILL to animate a rectangle, then ANIM on a different cell

## Note

- If you do OLED TILEMAP SET on an animated cell, that animation is turned off for that cell.
- For “many cells together”, use OLED TILEMAP ANIMFILL (we document this in the next dedicated section).

# OLED TILEMAP ANIMFILL

## Sintassi

```
OLED TILEMAP ANIMFILL <mapId> <animId>
OLED TILEMAP ANIMFILL <mapId> RECT <col> <row> <w> <h> <animId>
```

## Descrizione

Activate (or deactivate) an animation on: the entire map (first shape), or a rectangle of cells (RECT shape). If animId is invalid/does not exist, set “no animation” (disable).

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'Tile 1 full
40 P$=""
50 FOR I=1 TO 256
60   P$=P$+"1"
70 NEXT I
80 OLED TILE NEW 1 16 16 P$ 1
90 'Tile 2 empty
100 Z$=""
110 FOR I=1 TO 256
120   Z$=Z$+"0"
130 NEXT I
140 OLED TILE NEW 2 16 16 Z$ 1
150 OLED TILE ANIM NEW 0 2 1 2 MS 250
160 OLED TILEMAP NEW 0 8 4 16 16 1 1
170 OLED TILEMAP ANIMFILL 0 0
180 OLED TILE ANIM TICK 16
190 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
200 GOTO 180
```

Animate entire map (blink full/empty)

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'Tile 1 chessboard
40 S$=""
50 FOR Y=0 TO 15
60  FOR X=0 TO 15
70    A=X+Y
80    B=A-(INT(A/2)*2)
90    IF B=0 THEN S$=S$+"1"
100   IF B<>0 THEN S$=S$+"0"
110  NEXT X
120 NEXT Y
130 OLED TILE NEW 1 16 16 S$ 1
140 'Tile 2 diagonal
150 D$=""
160 FOR Y=0 TO 15
170  FOR X=0 TO 15
180    IF X=Y THEN D$=D$+"1"
190    IF X<>Y THEN D$=D$+"0"
200  NEXT X
210 NEXT Y
220 OLED TILE NEW 2 16 16 D$ 1
230 OLED TILE ANIM NEW 1 2 1 2 MS 140
```

```
240 OLED TILEMAP NEW 0 8 4 16 16 1 0
250 'I fill background with static tile 1
260 OLED TILEMAP FILL 0 1
270 'animate only the rectangle: col=1 row=1 w=6 h=2
280 OLED TILEMAP ANIMFILL 0 RECT 1 1 6 2 1
290 OLED TILE ANIM TICK 16
300 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
310 GOTO 290
```

It just animates a central 6×2 rectangle

## Note

- To see the animation you need to call OLED TILE ANIM TICK in the loop.
- RECT is in cells, not pixels.

# OLED TILEMAP CLEAR

## Sintassi

```
OLED TILEMAP CLEAR
```

## Descrizione

Delete all tilemaps (0..3) and free the associated memory.

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 P$=""
40 FOR I=1 TO 256
50   P$=P$+"1"
60 NEXT I
70 OLED TILE NEW 1 16 16 P$ 1
80 OLED TILEMAP NEW 0 8 4 16 16 1 0
90 OLED TILEMAP FILL 0 1
100 OLED TILEMAP NEW 1 8 4 16 16 1 0
110 OLED TILEMAP SET 1 3 1 1
120 OLED TILEMAP SET 1 4 2 1
130 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
140 FOR K=1 TO 20000
150 NEXT K
160 OLED TILEMAP DRAW 1 0 0 CAMERA 0 0 1
170 FOR K=1 TO 20000
180 NEXT K
190 OLED TILEMAP CLEAR
200 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
```

Create two maps, draw, then CLEAR and redraw (they disappear)

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 P$=""
40 FOR I=1 TO 256
50   P$=P$+"1"
60 NEXT I
70 OLED TILE NEW 1 16 16 P$ 1
80 OLED TILEMAP NEW 0 8 4 16 16 1 0
90 OLED TILEMAP SET 0 1 1 1
100 OLED TILEMAP SET 0 2 2 1
110 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
120 FOR K=1 TO 20000
130 NEXT K
140 OLED TILEMAP CLEAR
150 OLED TILEMAP NEW 0 8 4 16 16 1 0
160 OLED TILEMAP SET 0 6 1 1
170 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
```

CLEAR as “level reset”

## Note

- TILEMAP CLEAR does not affect the tiles (you manage those with OLED TILE CLEAR).

# OLED TILEMAP COLLIDE

## Sintassi

```
OLED TILEMAP COLLIDE <mapId> RECT <x> <y> <w> <h> FLAGS <mask> SET <var>
OLED TILEMAP COLLIDE <mapId> RECT <x> <y> <w> <h> FLAGS <mask> GOTO <line>
```

## Descrizione

Checks if a rectangle (in “world” pixels) touches at least one tile with flags compatible with . SET mode: writes 1 (hit) or 0 (no hit) to a variable. GOTO mode: if there is a hit it jumps to a BASIC line.

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'SOLID wall tile (full)
40 W$=""
50 FOR I=1 TO 256
60   W$=W$+"1"
70 NEXT I
80 OLED TILE NEW 1 16 16 W$ 1
90 OLED TILE FLAGS 1 1
100 'Diagonal marker tile
110 D$=""
120 FOR Y=0 TO 15
130  FOR X=0 TO 15
140    IF X=Y THEN D$=D$+"1"
150    IF X<>Y THEN D$=D$+"0"
160  NEXT X
170 NEXT Y
180 OLED TILE NEW 2 16 16 D$ 1
190 OLED TILEMAP NEW 0 20 20 16 16 1 0
200 FOR R=0 TO 19
210   OLED TILEMAP SET 0 8 R 1
220 NEXT R
230 PX=0: PY=16
240 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
250 PX=PX+2
260 IF PX>304 THEN PX=0
270 OLED TILEMAP COLLIDE 0 RECT PX PY 16 16 FLAGS 1 SET HIT
280 'I show indicator up if I touch the wall
290 OLED TILEMAP SET 0 0 0 0
300 IF HIT<>0 THEN OLED TILEMAP SET 0 0 0 2
310 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
320 GOTO 250
```

SET: moves a point and lights an indicator when it touches a SOLID wall

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'Tile hazard: board + HAZARD
40 H$=""
50 FOR Y=0 TO 15
60  FOR X=0 TO 15
70    A=X+Y
80    B=A-(INT(A/2)*2)
90    IF B=0 THEN H$=H$+"1"
100   IF B<>0 THEN H$=H$+"0"
110  NEXT X
```

```
120 NEXT Y
130 OLED TILE NEW 1 16 16 H$ 1
140 OLED TILE FLAGS 1 32
150 'Tile player: full
160 P$=""
170 FOR I=1 TO 256
180   P$=P$+"1"
190 NEXT I
200 OLED TILE NEW 2 16 16 P$ 1
210 OLED TILEMAP NEW 0 20 20 16 16 1 0
220 OLED TILEMAP SET 0 6 1 1
230 PX=0: PY=16
240 OLD_C=-1: OLD_R=-1
250 'check hazard, if yes go back to START
260 OLED TILEMAP COLLIDE 0 RECT PX PY 16 16 FLAGS 32 GOTO 400
270 IF OLD_C>=0 THEN OLED TILEMAP SET 0 OLD_C OLD_R 0
280 C=INT(PX/16)
290 R=INT(PY/16)
300 OLED TILEMAP SET 0 C R 2
310 OLD_C=C: OLD_R=R
320 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
330 PX=PX+1
340 IF PX>304 THEN PX=0
350 GOTO 250
400 PX=0
410 GOTO 250
```

GOTO: if you touch HAZARD (32) you start again from the left

## Note

- <x> <y> <w> <h> are in pixel world.
- Typical wall mask: 1 (SOLID). Per hazard: 32.

# OLED TILEMAP DELETE

## Sintassi

```
OLED TILEMAP DELETE <mapId>
```

## Descrizione

Delete a tilemap from memory (cells and associated animations).

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 P$=""
40 FOR I=1 TO 256
50   P$=P$+"1"
60 NEXT I
70 OLED TILE NEW 1 16 16 P$ 1
80 OLED TILEMAP NEW 0 8 4 16 16 1 0
90 OLED TILEMAP FILL 0 1
100 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
110 FOR K=1 TO 30000
120 NEXT K
130 OLED TILEMAP DELETE 0
140 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
```

You draw, erase the map, then try to redraw (it doesn't change anymore)

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 P$=""
40 FOR I=1 TO 256
50   P$=P$+"1"
60 NEXT I
70 OLED TILE NEW 1 16 16 P$ 1
80 OLED TILEMAP NEW 0 8 4 16 16 1 0
90 OLED TILEMAP FILL 0 1
100 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
110 FOR K=1 TO 20000
120 NEXT K
130 OLED TILEMAP DELETE 0
140 OLED TILEMAP NEW 0 4 4 16 16 1 0
150 OLED TILEMAP FILL 0 1
160 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
```

Delete and recreate with different size (visible effect)

## Note

- After DELETE, the map is no longer valid until you recreate it with TILEMAP NEW.

# OLED TILEMAP DRAW

## Sintassi

```
OLED TILEMAP DRAW <mapId> <sx> <sy> CAMERA <camX> <camY> [drawNow]
```

## Descrizione

Draw the tilemap on the display. : position on the screen (pixels) where to draw camX camY: “camera” offset in pixels (scrolling)

```
[drawNow] (optional): 1 = update the display immediately, 0 = leave in buffer
```

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'Tile 1 full
40 P$=""
50 FOR I=1 TO 256
60   P$=P$+"1"
70 NEXT I
80 OLED TILE NEW 1 16 16 P$ 1
90 OLED TILEMAP NEW 0 8 4 16 16 1 0
100 FOR R=0 TO 3
110  FOR C=0 TO 7
120    IF R=0 THEN OLED TILEMAP SET 0 C R 1
130    IF R=3 THEN OLED TILEMAP SET 0 C R 1
140    IF R<>0 THEN IF R<>3 THEN OLED TILEMAP SET 0 C R 0
150  NEXT C
160 NEXT R
170 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
```

Draw a fixed 8x4 (camera 0.0)

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'Tile 1 full
40 P$=""
50 FOR I=1 TO 256
60   P$=P$+"1"
70 NEXT I
80 OLED TILE NEW 1 16 16 P$ 1
90 'Tile 2 chessboard
100 S$=""
110 FOR Y=0 TO 15
120  FOR X=0 TO 15
130    A=X+Y
140    B=A-(INT(A/2)*2)
150    IF B=0 THEN S$=S$+"1"
160    IF B<>0 THEN S$=S$+"0"
170  NEXT X
180 NEXT Y
190 OLED TILE NEW 2 16 16 S$ 1
200 OLED TILEMAP NEW 0 20 20 16 16 1 0
210 'checkerboard of tileId 1 and 2
220 FOR R=0 TO 19
230  FOR C=0 TO 19
240    A=C+R
250    B=A-(INT(A/2)*2)
260    IF B=0 THEN OLED TILEMAP SET 0 C R 1
```

```
270    IF B<>0 THEN OLED TILEMAP SET 0 C R 2
280  NEXT C
290 NEXT R
300 CAMX=0: CAMY=0
310 CAMX=CAMX+2
320 IF CAMX>192 THEN CAMX=0
330 OLED TILEMAP DRAW 0 0 0 CAMERA CAMX CAMY 1
340 GOTO 310
```

Automatic scrolling on 20×20 map (moving camera)

## Note

- camX/camY are in pixels, not cells.
- drawNow=0 is useful if you want to update later with DRAWRECT.

# OLED TILEMAP DRAWRECT

## Sintassi

```
OLED TILEMAP DRAWRECT <mapId> <sx> <sy> CAMERA <camX> <camY> <x> <y> <w> <h>
```

## Descrizione

Redraws the tilemap in the buffer and updates only a display rectangle (in pixels). It is used to optimize: instead of updating the entire screen, you update only the changed area. are coordinates/sizes on the display (pixels)

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'Tile 1 full
40 P$=""
50 FOR I=1 TO 256
60   P$=P$+"1"
70 NEXT I
80 OLED TILE NEW 1 16 16 P$ 1
90 'Tile 2 diagonal (player)
100 D$=""
110 FOR Y=0 TO 15
120  FOR X=0 TO 15
130    IF X=Y THEN D$=D$+"1"
140    IF X<>Y THEN D$=D$+"0"
150  NEXT X
160 NEXT Y
170 OLED TILE NEW 2 16 16 D$ 1
180 OLED TILEMAP NEW 0 8 4 16 16 1 0
190 OLED TILEMAP FILL 0 0
200 PX=0: PY=16
210 OLDX=0
220 'complete initial drawing
230 OLED TILEMAP SET 0 0 1 2
240 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
250 'loop: move right
260 OLDX=PX
270 PX=PX+16
280 IF PX>112 THEN PX=0
290 OLED TILEMAP SET 0 INT(OLDX/16) 1 0
300 OLED TILEMAP SET 0 INT(PX/16) 1 2
310 'I only update high end 32x16 where it moves (2 cells)
320 XU=OLDX
330 IF PX<XU THEN XU=PX
340 OLED TILEMAP DRAWRECT 0 0 0 CAMERA 0 0 XU 16 32 16
350 FOR K=1 TO 6000
360 NEXT K
370 GOTO 260
```

Moves a “player” and updates only 32×16 pixels (2 cells)

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'Tile 1 full
40 P$=""
50 FOR I=1 TO 256
60   P$=P$+"1"
```

```
70 NEXT I
80 OLED TILE NEW 1 16 16 P$ 1
90 OLED TILEMAP NEW 0 8 4 16 16 1 0
100 OLED TILEMAP FILL 0 0
110 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
120 V=0
130 IF V=0 THEN OLED TILEMAP SET 0 0 0 1
140 IF V<>0 THEN OLED TILEMAP SET 0 0 0 0
150 OLED TILEMAP DRAWRECT 0 0 0 CAMERA 0 0 0 0 16 16
160 V=1-V
170 FOR K=1 TO 12000
180 NEXT K
190 GOTO 130
```

Only change one corner (16×16) with blink

## Note

- The rectangle is in screen pixels, so for a 16×16 cell: x=col*16, y=row*16.
- This is ideal when you change very few cells per frame.

# OLED TILEMAP FILL

## Sintassi

```
OLED TILEMAP FILL <mapId> <tileId>
```

## Descrizione

Fills the entire tilemap with the same tileId. It is useful for "cleaning up" the layer or quickly creating a uniform background.

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'Tile 1: chessboard
40 T$=""
50 FOR Y=0 TO 15
60   FOR X=0 TO 15
70     A=X+Y
80     B=A-(INT(A/2)*2)
90     IF B=0 THEN T$=T$+"1"
100    IF B<>0 THEN T$=T$+"0"
110  NEXT X
120 NEXT Y
130 OLED TILE NEW 1 16 16 T$ 1
140 OLED TILEMAP NEW 0 8 4 16 16 1 0
150 OLED TILEMAP FILL 0 1
160 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
```

Fills the screen with chessboard

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'Tile 1: full
40 P$=""
50 FOR I=1 TO 256
60   P$=P$+"1"
70 NEXT I
80 OLED TILE NEW 1 16 16 P$ 1
90 OLED TILEMAP NEW 0 8 4 16 16 1 0
100 OLED TILEMAP FILL 0 1
110 'I dig an empty lane on row 2
120 FOR C=0 TO 7
130   OLED TILEMAP SET 0 C 2 0
140 NEXT C
150 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
```

Fills, then “hollows out” an empty line with SET

## Note

- FILL deactivates any animations on the cells (makes them “static”).
- It is faster than doing SET cell by cell.

# OLED TILEMAP GET

## Sintassi

```
OLED TILEMAP GET <mapId> <col> <row> SET <var> [FLAGS <varFlags>]
```

## Descrizione

Reads what is in a cell of the tilemap and writes it to a variable. Optionally also reads the flags of the tile present in that cell. SET receives tileId FLAGS receives tile flags

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 P$=""
40 FOR I=1 TO 256
50   P$=P$+"1"
60 NEXT I
70 OLED TILE NEW 1 16 16 P$ 1
80 Q$=""
90 FOR Y=0 TO 15
100  FOR X=0 TO 15
110    IF X=Y THEN Q$=Q$+"1"
120    IF X<>Y THEN Q$=Q$+"0"
130  NEXT X
140 NEXT Y
150 OLED TILE NEW 2 16 16 Q$ 1
160 OLED TILEMAP NEW 0 8 4 16 16 1 0
170 OLED TILEMAP SET 0 2 1 2
180 OLED TILEMAP GET 0 2 1 SET TID
190 OLED TILEMAP SET 0 5 2 TID
200 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
```

Read a cell and “copy” its tile to another location

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'Tile 1 full SOLID
40 P$=""
50 FOR I=1 TO 256
60   P$=P$+"1"
70 NEXT I
80 OLED TILE NEW 1 16 16 P$ 1
90 OLED TILE FLAGS 1 1
100 'Tile 2 diagonal (marker)
110 Q$=""
120 FOR Y=0 TO 15
130  FOR X=0 TO 15
140    IF X=Y THEN Q$=Q$+"1"
150    IF X<>Y THEN Q$=Q$+"0"
160  NEXT X
170 NEXT Y
180 OLED TILE NEW 2 16 16 Q$ 1
190 OLED TILEMAP NEW 0 8 4 16 16 1 0
200 OLED TILEMAP SET 0 3 1 1
210 OLED TILEMAP GET 0 3 1 SET TID FLAGS FLG
220 'If FLG contains SOLID (1), I put a marker in the top left
230 IF FLG=1 THEN OLED TILEMAP SET 0 0 0 2
240 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
```

Read tileId and FLAGS, then if it is SOLID (1) draw a marker

## Note

- If col/row are outside the map, the command writes 0 to the variables.
- FLAGS returns the tile mask (0–255).

# OLED TILEMAP HITINFO

## Sintassi

```
OLED TILEMAP HITINFO <mapId> RECT <x> <y> <w> <h> FLAGS <mask>
SET <hit> <tid> <col> <row> <flags>
```

## Descrizione

Like COLLIDE, but in addition to telling you if there is a collision, it returns: the tileId hit the cell with the row the flags of the tile hit

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'SOLID wall tile (full)
40 W$=""
50 FOR I=1 TO 256
60   W$=W$+"1"
70 NEXT I
80 OLED TILE NEW 1 16 16 W$ 1
90 OLED TILE FLAGS 1 1
100 'Tile hazard (board) + HAZARD
110 H$=""
120 FOR Y=0 TO 15
130  FOR X=0 TO 15
140    A=X+Y
150    B=A-(INT(A/2)*2)
160    IF B=0 THEN H$=H$+"1"
170    IF B<>0 THEN H$=H$+"0"
180  NEXT X
190 NEXT Y
200 OLED TILE NEW 2 16 16 H$ 1
210 OLED TILE FLAGS 2 32
220 'Diagonal tile player
230 P$=""
240 FOR Y=0 TO 15
250  FOR X=0 TO 15
260    IF X=Y THEN P$=P$+"1"
270    IF X<>Y THEN P$=P$+"0"
280  NEXT X
290 NEXT Y
300 OLED TILE NEW 3 16 16 P$ 1
310 OLED TILEMAP NEW 0 20 20 16 16 1 0
320 'column walls SOLID
330 FOR R=0 TO 19
340   OLED TILEMAP SET 0 8 R 1
350 NEXT R
360 'a hazard cell
370 OLED TILEMAP SET 0 12 1 2
380 PX=0: PY=16
390 OLD_C=-1: OLD_R=-1
400 'mask = SOLID + HAZARD = 1 + 32 = 33
410 OLED TILEMAP HITINFO 0 RECT PX PY 16 16 FLAGS 33 SET HIT TID TC TR FL
420 OLED TILEMAP SET 0 0 0 0
430 IF HIT<>0 THEN OLED TILEMAP SET 0 0 0 TID
440 IF OLD_C>=0 THEN OLED TILEMAP SET 0 OLD_C OLD_R 0
450 C=INT(PX/16)
460 R=INT(PY/16)
```

```
470 OLED TILEMAP SET 0 C R 3
480 OLD_C=C: OLD_R=R
490 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
500 PX=PX+1
510 IF PX>304 THEN PX=0
520 GOTO 410
```

Do you touch? then shows the affected tile at the top left

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'Tile hazards
40 H$=""
50 FOR Y=0 TO 15
60  FOR X=0 TO 15
70    A=X+Y
80    B=A-(INT(A/2)*2)
90    IF B=0 THEN H$=H$+"1"
100   IF B<>0 THEN H$=H$+"0"
110  NEXT X
120 NEXT Y
130 OLED TILE NEW 1 16 16 H$ 1
140 OLED TILE FLAGS 1 32
150 ' player
160 P$=""
170 FOR I=1 TO 256
180   P$=P$+"1"
190 NEXT I
200 OLED TILE NEW 2 16 16 P$ 1
210 OLED TILEMAP NEW 0 20 20 16 16 1 0
220 OLED TILEMAP SET 0 6 1 1
230 PX=0: PY=16
240 OLD_C=-1: OLD_R=-1
250 OLED TILEMAP HITINFO 0 RECT PX PY 16 16 FLAGS 32 SET HIT TID TC TR FL
260 IF HIT<>0 THEN PX=0
270 IF OLD_C>=0 THEN OLED TILEMAP SET 0 OLD_C OLD_R 0
280 C=INT(PX/16)
290 R=INT(PY/16)
300 OLED TILEMAP SET 0 C R 2
310 OLD_C=C: OLD_R=R
320 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
330 PX=PX+1
340 IF PX>304 THEN PX=0
350 GOTO 250
```

If you hit HAZARD (32) then reset position

## Note

- HITINFO is ideal for understanding “what kind of tile did I touch?” (water, stairs, lava…).
- The mask can include multiple summed flags.

# OLED TILEMAP LOAD

## Sintassi

```
OLED TILEMAP LOAD <mapId> <data$> [base]
```

## Descrizione

Load the tilemap from a given$ string. Each character represents a value (tileId) in a numeric base.

```
base default = 36 (0–9, A–Z)
```

if you specify [base], you can limit it (e.g. 10 → 0–9 only) The string is read in line-by-line order.

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'Tile 1: full
40 P$=""
50 FOR I=1 TO 256
60   P$=P$+"1"
70 NEXT I
80 OLED TILE NEW 1 16 16 P$ 1
90 'Tile 2: diagonal
100 D$=""
110 FOR Y=0 TO 15
120  FOR X=0 TO 15
130    IF X=Y THEN D$=D$+"1"
140    IF X<>Y THEN D$=D$+"0"
150  NEXT X
160 NEXT Y
170 OLED TILE NEW 2 16 16 D$ 1
180 OLED TILEMAP NEW 0 8 4 16 16 1 0
190 '8x4 = 32 cells: I make a border of 1 and interior 2/0
200 DAT$="11111111"
210 DAT$=DAT$+"12000021"
220 DAT$=DAT$+"12000021"
230 DAT$=DAT$+"11111111"
240 OLED TILEMAP LOAD 0 DAT$ 10
250 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
```

Load an 8x4 map using base 10 (32 characters)

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'Tile 10 (A): full
40 P$=""
50 FOR I=1 TO 256
60   P$=P$+"1"
70 NEXT I
80 OLED TILE NEW 10 16 16 P$ 1
90 'Tile 11 (B): chessboard
100 S$=""
110 FOR Y=0 TO 15
120  FOR X=0 TO 15
130    A=X+Y
140    B=A-(INT(A/2)*2)
150    IF B=0 THEN S$=S$+"1"
160    IF B<>0 THEN S$=S$+"0"
170  NEXT X
```

```
180 NEXT Y
190 OLED TILE NEW 11 16 16 S$ 1
200 OLED TILEMAP NEW 0 8 4 16 16 1 0
210 'I use A and B in the string (base 36)
220 DAT$="AAAAAAAA"
230 DAT$=DAT$+"ABBBBBBA"
240 DAT$=DAT$+"ABBBBBBA"
250 DAT$=DAT$+"AAAAAAAA"
260 OLED TILEMAP LOAD 0 DAT$
270 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
```

Base 36: I also use letters (A=10, B=11, etc.)

## Note

- If data$ is shorter, the missing cells become 0.
- If base is 10, letters become 0 (invalid in that base).
- With base 36 you can represent up to 35 tileIds with a single character.

# OLED TILEMAP MOVE

## Sintassi

```
OLED TILEMAP MOVE <mapId> RECT <x> <y> <w> <h> BY <dx> <dy> FLAGS <mask>
SET <outX> <outY> <hitX> <hitY> <ground>
```

## Descrizione

Moves a rectangle in a tiled world with collisions. Returns: outX outY: new position (corrected to avoid walls) hitX hitY: 1 if it hit on that axis, otherwise 0 ground: 1 if it is "on the ground" (useful for gravity/jumping), otherwise 0

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'SOLID ground tile (full)
40 G$=""
50 FOR I=1 TO 256
60   G$=G$+"1"
70 NEXT I
80 OLED TILE NEW 1 16 16 G$ 1
90 OLED TILE FLAGS 1 1
100 'Diagonal tile player
110 D$=""
120 FOR Y=0 TO 15
130  FOR X=0 TO 15
140    IF X=Y THEN D$=D$+"1"
150    IF X<>Y THEN D$=D$+"0"
160  NEXT X
170 NEXT Y
180 OLED TILE NEW 2 16 16 D$ 1
190 OLED TILEMAP NEW 0 20 20 16 16 1 0
200 'row floor 12
210 FOR C=0 TO 19
220   OLED TILEMAP SET 0 C 12 1
230 NEXT C
240 PX=64: PY=0: VX=0: VY=0
250 OLD_C=-1: OLD_R=-1
260 VY=VY+1
270 OLED TILEMAP MOVE 0 RECT PX PY 16 16 BY VX VY FLAGS 1 SET PX PY HX HY GR
280 IF GR<>0 THEN VY=0
290 IF OLD_C>=0 THEN OLED TILEMAP SET 0 OLD_C OLD_R 0
300 C=INT(PX/16)
310 R=INT(PY/16)
320 OLED TILEMAP SET 0 C R 2
330 OLD_C=C: OLD_R=R
340 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
350 GOTO 260
```

Gravity + SOLID platform (falls and stops)

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'Solid SOLID wall
40 W$=""
50 FOR I=1 TO 256
60   W$=W$+"1"
70 NEXT I
```

```
80 OLED TILE NEW 1 16 16 W$ 1
90 OLED TILE FLAGS 1 1
100 'Full player
110 P$=""
120 FOR I=1 TO 256
130   P$=P$+"1"
140 NEXT I
150 OLED TILE NEW 2 16 16 P$ 1
160 OLED TILEMAP NEW 0 20 20 16 16 1 0
170 FOR R=0 TO 19
180   OLED TILEMAP SET 0 8 R 1
190 NEXT R
200 PX=0: PY=16: VX=2: VY=0
210 OLD_C=-1: OLD_R=-1
220 OLED TILEMAP MOVE 0 RECT PX PY 16 16 BY VX VY FLAGS 1 SET PX PY HX HY GR
230 IF HX<>0 THEN VX=0-VX
240 IF OLD_C>=0 THEN OLED TILEMAP SET 0 OLD_C OLD_R 0
250 C=INT(PX/16)
260 R=INT(PY/16)
270 OLED TILEMAP SET 0 C R 2
280 OLD_C=C: OLD_R=R
290 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
300 GOTO 220
```

Horizontal movement + rebound on SOLID wall

## Note

- FLAGS <mask> decides which tiles it collides with (e.g. SOLID=1, PLATFORM=2).
- dx dy are in pixels per frame.

# OLED TILEMAP NEW

## Sintassi

```
OLED TILEMAP NEW <mapId> <cols> <rows> <tileW> <tileH> [scale] [fill]
```

## Descrizione

Creates a tilemap (grid of cells) made up of tiles. , : dimensions in cells , : tile size in pixels (typical: 16×16) [scale]: rendering scale of the tilemap (default 1) [fill]: initial value of all cells (default 0)

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 P$=""
40 FOR I=1 TO 256
50   P$=P$+"1"
60 NEXT I
70 OLED TILE NEW 1 16 16 P$ 1
80 OLED TILEMAP NEW 0 8 4 16 16 1 1
90 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
```

8×4 map filled with tiles 1

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 P$=""
40 FOR I=1 TO 256
50   P$=P$+"1"
60 NEXT I
70 OLED TILE NEW 1 16 16 P$ 1
80 OLED TILEMAP NEW 0 8 4 16 16 1 0
90 'top/bottom edge
100 FOR C=0 TO 7
110   OLED TILEMAP SET 0 C 0 1
120   OLED TILEMAP SET 0 C 3 1
130 NEXT C
140 'left/right border
150 FOR R=0 TO 3
160   OLED TILEMAP SET 0 0 R 1
170   OLED TILEMAP SET 0 7 R 1
180 NEXT R
190 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
```

Empty 8×4 map, then “framed” by hand

## Note

- Recreate the same free mapId and recreate the map.
- initial fill is useful for filling everything with a basic tile.

# OLED TILEMAP RAYCAST

## Sintassi

```
OLED TILEMAP RAYCAST <mapId> <x1> <y1> <x2> <y2> FLAGS <mask>
SET <hit> <x> <y> <col> <row> <tid>
```

## Descrizione

Cast a “ray” (line) from (x1,y1) to (x2,y2) in pixel world and find the first tile along the path that matches the mask. Returns: hit: 1 if hit, 0 if not x y: point of impact (approximate) in pixel world col row: cell hit tid: tileId hit

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'Solid SOLID wall
40 W$=""
50 FOR I=1 TO 256
60   W$=W$+"1"
70 NEXT I
80 OLED TILE NEW 1 16 16 W$ 1
90 OLED TILE FLAGS 1 1
100 'Diagonal marker
110 D$=""
120 FOR Y=0 TO 15
130  FOR X=0 TO 15
140    IF X=Y THEN D$=D$+"1"
150    IF X<>Y THEN D$=D$+"0"
160  NEXT X
170 NEXT Y
180 OLED TILE NEW 2 16 16 D$ 1
190 OLED TILEMAP NEW 0 20 20 16 16 1 0
200 FOR R=0 TO 19
210   OLED TILEMAP SET 0 10 R 1
220 NEXT R
230 X1=0: Y1=32
240 X2=319: Y2=32
250 OLED TILEMAP RAYCAST 0 X1 Y1 X2 Y2 FLAGS 1 SET HIT HX HY HC HR HT
260 'I highlight the affected cell if there is a hit
270 IF HIT<>0 THEN OLED TILEMAP SET 0 HC HR 2
280 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
290 GOTO 290
```

Horizontal ray to a SOLID wall (shows the affected cell)

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'SOLID floor
40 P$=""
50 FOR I=1 TO 256
60   P$=P$+"1"
70 NEXT I
80 OLED TILE NEW 1 16 16 P$ 1
90 OLED TILE FLAGS 1 1
100 'Marker full
110 M$=""
120 FOR I=1 TO 256
130   M$=M$+"1"
```

```
140 NEXT I
150 OLED TILE NEW 2 16 16 M$ 1
160 OLED TILEMAP NEW 0 20 20 16 16 1 0
170 FOR C=0 TO 19
180   OLED TILEMAP SET 0 C 12 1
190 NEXT C
200 X1=0
210 X1=X1+2
220 IF X1>319 THEN X1=0
230 Y1=0: X2=X1: Y2=319
240 OLED TILEMAP RAYCAST 0 X1 Y1 X2 Y2 FLAGS 1 SET HIT HX HY HC HR HT
250 IF HIT<>0 THEN OLED TILEMAP SET 0 HC HR 2
260 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
270 GOTO 210
```

Vertical “probe” ray and point move

## Note

- Excellent for “line of sight”, shots, lasers, sensors.
- The mask filters which tiles matter (SOLID=1, HAZARD=32, etc.).

# OLED TILEMAP SET

## Sintassi

```
OLED TILEMAP SET <mapId> <col> <row> <tileId>
```

## Descrizione

Set the tile of a cell in the tilemap.

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 P$=""
40 FOR I=1 TO 256
50   P$=P$+"1"
60 NEXT I
70 OLED TILE NEW 1 16 16 P$ 1
80 OLED TILEMAP NEW 0 8 4 16 16 1 0
90 'X on the 4 lines: (0,0)(7,0) (1,1)(6,1) (1,2)(6,2) (0,3)(7,3)
100 OLED TILEMAP SET 0 0 0 1
110 OLED TILEMAP SET 0 7 0 1
120 OLED TILEMAP SET 0 1 1 1
130 OLED TILEMAP SET 0 6 1 1
140 OLED TILEMAP SET 0 1 2 1
150 OLED TILEMAP SET 0 6 2 1
160 OLED TILEMAP SET 0 0 3 1
170 OLED TILEMAP SET 0 7 3 1
180 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
```

Draw a large “X” made of cells

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 A$="": FOR I=1 TO 256: A$=A$+"1": NEXT I
40 'I'll rewrite without ":" to be safe
50 A$=""
60 FOR I=1 TO 256
70   A$=A$+"1"
80 NEXT I
90 OLED TILE NEW 1 16 16 A$ 1
100 B$=""
110 FOR Y=0 TO 15
120  FOR X=0 TO 15
130    S=X+Y
140    T=S-(INT(S/2)*2)
150    IF T=0 THEN B$=B$+"1"
160    IF T<>0 THEN B$=B$+"0"
170  NEXT X
180 NEXT Y
190 OLED TILE NEW 2 16 16 B$ 1
200 OLED TILEMAP NEW 0 8 4 16 16 1 0
210 FOR R=0 TO 3
220   FOR C=0 TO 7
230     S=C+R
240     T=S-(INT(S/2)*2)
250     IF T=0 THEN OLED TILEMAP SET 0 C R 1
260     IF T<>0 THEN OLED TILEMAP SET 0 C R 2
270   NEXT C
```

```
280 NEXT R
290 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
```

Checkerboard (alternating cells 1 and 2)

## Note

- If an animation was active on the cell, SET deactivates it (that cell returns to “static”).

# OLED TILEMAP TILEAT

## Sintassi

```
OLED TILEMAP TILEAT <mapId> <wx> <wy> SET <col> <row> <tid> <flags>
```

## Descrizione

Given a world point (wx, wy) (in pixels), returns: col row: the map cell tid: tileId present in that cell flags: tile flag

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'Tile 1 filled (cell marked)
40 P$=""
50 FOR I=1 TO 256
60   P$=P$+"1"
70 NEXT I
80 OLED TILE NEW 1 16 16 P$ 1
90 OLED TILEMAP NEW 0 8 4 16 16 1 0
100 WX=0: WY=0
110 OLD_C=-1: OLD_R=-1
120 WX=WX+4
130 IF WX>127 THEN WX=0
140 WY=32
150 OLED TILEMAP TILEAT 0 WX WY SET TC TR TT TF
160 IF OLD_C>=0 THEN OLED TILEMAP SET 0 OLD_C OLD_R 0
170 OLED TILEMAP SET 0 TC TR 1
180 OLD_C=TC: OLD_R=TR
190 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
200 GOTO 120
```

It scans a moving point and lights a “probe” in the cell being read

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'Water tile + WATER flag
40 W$=""
50 FOR Y=0 TO 15
60  FOR X=0 TO 15
70    A=X+Y
80    B=A-(INT(A/2)*2)
90    IF B=0 THEN W$=W$+"1"
100   IF B<>0 THEN W$=W$+"0"
110  NEXT X
120 NEXT Y
130 OLED TILE NEW 1 16 16 W$ 1
140 OLED TILE FLAGS 1 8
150 'Full indicator tile
160 P$=""
170 FOR I=1 TO 256
180   P$=P$+"1"
190 NEXT I
200 OLED TILE NEW 2 16 16 P$ 1
210 OLED TILEMAP NEW 0 8 4 16 16 1 0
220 'I put a water stripe on row 2
230 FOR C=0 TO 7
240   OLED TILEMAP SET 0 C 2 1
```

```
250 NEXT C
260 WX=0: WY=32
270 OLD_C=-1: OLD_R=-1
280 WX=WX+2
290 IF WX>127 THEN WX=0
300 OLED TILEMAP TILEAT 0 WX WY SET TC TR TT TF
310 OLED TILEMAP SET 0 0 0 0
320 IF TF=8 THEN OLED TILEMAP SET 0 0 0 2
330 IF OLD_C>=0 THEN OLED TILEMAP SET 0 OLD_C OLD_R TT
340 OLD_C=TC: OLD_R=TR
350 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
360 GOTO 280
```

Use TILEAT to understand if the point is on WATER (8)

## Note

- It is convenient for "sensor" under the player's feet, checking terrain, water, stairs...
- wx/wy are pixel world, so if you use camera, keep this in mind.

# OLED TILEMAP TRIGGER

## Sintassi

```
OLED TILEMAP TRIGGER <mapId> RECT <x> <y> <w> <h> FLAGS <mask>
SET <hit> <tid> <col> <row>
```

## Descrizione

Checks if a rectangle (pixel world) is touching a tile with flags matching , and returns information about the first cell found. Returns: hit: 1 if it finds at least one valid tile, otherwise 0 tid: tileId of the hit cell with row: hit cell coordinates

## Esempi

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'Tile trigger: checkerboard + TRIGGER
40 T$=""
50 FOR Y=0 TO 15
60  FOR X=0 TO 15
70    A=X+Y
80    B=A-(INT(A/2)*2)
90    IF B=0 THEN T$=T$+"1"
100   IF B<>0 THEN T$=T$+"0"
110  NEXT X
120 NEXT Y
130 OLED TILE NEW 1 16 16 T$ 1
140 OLED TILE FLAGS 1 128
150 'Tile player: diagonal
160 P$=""
170 FOR Y=0 TO 15
180  FOR X=0 TO 15
190    IF X=Y THEN P$=P$+"1"
200    IF X<>Y THEN P$=P$+"0"
210  NEXT X
220 NEXT Y
230 OLED TILE NEW 2 16 16 P$ 1
240 'Tile marker: full
250 M$=""
260 FOR I=1 TO 256
270   M$=M$+"1"
280 NEXT I
290 OLED TILE NEW 3 16 16 M$ 1
300 OLED TILEMAP NEW 0 20 20 16 16 1 0
310 OLED TILEMAP SET 0 6 1 1
320 PX=0: PY=16
330 OLD_C=-1: OLD_R=-1
340 ' loop
350 OLED TILEMAP TRIGGER 0 RECT PX PY 16 16 FLAGS 128 SET HIT TID TC TR
360 OLED TILEMAP SET 0 0 0 0
370 IF HIT<>0 THEN OLED TILEMAP SET 0 0 0 3
380 IF OLD_C>=0 THEN OLED TILEMAP SET 0 OLD_C OLD_R 0
390 C=INT(PX/16)
400 R=INT(PY/16)
410 OLED TILEMAP SET 0 C R 2
420 OLD_C=C: OLD_R=R
430 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
440 PX=PX+1
```

```
450 IF PX>304 THEN PX=0
460 GOTO 350
```

TRIGGER area: when you hover over it, a marker appears at the top left

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 'Tile door: full + DOOR
40 D$=""
50 FOR I=1 TO 256
60   D$=D$+"1"
70 NEXT I
80 OLED TILE NEW 1 16 16 D$ 1
90 OLED TILE FLAGS 1 16
100 'Tile player: diagonal
110 P$=""
120 FOR Y=0 TO 15
130  FOR X=0 TO 15
140    IF X=Y THEN P$=P$+"1"
150    IF X<>Y THEN P$=P$+"0"
160  NEXT X
170 NEXT Y
180 OLED TILE NEW 2 16 16 P$ 1
190 'Alternate background tile: chessboard
200 S$=""
210 FOR Y=0 TO 15
220  FOR X=0 TO 15
230    A=X+Y
240    B=A-(INT(A/2)*2)
250    IF B=0 THEN S$=S$+"1"
260    IF B<>0 THEN S$=S$+"0"
270  NEXT X
280 NEXT Y
290 OLED TILE NEW 3 16 16 S$ 1
300 OLED TILEMAP NEW 0 20 20 16 16 1 0
310 OLED TILEMAP SET 0 8 1 1
320 PX=0: PY=16
330 OLD_C=-1: OLD_R=-1
340 MODE=0
350 OLED TILEMAP TRIGGER 0 RECT PX PY 16 16 FLAGS 16 SET HIT TID TC TR
360 IF HIT<>0 THEN MODE=1
370 IF MODE=0 THEN OLED TILEMAP FILL 0 0
380 IF MODE<>0 THEN OLED TILEMAP FILL 0 3
390 OLED TILEMAP SET 0 8 1 1
400 IF OLD_C>=0 THEN OLED TILEMAP SET 0 OLD_C OLD_R 0
410 C=INT(PX/16)
420 R=INT(PY/16)
430 OLED TILEMAP SET 0 C R 2
440 OLD_C=C: OLD_R=R
450 OLED TILEMAP DRAW 0 0 0 CAMERA 0 0 1
460 PX=PX+1
470 IF PX>304 THEN PX=0
480 GOTO 350
```

“Door” trigger: when you tap DOOR (16) you change the background (FILL)

## Note

- TRIGGER is like a “sensor”: it doesn't move anything, it just tells you if you're touching a type of tile.

• The mask is a sum of flags: e.g. DOOR + TRIGGER = 16 + 128 = 144.

# ON x GOTO ...

## Sintassi

```
ON espressione GOTO riga1, riga2, riga3, ...
ON espressione GOSUB riga1, riga2, riga3, ...
```

## Descrizione

ON x GOTO (or ON x GOSUB) performs a conditional jump to the line corresponding to position x. It works like a numeric menu or switch-case, selecting the line to execute based on the value of x.

```
If x = 1, jump to the first line listed
If x = 2, jump to the second line, and so on
```

If x is out of range, nothing happens

## Esempi

```
10 INPUT X
20 ON X GOTO 100, 200, 300
30 PRINT "NO VALID CHOICE"
40 END
100 PRINT "CHOSEN 1": END
200 PRINT "CHOSEN 2": END
300 PRINT "CHOSEN 3": END

Expected output (if input = 2):
CHOSEN 2
```

Conditional GOTO

```
10 INPUT N
20 ON N GOSUB 100, 200
30 PRINT "RETURN TO THE MAIN"
40 END
100 PRINT "FUNCTION A": RETURN
200 PRINT "FUNCTION B": RETURN

Expected output (if input = 1):
FUNCTION A
RETURN TO THE MAIN
```

Conditional GOSUB

```
10 X = 0
20 ON X GOTO 100, 200
30 PRINT "X INVALID"

Expected output:
X INVALID
If x is zero or greater than the number of options, the jump does not occur
```

Value out of range

```
10 A = INT(RND(1) * 3) + 1
20 ON A GOTO 100, 200, 300
100 PRINT "PRIMO": END
200 PRINT "SECONDO": END
300 PRINT "TERZO": END

Expected output:
```

```
One of the three lines is performed randomly.
```

Use with numerical variable from calculation

## Note

- ON ... GOTO can have 1 to 255 jumps indicated
- If you use ON ... GOSUB, always remember the RETURN at the end of the subroutine

# OPEN

## Sintassi

```
OPEN [SPIFFS | SD] "<filename>" <mode>
```

## Descrizione

Opens a text file on the chosen filesystem: SPIFFS → internal memory of the ESP32 SD → SD card If you do not specify anything → default SPIFFS can be: If another file was already open, it is automatically closed.

## Esempi

```
10 OPEN SPIFFS "log.txt" W
20 FPRINT "CIAO"
30 FPRINT "VALORE=42"
40 FCLOSE
```

Write a file

```
10 OPEN SD "history.txt" A
20 FPRINT "New line"
30 FCLOSE
```

Append on SD

```
10 OPEN "notes.txt" W
20 FPRINT "Hello!"
30 FCLOSE
```

Default SPIFFS

## Note

- Only one file can be open at a time.
- Mode A also allows reading before append.
- The filename is always a quoted STRING.
- If the filesystem is not initialized (e.g. SD not mounted), an error is generated.

# PEEK

## Sintassi

```
PEEK(indirizzo)
```

## Descrizione

The PEEK function reads the numerical value (byte) contained in a specific memory location. It is the complement of POKE, used to get what was previously written or to read areas of mapped memory. The returned value is between 0 and 255.

## Esempi

```
10 POKE 1000, 88
20 PRINT "ADDRESS VALUE 1000:"; PEEK(1000)

Expected output:
ADDRESS VALUE 1000: 88
```

Reading a value after POKE

```
10 PRINT PEEK(2000)

Expected output:
Value currently stored at address 2000.
```

Direct reading if a value had previously been written

```
10 FOR I = 0 TO 4
20 POKE 3000 + I, I * 2
30 NEXT I
40 FOR I = 0 TO 4
50 PRINT "PEEK("; 3000 + I; ") = "; PEEK(3000 + I)
60 NEXT I

Expected output:
PEEK(3000) = 0
PEEK(3001) = 2
PEEK(3002) = 4
PEEK(3003) = 6
PEEK(3004) = 8
```

Reading cycle

```
10 PRINT "VALUE:"; PEEK(5000)

Expected output:
The value depends on the state of the memory (it could be 0 or other).
```

Reading an empty area

## Note

- It is always good to make sure that the address read is valid in the context of the firmware
- To change a value in memory, use POKE

# PINMODE

## Sintassi

```
PINMODE pin INPUT|OUTPUT NOPULL|PULLUP|PULLDOWN  DEBOUNCE ms
```

## Descrizione

Configure an ESP32 GPIO: mode: INPUT (input) or OUTPUT (output) resistance: NOPULL, PULLUP, PULLDOWN DEBOUNCE [ms] (optional, only for INPUT): enables one-shot debounce for DREAD(pin). With DEBOUNCE, DREAD(pin) generates only one event per press and does not repeat until you release. Without DEBOUNCE, DREAD(pin) returns the raw layer and can loop while held. ms is the stabilization time (default 50 ms if omitted).

```
With PULLUP: pressed = LOW (0), released = HIGH (1).
With PULLDOWN: pressed = HIGH (1), released = LOW (0).
```

DLEVEL(pin) always returns the raw level, ignoring debounce.

## Esempi

```
10 PINMODE 2 OUTPUT NOPULL
20 DWRITE 2 1
```

OUTPUT without resistors, lights an LED on GPIO2

```
10 PINMODE 12 INPUT PULLUP DEBOUNCE 60
20 IF DREAD(12) = 0 THEN PRINT “PULSANTE PREMUTO”
30 GOTO 20
```

INPUT with PULLUP and DEBOUNCE (one-shot), count one press at a time on the GPIO5

```
10 PINMODE 12 INPUT PULLUP
20 IF DREAD(12) = 0 THEN PRINT "PULSANTE PREMUTO”
30 GOTO 20
```

INPUT with PULLUP without DEBOUNCE (raw), repeat as long as you hold

```
10 PINMODE 12 INPUT PULLUP DEBOUNCE 60
20 IF DLEVEL(12) = 0 THEN PRINT "HOLD" ELSE PRINT "UP"
30 DELAY 500
40 GOTO 20
```

Instant status with DLEVEL (ignore debounce), shows “held down” in real time

```
10 PINMODE 12 INPUT PULLDOWN DEBOUNCE 50
20 IF DREAD(12) = 1 THEN PRINT "EVENTO"
30 GOTO 20
```

PULLDOWN + DEBOUNCE (one-shot on 1), with sensor active-high

## Note

- Always use PINMODE before DREAD, DLEVEL, DWRITE.
- DEBOUNCE applies only to INPUT and modifies the behavior of DREAD(pin) (one-shot); DLEVEL(pin) remains raw.
- The ms value (if specified) sets the debounce window; if omitted, 50 ms.
- On some boards PULLDOWN hardware may not be available on all pins: in this case use external resistors.

# POKE

## Sintassi

```
POKE indirizzo, valore
```

## Descrizione

The POKE command writes a numeric value (1 byte) to an internal BASIC virtual memory, dedicated to the use of PEEK and POKE. This memory does not correspond to the real RAM of the ESP32, but to an area protected and controlled by the BASIC firmware, designed to: simulate the behavior of classic BASICs (C64, ZX, etc.) create temporary buffers store binary data or simple structures avoid crashes or invalid accesses to the microcontroller's memory Address parameters Integer number indicating the memory location to be modified. Must be between 0 and 511. value Integer between 0 and 255. It is written as a byte in the indicated location. If the address is outside the allowed range, the command generates a runtime error. Available Memory The BASIC virtual memory for POKE and PEEK is:

```
512 bytes (addresses 0 to 511)
```

Each address contains a single byte.

## Esempi

```
10 POKE 100, 42

Writes the value 42 to memory location 100.
No video output.
```

Writing a value

```
10 POKE 200, 77
20 PRINT "VALUE IN MEMORY:"; PEEK(200)

Expected output
MEMORY VALUE: 77
```

Writing followed by reading with PEEK

```
10 FOR I = 0 TO 9
20 POKE I, I * 2
30 NEXT I

Outputs:
ADDRESS 0 = 0
ADDRESS 1 = 2
ADDRESS 2 = 4
...
ADDRESS 9 = 18
```

Use in a loop, fills a series of contiguous locations

```
10 FOR I = 0 TO 4
20 POKE 100 + I, I * 10
30 NEXT I
40 FOR I = 0 TO 4
50 PRINT PEEK(100 + I)
60 NEXT I

Expected output:
0
```

```
10
20
30
40
```

Buffer simulation

```
POKE 600, 1
Errore:
POKE: address out of range (0-511)
```

Address out of range

## Note

- POKE operates only on BASIC virtual memory
- The ESP32's RAM, registers, or peripherals cannot be accessed
- This ensures stability and security of the firmware
- To read the written values, use the companion command PEEK

# PRINT

## Sintassi

```
PRINT espressione
PRINT variabile [,|;] ...
```

## Descrizione

```
PRINT displays text, numbers, results of expressions or variables on the screen (on the seri
```

It is one of the fundamental commands for: showing messages viewing results debugging

```
Can print strings ("Text"), numbers (123, A), combinations ("X = ";
; → print on the same line without space
```

, → align to next column

## Esempi

```
10 PRINT "HELLO WORLD"

Expected output:
HELLO WORLD
```

Simple printing of a string

```
10 A = 5
20 PRINT "VALUE OF A:"; A

Expected output:
A VALUE: 5
```

Printing a number and text

```
10 PRINT "X", "Y", "Z"

Expected output:
X Y Z
```

Printing on columns with comma,

```
10 PRINT "2 + 3 = "; 2 + 3

Expected output:
2 + 3 = 5
```

Expression print

```
10 A$ = "LUCA"
20 PRINT "CIAO " + A$

Expected output:
HELLO LUCA
```

Printing concatenated strings

## Note

- PRINT can be abbreviated with ? (e.g. ? "HELLO")
- To wrap, use PRINT without arguments

# PRINTUSING

## Sintassi

```
PRINTUSING "maschera"; espressione
```

## Descrizione

The PRINTUSING command prints the result of an expression using a simple formatting mask. In your interpreter: The mask is a quoted string, such as "0.00", "###.###", "VALUE: 0.000". Only the number of decimals is obtained from the mask (counting the digits after the point . or comma ,). If the expression is numeric, the value is printed with that number of decimals. If the expression is string, the mask is ignored and the string is printed as is. At the end, a newline is always sent (equivalent to PRINT alone). Does not handle: multiple values on the same PRINTUSING complex alignments fillings with #, +, *, etc. It is a simplified formatted PRINT to fix decimals.

## Esempi

```
10 X = 12.3456
20 PRINTUSING "0.00"; X

Expected output:
12.35
(the mask has 2 decimal places, so it prints with 2 decimals)
```

Number with two decimals

```
10 PI = 3.14159
20 PRINTUSING "###.000"; PI

Expected output:
3.142
(the mask has 3 decimal places after the point)
```

Number with three decimals

```
10 VALORE = 7.5
20 PRINT "RESULT:"
30 PRINTUSING "0.0000"; VALORE

Expected output:
RESULT:
7.5000
```

Text + number formatted

```
10 VALORE = 123.456
20 PRINTUSING "VALUE = 0.00"; VALORE

Expected output:
VALUE = 123.46
```

Form with fixed text (managed by the program)

```
10 PRINTUSING "0.000"; 10 / 3

Expected output:
3.333
```

Expression instead of variable

```
10 S$ = "CIAO"
```

```
20 PRINTUSING "0.00"; S$

Expected output:
CIAO
```

String expression, if the expression is string, the mask is ignored and the string is printed

## Note

- The mask must be a quoted string after PRINTUSING.
- Only the number of decimal places is used by the mask to format the numbers.
- If the value is not numeric, it is printed as a string, without formatting.
- After each PRINTUSING a CRLF (new line) is output, as PRINT.

# READ

## Sintassi

```
READ variabile
```

## Descrizione

The READ command takes a value from a sequence defined by one or more DATA commands. It is used to load predefined data into the program in an orderly and sequential manner. Each call to READ extracts the next value from the DATA list. DATA values can be numbers or strings, separated by commas. To reinitialize data reading, use RESTORE.

## Esempi

```
10 READ A
20 READ B
30 PRINT A + B
40 DATA 3, 7

Expected output:
10
```

Reading numbers from DATA

```
10 READ NOME$
20 PRINT "CIAO "; NOME$
30 DATA "Luca"

Expected output:
HELLO Luca
```

Reading strings

```
10 FOR I = 1 TO 3
20 READ X
30 PRINT "VALORE "; I; ": "; X
40 NEXT I
50 DATA 10, 20, 30

Expected output:
VALUE 1: 10
VALUE 2: 20
VALUE 3: 30
```

Reading in a loop

```
10 READ A
20 READ B
30 PRINT A, B
40 RESTORE
50 READ C
60 PRINT C
70 DATA 1, 2

Expected output:
1       2
1
```

Using RESTORE to restart reading

### Note

- READ commands always read in the order defined by DATA
- You can place DATA on any line, even after READ

# REBOOT

## Sintassi

```
REBOOT
```

## Descrizione

The REBOOT command forces a complete reboot of the ESP32 microcontroller. It is useful for: Restoring the initial state Applying permanent changes Exiting error conditions Simulating a "power reset" from software It runs immediately and interrupts any running program.

## Esempi

```
10 INPUT "ARE YOU SURE TO RESTART? (1 = YES)"; A
20 IF A = 1 THEN REBOOT

Expected output:
If the user enters 1, the ESP32 restarts.
```

Reboot after confirmation

```
10 PRINT "INSTALLATION COMPLETED"
20 PRINT "I WILL RESTART IN 5 SECONDS..."
30 DELAY 5000
40 REBOOT

Expected output:
Message followed by automatic restart.
```

Use in an installer

## Note

- No automatic saving occurs: save any important data first
- The REBOOT command has no parameters and returns no output

# RENAME

## Sintassi

```
RENAME "vecchio" "nuovo"
RENAME variabile1$ variabile2$
RENAME SD "vecchio" "nuovo"
RENAME SD variabile1$ variabile2$
RENAME SPIFFS "vecchio" "nuovo"
RENAME SPIFFS variabile1$ variabile2$
```

## Descrizione

The RENAME command renames a file in both SPIFFS internal storage and the SD card, depending on the specified argument. Without memory arguments, it acts on SPIFFS (internal memory). With SD, it renames files on the SD card. It accepts both direct names in quotes and string variables or expressions (A$, LEFT$(N$,3)+".bas"). It does not modify the contents of the files: it is an operation only on the names.

## Esempi

```
RENAME "boot.bas" "boot_old.bas"

Output atteso:
File renamed on SPIFFS.
```

Rename to SPIFFS (default)

```
RENAME SD "log.txt" "log_old.txt"

Output atteso:
File renamed on SD.
```

Rename on SD

```
10 F$ = "data.csv"
20 N$ = "data_2025.csv"
30 RENAME SD F$ N$

Expected output:
File renamed on SD.
```

With string variables

```
RENAME "missing.txt" "whatever.txt"

Output atteso:
FILE RENAME: Rename failed on SPIFFS.
```

Non-existent file

## Note

- If no memory is specified, the operation is performed on SPIFFS.
- The command does not create folders and does not move files between different directories: it renames only within the indicated path.
- File names are case-sensitive depending on the file system.
- In case of error (missing file, permissions, invalid name), FILE RENAME is shown: Rename failed.

# RESTORE

## Sintassi

```
RESTORE
```

## Descrizione

The RESTORE command returns the READ pointer to the beginning of the DATA, allowing the data to be read again from the beginning. This is useful when you want to reuse the same data in a second reading cycle. RESTORE takes no arguments and always acts on all DATA, starting again from the first available entry.

## Esempi

```
10 READ A
20 READ B
30 PRINT "FIRST READING:", A, B
40 RESTORE
50 READ X
60 PRINT "AFTER RESTORE:", X
70 DATA 5, 10

Expected output:
FIRST READING: 5 10
AFTER RESTORE: 5
```

Reading and repeating data

```
10 FOR I = 1 TO 2
20 RESTORE
30 READ A, B
40 PRINT "CICLO"; I; ": "; A; B
50 NEXT I
60 DATA 1, 2

Expected output:
CYCLE1: 1 2
CYCLE2: 1 2
```

Use in a loop to read the same sequence twice

```
10 FOR I = 1 TO 3
20 READ X
30 PRINT "X ="; X
40 NEXT I
50 RESTORE
60 READ Y
70 PRINT "Y after RESTORE ="; Y
80 DATA 10, 20, 30

RUN
Expected output:
X = 10
X = 20
X = 30
Y after RESTORE = 10
```

Combination with FOR...NEXT

```
10 READ A
```

```
20 READ B
30 READ C
40 PRINT A, B, C
50 DATA 1, 2

Expected output:
Error or unexpected value, because DATA only has 2 elements.
```

Incorrect reading without RESTORE

## Note

- If you have multiple DATA blocks in different rows, RESTORE starts again from the first available one
- You cannot aim for an intermediate position (there is no RESTORE n)

# RETURN

## Sintassi

```
RETURN
```

## Descrizione

The RETURN command signals the end of a subroutine started with GOSUB. When executed, the program returns to the line immediately following the one from which GOSUB was called. Each GOSUB must have a corresponding RETURN, otherwise the program does not return correctly to the main flow.

## Esempi

```
10 GOSUB 100
20 PRINT "RETURN TO THE MAIN"
30 END
100 PRINT "SUBROUTINE"
110 RETURN

Expected output:
SUBROUTINE
RETURN TO THE MAIN
```

Simple subroutine

```
10 A = 5: B = 7
20 GOSUB 100
30 PRINT "SUM:"; R
40 END
100 R = A + B
110 RETURN

Expected output:
SUM: 12
```

Use with indirect parameters (global variables)

```
10 INPUT X
20 ON X GOSUB 100, 200
30 PRINT "FINE"
40 END
100 PRINT "CHOICE 1": RETURN
200 PRINT "CHOICE 2": RETURN

Expected output (e.g. input 1):
CHOICE 1
FINE
```

Use with ON x GOSUB

```
10 GOSUB 100
20 PRINT "THIS IS NOT EXECUTED"
100 PRINT "FAILURE TO RETURN"

Expected output:
FAILURE TO RETURN
```

Error if RETURN is missing

### Note

- Subroutines can be nested, but each GOSUB must end with a RETURN
- There can be more than one RETURN within conditions (IF) for flexible subroutines

# RFID HALT

## Sintassi

```
RFID HALT
```

## Descrizione

Stops communication with the currently selected tag, freeing it for new readings. It is good practice to call it after reading or writing a tag.

## Esempi

```
10 RFID INIT 5 27
20 RFID WAITUID U$
30 PRINT "Found UID=";U$
40 RFID HALT

Expected output:
Found UID=04A1B2C3D4
```

Read UID and release

```
10 RFID INIT 5 27
20 FOR I=1 TO 3
30   PRINT "Bring a tag closer..."
40   RFID WAITUID U$
50   PRINT "UID=";U$
60   RFID HALT
70 NEXT I

Expected output:
Bring a tag closer...
UID=1122334455
Bring a tag closer...
UID=8899AABBCC
...
```

Use in a cycle

## Note

- It is not mandatory, but useful to avoid multiple readings of the same tag without removing it.
- Call it before waiting for a new tag (WAITUID).

# RFID INIT (MFRC522)

## Sintassi

```
RFID INIT ss rst
RFID INIT ss rst mosi miso sck
```

## Descrizione

Initializes the MFRC522 RFID module. Parameters: ss → RFID module SDA / SS pin rst → module RST pin Optional SPI parameters: mosi, miso, sck If SPI pins are not specified, ESP32 defaults are used:

```
MOSI = 23
MISO = 19
SCK = 18
```

## Esempi

```
10 RFID INIT 5 27
20 PRINT "RFID ready"
```

Basic initialization

```
10 RFID INIT 5 27 23 19 18
20 PRINT "RFID ready"
```

Initialization with custom SPI pins

```
10 PINMODE 17 OUTPUT NOPULL   'CS display
20 DWRITE 17 1
30 PINMODE 4 OUTPUT NOPULL    'CS touch
40 DWRITE 4 1
50 RFID INIT 5 27
60 PRINT "RFID ready"
```

With other SPI devices

## Note

- Must be called only once at startup.
- If not called, all other RFID commands fail.
- If you share SPI, make sure other CSs are HIGH.

# RFID PRESENT

## Sintassi

```
RFID PRESENT
```

## Descrizione

Returns 1 if a tag is currently in the reader's field, 0 otherwise. Useful for quick checks without blocking.

## Esempi

```
10 RFID INIT 5 27
20 PRINT RFID PRESENT

Expected output:
1 (se presente), 0 (se assente)
```

Print status

```
10 RFID INIT 5 27
20 IF RFID PRESENT=1 THEN PRINT "Tag detected!"
30 GOTO 20
```

Polling loop

## Note

- It is non-blocking, great for loops that do multiple things at once.

# RFID READBLOCKABS

## Sintassi

```
RFID READBLOCKABS block var$ keyType keyHex
```

## Descrizione

Reads 16 bytes from an absolute block of a MIFARE Classic card. Result in var$ as a HEX string (32 characters).

## Esempi

```
10 RFID INIT 5 27
20 PRINT "Bring card"
30 RFID WAITUID U$
40 RFID READBLOCKABS 4 D$ A "FFFFFFFFFFFF"
50 PRINT "Blocco4=";D$
60 RFID HALT

Expected output:
Blocco4=00112233445566778899AABBCCDDEEFF
```

Reading a block

```
10 RFID INIT 5 27
20 RFID WAITUID U$
30 RFID READBLOCKABS 4 DATA$ A "FFFFFFFFFFFF"
40 IF DATA$="CAFEBABE00000000000000000000000" THEN PRINT "TAG OK"
```

Check data in bulk

## Note

- block starts from 0; do not use trailer blocks (every 4th block).
- WAITUID or READUID required first.

# RFID READUID

## Sintassi

```
RFID READUID var$
```

## Descrizione

```
Reads the UID of the tag if present at that moment, otherwise var$="".
```

Does not block the program.

## Esempi

```
10 RFID INIT 5 27
20 RFID READUID U$
30 IF U$<>"" THEN PRINT "UID=";U$ ELSE PRINT "No Tags"
```

Polling UID

```
10 RFID INIT 5 27
20 FOR I=1 TO 10
30   RFID READUID U$
40   IF U$<>"" THEN PRINT "Trovato ";U$
50   WAIT 500
60 NEXT I
```

Used in a timed loop

## Note

- Does not replace WAITUID if you want to wait for the tag.

# RFID WAITUID

## Sintassi

```
RFID WAITUID var$ [timeout_ms] [GOTO line]
```

## Descrizione

Waits for a tag to be presented and puts the UID in var$. If timeout_ms expires and there is a GOTO line, it does not give an error: it jumps to the indicated line. If timeout_ms expires without GOTO, it gives no error: var$ remains "" and the program continues.

## Esempi

```
10 RFID INIT 5 27
20 PRINT "Bring a tag closer..."
30 RFID WAITUID U$
40 PRINT "UID=";U$
```

Indefinite wait

```
10 RFID INIT 5 27
20 RFID WAITUID U$ 3000
30 IF U$="" THEN PRINT "Timeout" ELSE PRINT "UID=";U$
```

3s timeout (handled in IF)

```
10 RFID INIT 5 27
20 PRINT "Has a tag (3s)..."
30 RFID WAITUID U$ 3000 GOTO 80
40 PRINT "UID=";U$
50 GOTO 100
80 PRINT "No tags within the time"
100 END
```

3s timeout with jump

## Note

- Block until a tag arrives or timeout expires.
- GOTO is optional and applies only to the case of timeout.
- var$ is a string variable (must end with $).

# RFID WRITEBLOCKABS

## Sintassi

```
RFID WRITEBLOCKABS block "dataHex32" keyType keyHex
```

## Descrizione

Writes 16 bytes (32 HEX) to the absolute block of a MIFARE Classic card.

## Esempi

```
10 RFID INIT 5 27
20 PRINT "Bring card closer for writing"
30 RFID WAITUID U$
40 RFID WRITEBLOCKABS 4 "11223344556677889900AABBCCDDEEFF" A "FFFFFFFFFFFF"
50 PRINT "Scritto"
```

Writing

```
10 DATA$="HELLO WORLD"
20 HEX$=TOHEX(DATA$) 'hypothetical conversion command
30 RFID WRITEBLOCKABS 4 HEX$ A "FFFFFFFFFFFF"
```

Write encoded string

## Note

- Dangerous: can block the card if you write trailers.
- Always check locking and permissions.

# RFIDDB ADD

## Sintassi

```
RFIDDB ADD uid ["label"|label$]
RFIDDB ADD PRESENT ["label"|label$] [timeout_ms] [GOTO line]
RFIDDB ADD SCAN    ["label"|label$] [timeout_ms] [GOTO line]
```

## Descrizione

Adds or updates a tag in the DB in RAM. uid can be HEX string or string variable (U$).

```
With PRESENT/SCAN the system waits for a tag: if read within timeout_ms, it adds it; otherwi
with GOTO line → jump to line;
```

without GOTO → it does not give an error, it simply adds nothing and continues. label is optional and can be string or string variable (N$).

## Esempi

```
10 RFIDDB INIT "tags.json"
20 RFIDDB ADD “04A1B2C3D4” "Mario"
30 RFIDDB SAVE
```

Manual entry (UID and literal labels)

```
10 RFIDDB INIT "tags.json"
20 U$="04A1B2C3D4"
30 N$="Mario"
40 RFIDDB ADD U$ N$
50 RFIDDB SAVE
```

Manual entry with variables

```
10 RFID INIT 5 27
20 RFIDDB INIT "tags.json"
30 PRINT "Approach a new tag (5s)"
40 RFIDDB ADD PRESENT "Alice" 5000
50 RFIDDB SAVE
```

PRESENT with label and timeout (no jump)

```
10 RFID INIT 5 27
20 RFIDDB INIT "tags.json"
30 N$="Ospite"
40 PRINT "Approach a new tag (5s)"
50 RFIDDB ADD PRESENT N$ 5000 GOTO 120
60 PRINT "Tag added!"
70 RFIDDB SAVE
80 RFIDDB LIST
90 GOTO 200
120 PRINT "Timed out: No tags added"
200 END
```

PRESENT with label in variable and jump on timeout

## Note

- Only modify RAM → use RFIDDB SAVE to write to SPIFFS.
- uid and label accept string variables (e.g. U$, N$) or quoted strings.
- PRESENT and SCAN are equivalent in UID acquisition behavior.

# RFIDDB CLEAR

## Sintassi

```
RFIDDB CLEAR
```

## Descrizione

Empties the DB into RAM (does not touch the file on SPIFFS).

## Esempi

```
10 RFIDDB INIT "tags.json"
20 RFIDDB CLEAR
30 PRINT "Totale=";RFIDDB COUNT

Expected output:
Totale=0
```

Clear the database

## Note

- For permanent cleaning: CLEAR + SAVE.

# RFIDDB COUNT

## Sintassi

```
RFIDDB COUNT
```

## Descrizione

Returns the number of tags present in RAM.

## Esempi

```
10 RFIDDB INIT "tags.json"
20 RFIDDB LOAD
30 PRINT "There are";RFIDDB COUNT;" tag"
```

Returns the number of tags present

# RFIDDB DELETEFILE

## Sintassi

```
RFIDDB DELETEFILE
```

## Descrizione

Delete the JSON file from SPIFFS. The DB in RAM remains intact.

## Esempi

```
10 RFIDDB INIT "tags.json"
20 RFIDDB DELETEFILE
30 PRINT "Archive deleted"
```

Delete the JSON file from SPIFFS

# RFIDDB EXISTS

## Sintassi

```
RFIDDB EXISTS uid
```

## Descrizione

Returns 1 if the UID is in RAM, 0 otherwise. Usable in IF, PRINT, LET.

## Esempi

```
10 RFID INIT 5 27
20 RFIDDB INIT "tags.json"
30 RFIDDB LOAD
40 RFID WAITUID U$
50 IF RFIDDB EXISTS U$ THEN PRINT "OK" ELSE PRINT "NO"
```

Returns 1 if the UID is in RAM, 0 otherwise

# RFIDDB INIT

## Sintassi

```
RFIDDB INIT "file.json"
RFIDDB INIT SD "file.json"
```

## Descrizione

Select the storage and load the database into RAM. Without SD → SPIFFS With SD → SD card

## Esempi

```
10 RFIDDB INIT "tags.json"
```

Select the storage and load the database into RAM

```
10 RFIDDB INIT SD "tags.json"
```

Select the storage and load the database into RAM

## Note

- The chosen storage remains valid for all subsequent RFIDDB commands.

# RFIDDB LABEL

## Sintassi

```
RFIDDB LABEL uid var$
```

## Descrizione

Returns the label associated with a UID in var$. If the UID is not present, var$ will be empty.

## Esempi

```
10 RFIDDB INIT "tags.json"
20 RFIDDB LABEL “04A1B2C3D4” NOME$
30 PRINT "Nome=";NOME$

Expected output:
Nome=Mario
```

Returns the label associated with a UID in var$

# RFIDDB LIST

## Sintassi

```
RFIDDB LIST
```

## Descrizione

Prints to serial the list of all the tags present in RAM, with UID and label.

## Esempi

```
10 RFIDDB INIT "tags.json"
20 RFIDDB LIST

Expected output:
UID=04A1B2C3D4 LABEL="Mario"
UID=1122334455 LABEL="Alice"
```

Prints on serial the list of all the tags present in RAM

# RFIDDB LOAD

## Sintassi

```
RFIDDB LOAD
```

## Descrizione

Reload the database from the JSON file in SPIFFS, replacing the contents in RAM.

## Esempi

```
10 RFIDDB INIT "tags.json"
20 RFIDDB LOAD
30 PRINT "DB reloaded"
```

Reload the database from the JSON file in SPIFFS

# RFIDDB REMOVE

## Sintassi

```
RFIDDB REMOVE uid
RFIDDB REMOVE PRESENT [timeout_ms] [GOTO line]
RFIDDB REMOVE SCAN    [timeout_ms] [GOTO line]
```

## Descrizione

Removes a tag from the DB in RAM. With uid: remove directly (accept string or U$ string variable).

```
With PRESENT/SCAN: wait for a tag; if read within timeout_ms, removes it; if it expires:
with GOTO line → jump;
```

without GOTO → it does not give an error, it simply does not remove and continues.

## Esempi

```
10 RFIDDB INIT "tags.json"
20 RFIDDB REMOVE “04A1B2C3D4”
30 RFIDDB SAVE
```

Manual removal (literal UID)

```
10 RFIDDB INIT "tags.json"
20 U$="04A1B2C3D4"
30 RFIDDB REMOVE U$
40 RFIDDB SAVE
```

Manual removal with variable

```
10 RFID INIT 5 27
20 RFIDDB INIT "tags.json"
30 PRINT "Bring tag to remove (5s)"
40 RFIDDB REMOVE PRESENT 5000
50 RFIDDB SAVE
```

Removal with PRESENT and timeout (without jumping)

```
10 RFID INIT 5 27
20 RFIDDB INIT "tags.json"
30 PRINT "Bring tag to remove (5s)"
40 RFIDDB REMOVE PRESENT 5000 GOTO 120
50 PRINT "Removed!"
60 RFIDDB SAVE
70 GOTO 200
120 PRINT "No tags: nothing removed"
200 END
```

Removal with PRESENT and GOTO on timeout

## Note

- Only modify RAM → use RFIDDB SAVE to make permanent.
- uid accepts string or variable (U$).
- PRESENT/SCAN are equivalent for reading the UID.
- GOTO jump occurs only in case of timeout.

# RFIDDB SAVE

## Sintassi

```
RFIDDB SAVE
```

## Descrizione

Save the database in RAM to the JSON file in SPIFFS.

## Esempi

```
10 RFIDDB INIT "tags.json"
20 RFIDDB ADD “04A1B2C3D4” "Mario"
30 RFIDDB SAVE
```

Save database in RAM to JSON file in SPIFFS

# RIGHT$(A$, N)

## Sintassi

```
RIGHT$(stringa$, N)
```

## Descrizione

The RIGHT$ function returns a substring containing the last N characters of the string A$. If N is greater than the length of the string, returns the entire string. It is useful for: Extracting file extensions Checking suffixes Managing codes, final numbers, structured strings

## Esempi

```
10 A$ = "BASIC32"
20 PRINT RIGHT$(A$, 3)

Expected output:
C32
```

Last 3 characters

```
10 FILE$ = "config.bas"
20 EST$ = RIGHT$(FILE$, 4)
30 PRINT "EXTENSION:"; EST$

Expected output:
EXTENSION: .bas
```

Extension of a file name

```
10 S$ = "OK"
20 PRINT RIGHT$(S$, 10)

Expected output:
OK
```

N greater than length

```
10 T$ = "TEST"
20 PRINT RIGHT$(T$, 0)

Expected output:
(empty string)
```

Empty substring

```
10 F$ = "AUTOEXEC.BAS"
20 IF RIGHT$(F$, 4) = ".BAS" THEN PRINT "BASIC FILES"

Expected output:
BASIC FILES
```

Check if a string ends in ".BAS"

## Note

- N must be ≥ 0
- Only works with string variables ($)
- Can be combined with LEFT$, MID$, LEN, CHR$, ASC

# RND(x) / RND(a, b)

## Sintassi

```
RND(x)          ' Numero casuale da 0 a x - 1
RND(a, b)       ' Numero casuale da a a b - 1
```

## Descrizione

The RND function generates random integers in two ways: RND(x) – Basic mode Returns an integer between 0 and x - 1

```
If x = -1 → always generate the same random number (fixed seed)
If x = 0 → returns the last generated random number
```

RND(a, b) – Extended mode Returns an integer between a and b - 1 Also valid with a > b (automatically reverses) Useful for arbitrary intervals (e.g. simulating a dice, choosing from a range)

## Esempi

```
10 PRINT RND(10)

Expected output:
Number from 0 to 9 (e.g. 7)
```

RND(10)

```
10 PRINT RND(5, 15)

Expected output:
Number from 5 to 14 (e.g. 12)
```

RND(5, 15)

```
10 PRINT RND(-1)

Expected output:
(Constant number, e.g. 4)
```

RND(-1)

```
10 A = RND(10)
20 PRINT "Last:", RND(0)

Expected output:
Last: (same number as line 10)
```

RND(0)

```
10 DADO = RND(1, 7)
20 PRINT "LAUNCH:"; DADO

Expected output:
THROW: 5
```

Simulate die roll (1–6)

```
10 PRINT RND(10, 5)

Expected output:
Number between 5 and 9
```

RND with inverted parameters (valid)

### Note

- Always returns an integer
- No need to initialize the random seed
- Great for menus, games, general randomization

# RREAD

## Sintassi

```
RREAD <pin> <VCC> <R_known>
```

## Descrizione

Calculate an unknown resistance R_x by measuring the voltage at the divider node in the diagram: VCC ──[ R_known ]──●──[ R_x ]── GND │ └──→ PIN (ADC) Formula:

```
R_x = (Vnode * R_known) / (VCC - Vnode)
```

where Vnode is the voltage read at the pin (with ADC CAL applied).

## Esempi

```
10 RX = RREAD 34 3.3 10000
20 PRINT "Rx=", RX, " ohm"
```

R_known = 10k, VCC = 3.3V, pin 34

```
10 PIN=34: VCC=3.3: RK=10000
20 RX = RREAD PIN VCC RK
30 PRINT "Rx=", RX
```

With variables

```
10 VP = VREAD 34
20 PRINT "Node (V)=", VP
30 RX = RREAD 34 3.3 10000
40 PRINT "Rx=", RX

Expected output:
Rx= 9873.42 ohms
```

Quick node debugging

```
In caso di nodo a 0 V (corto a GND o partitore errato):
?RREAD ERROR
vout=0 (check wiring)
In caso di R_known non valido:
?RREAD ERROR
r_known must be > 0
```

## Note

- Mandatory pattern: R_known up to VCC, R_x to GND, ADC pin to center node.
- VCC must be the real power supply of the divider (e.g. 3.3).
- RREAD uses ADC CAL calibration.
- If you are using the reverse scheme (R_x on top, R_known on bottom) the formula changes; we can add a separate command if you need.

# RUN

## Sintassi

```
RUN
RUN FAST
```

## Descrizione

```
RUN starts execution of the BASIC program in memory.
RUN FAST runs the program at the maximum possible speed, temporarily disabling:
```

keyboard polling (PS/2 and CardKB) key capture ESC console echo interactive input controls Ideal for: loop-heavy gaming benchmarks high-speed graphics intensive loops During RUN FAST, keyboard input does not work unless reactivated by the program via KEYPOLL ON. RUN Behavior: Stops any running program. It can be called after STOP, after an error or from another program. If RUN FAST is executed, keyboard polling starts disabled.

## Esempi

```
10 PRINT "HELLO WORLD"
20 END

Outputs:
HELLO WORLD
```

Run the program in memory

```
10 INPUT X
20 IF X = 0 THEN STOP
30 PRINT "VALUE:"; X

Execution:
? 0
(STOP)
Then:
RUN
The program starts again from scratch.
```

Restart after STOP

```
RUN FAST

Esegue il programma in modalità turbo (nessun polling tastiera).
```

RUN FAST

## Note

- If memory is empty and you run RUN, the system does nothing.
- RUN FAST does not disable the BREAKPIN.
- During RUN FAST you can re-enable the code keyboard with KEYPOLL ON.

# SAVE

## Sintassi

```
SAVE "nomefile"
SAVE variabile$
SAVE SD "nomefile"
SAVE SD variabile$
SAVE SPIFFS "nomefile"
SAVE SPIFFS variabile$
```

## Descrizione

The SAVE command saves the BASIC program currently in memory to a file. Without specifying the memory, saving takes place in the SPIFFS internal memory. By specifying SD, the file is saved to the SD card. You can also indicate SPIFFS for clarity, but it is not mandatory. The file name can be indicated directly in quotes or via a string variable (e.g. F$). The file is saved in text format, with the lines numbered as in the BASIC listing. If a file with the same name already exists, it is overwritten. The recommended extension is .bas.

## Esempi

```
SAVE "config.bas"

Output atteso:
Il file config.bas viene salvato nella memoria interna, anche se è presente una scheda SD.
```

Save to SPIFFS internal memory

```
SAVE SD "basic.program"

Output atteso:
Il file programma.bas viene salvato sulla scheda SD.
```

Save to SD card

```
10 LET F$ = "menu.bas"
20 SAVE F$

Expected output:
The menu.bas file is saved in the SPIFFS internal memory.
```

Use a string variable for the file name

```
10 LET F$ = "demo.bas"
20 SAVE SD F$

Expected output:
The demo.bas file is saved to the SD card.
```

Save to SD using a variable

## Note

- If the SD card is not connected or not initialized, SAVE SD ... will return an error.
- If the file name does not end in .bas, the system will automatically suggest adding it.
- The SAVE command overwrites any files already present with the same name.
- You can use string variables to dynamically specify the file name.

# SAVEVAR

## Sintassi

```
SAVEVAR "file" "chiave" valore
SAVEVAR SPIFFS "file" "chiave" valore
SAVEVAR SD "file" "chiave" valore
```

## Descrizione

SAVEVAR saves (or updates) a variable inside a JSON file. If the file does not exist, it is created. If the key already exists, it is overwritten. The value can be: number (e.g. 150, 22.5, or a numeric variable such as A) string (e.g. "Anna", or a string variable such as U$) The file is saved in the selected filesystem: default if you do not specify anything SPIFFS if you write SPIFFS SD if you write SD

## Esempi

```
10 SAVEVAR SPIFFS "prefs.json" "SPEED" 150
20 SAVEVAR SPIFFS "prefs.json" "USER" "Anna"

Result (typical file content):
{
"SPEED": 150,
"USER": "Anna"
}
```

Save configuration to SPIFFS

```
10 SAVEVAR SD "data.json" "TEMP" 22.5
20 SAVEVAR SD "data.json" "STATO" "OK"
```

Save data to SD

```
10 A = 42
20 N$ = "Mario"
30 SAVEVAR "test.json" "NUM" A
40 SAVEVAR "test.json" "NOME" N$
```

Save using BASIC variables

## Note

- Strings must be in quotes: "text".
- If you save to SD, the SD must be initialized/mounted (otherwise runtime error).
- SAVEVAR keeps the other keys already present in the file: it updates only the one indicated.

# SCANI2C

## Sintassi

```
SCANI2C
```

## Descrizione

The SCANI2C command scans the I2C bus and prints all the addresses of the detected devices in the serial monitor. Before using SCANI2C it is mandatory to call the WIRE command to set the SDA and SCL pins of the I2C bus. If WIRE was not executed, an error is generated at runtime.

## Esempi

```
10 WIRE 21 22
20 SCANI2C

Typical Output:
Scanning I2C bus...
Found device at 0x3C (60)
Total I2C devices: 1
```

Simple bus scan

## Note

- Useful for checking if I2C sensors are correctly connected.
- Shows addresses in hexadecimal (e.g. 0x3C) and decimal (e.g. 60).
- The result appears in the serial monitor.
- Make sure you have set the pins correctly with WIRE sda scl before using it.

# SDFREE

## Sintassi

```
PRINT SDFREE
```

## Descrizione

The SDFREE command returns the free space available on the SD card. It is used to know the remaining memory before saving or reading from files.

## Esempi

```
10 PRINT SDFREE

Expected output:
available space
```

Show available space on SD

## Note

- The value is expressed in bytes
- This only works if the SD is mounted correctly

# SERVOATTACH

## Sintassi

```
SERVOATTACH <id> <pin> [minUS] [maxUS]
```

## Descrizione

Connect a servo to the ID indicated on the chosen pin.

- Frequency: 50 Hz.
- Pulse range: if specified, it is clamped to 300..3000 µs.
- Automatic fallbacks if the attach fails: first 500..2400, then 1000..2000.
- If the ID was on another pin, it detaches and re-attaches on the new.
- Reject pins 34..39 (input-only).

## Esempi

```
10 SERVOMAX 1
20 SERVOATTACH 0 25
30 SERVOSTATUS

Attach ID 0 to pin 25 with default range (500..2400).
```

Basic attachment

```
10 SERVOATTACH 0 25 600 2400
20 SERVOWRITEMICROS 0 1500

Use a wider low-side range; commands in µs will be clamped to 600..2400.
```

Custom range

## Note

- Recommended pins: 25, 26, 27, 32, 33.
- The actual range used (after fallbacks) is stored and used by commands in µs.

# SERVODETACH

## Sintassi

```
SERVODETACH <id>
```

## Descrizione

Disconnect the servo from the indicated ID and free the associated resources.

## Esempi

```
10 SERVOATTACH 0 25
20 SERVOWRITE 0 90
30 WAIT 1000
40 SERVODETACH 0
```

Detach when finished

```
10 SERVOMAX 3
20 FOR I=0 TO 2
30  SERVODETACH I
40 NEXT I

Expected output:
No printing. In error:
VALUE Invalid id or >= SERVOMAX
```

Multiple detachments

## Note

- After the detach, the pin remains in the set state; if necessary, reset it yourself (e.g. PINMODE / DIGITALWRITE in your BASIC shell if available).

# SERVOMAX

## Sintassi

```
SERVOMAX <n>
```

## Descrizione

Set how many servos can be managed at runtime.

- Allowed values: 1 … MAX_BASIC_SERVOS_CAP (cap = 16).
- If you reduce n, IDs ≥ n are automatically detached.

## Esempi

```
10 SERVOMAX 4
20 SERVOSTATUS

Set runtime limit to 4; “SERVOSTATUS” reflects the new value.
```

Limit to 4 servos

```
10 SERVOMAX 4
20 SERVOATTACH 3 27
30 SERVOATTACH 4 32
40 SERVOMAX 4
50 SERVOSTATUS

ID 4 is detached because it is over the new limit.
```

Reduction with automatic detach

## Note

- Typical default: BASIC_SERVO_MAX = 8.
- Reduce SERVOMAX if you have few servos to save resources.

# SERVOREAD

## Sintassi

```
SERVOREAD <id>
```

## Descrizione

Returns the current logical angle of the servo (the one set via SERVOWRITE), or -1 if the ID is invalid or the servo is not attached.

- Usable within expressions/assignments.

## Esempi

```
10 SERVOATTACH 0 25
20 SERVOWRITE 0 90
30 A = SERVOREAD 0
40 PRINT "ANGLE=", A
```

Simple reading

```
10 SERVOATTACH 0 25
20 SERVOWRITE 0 45
30 IF SERVOREAD 0 < 90 THEN PRINT "OK"

Expected output:
ANGLE= 90 (in example 1)
```

Conditional control

## Note

- It is an internal state value, not a physical measurement of the servo shaft.

# SERVOSTATUS

## Sintassi

```
SERVOSTATUS
```

## Descrizione

Prints the status of the LEDC/servo subsystem:

- Allocated LEDC timers (YES/NO).
- BASIC_SERVO_MAX and MAX_BASIC_SERVOS_CAP.
- For each ID: status (ATTACHED/detached), PIN and RANGE[us]=min..max.
- Count of attached servos and recommended notes/pins.

## Esempi

```
10 SERVOSTATUS

Show current configuration, useful for debugging before using servos.
```

Initial state

```
10 SERVOMAX 2
20 SERVOATTACH 0 25 500 2400
30 SERVOATTACH 1 26
40 SERVOSTATUS

Verify that IDs 0 and 1 are “ATTACHED” with PIN and range.
Expected output (indicative example):
=== SERVANT STATUS ===
Timers allocated: YES
BASIC_SERVO_MAX: 2 (CAP=16)
ID 0: ATTACHED PIN=25 RANGE[us]=500..2400
ID 1: ATTACHED PIN=26 RANGE[us]=500..2400
ID 2: detached PIN=-1 RANGE[us]=0..0
...
Attached count: 2
```

After attach

## Note

- Useful for identifying LEDC conflicts or bad pins.
- Recommended pins are: 25, 26, 27, 32, 33. Avoid 34..39 (input-only) and SPI pins (5, 13, 14, 15, 18, 19, 23)

# SERVOWRITE

## Sintassi

```
SERVOWRITE <id> <angle>
```

## Descrizione

Set the angle of a servo in degrees.

- The angle is saturated at 0..180.
- Requires servo to be attached.

## Esempi

```
10 SERVOATTACH 0 25
20 SERVOWRITE 0 0
30 WAIT 1000
40 SERVOWRITE 0 180
```

Go to 0°, then 180°

```
10 SERVOATTACH 0 25
20 SERVOWRITE 0 0
30 WAIT 500
40 SERVOWRITE 0 90
50 WAIT 500
60 SERVOWRITE 0 180
```

Three positions

## Note

- Use WAIT (ms) between movements to give the servo time to reach the position.

# SERVOWRITEMICROS

## Sintassi

```
SERVOWRITEMICROS <id> <micros>
```

## Descrizione

Directly sets the pulse width in microseconds.

- The value is clamped to the actual range recorded in SERVOATTACH (e.g. 500..2400).
- Requires the servo to be attached.

## Esempi

```
10 SERVOATTACH 0 25
20 SERVOWRITEMICROS 0 1500
```

Typical neutral

```
10 SERVOATTACH 0 25 600 2400
20 SERVOWRITEMICROS 0 500
30 SERVOWRITEMICROS 0 2500
```

With automatic clamp

## Note

- Useful for fine calibrations or for continuously rotating servos (where 1500 µs ≈ stop).

# SETDATE

## Sintassi

```
SETDATE anno mese giorno
```

## Descrizione

The SETDATE command manually sets the date of the system's internal clock. The string must be in the format year, month, day.

## Esempi

```
SETDATE 2025 6 15
PRINT DATED, DATEM, DATEY

Output atteso:
15 6 2025
```

Set the date to June 15, 2025

## Note

- The format must be exactly "year, month, day"
- Does not synchronize the clock: it is a manual change
- Useful in the absence of a network or to configure the RTC module if present
- Can be used together with SETTIME for complete settings

# SETTIME

## Sintassi

```
SETTIME ore minuti secondi
```

## Descrizione

The SETTIME command manually sets the time of the system's internal clock. The string must be in the format hours, minutes, seconds (24 hours).

## Esempi

```
10 SETTIME 14 38 00
20 PRINT TIMEH, TIMEM, TIMES

Expected output:
14 38 00
```

Set the time to 2.38pm

## Note

- The format must be exactly "hours, minutes, seconds"
- Does not synchronize the clock: it is a manual change
- Useful in the absence of a network or to configure the RTC module if present
- Recommended to use in combination with SETDATE for complete settings

# SFXDEF

## Sintassi

```
SFXDEF name$ v wave$ pw atk dec sus rel
```

## Descrizione

Defines a named sound effect (SFX) and sets its timbre and ADSR envelope. The effect is stored internally and can be played with SFXPLAY. name$ effect name (string in quotes) v voice used (4 recommended, dedicated SFX voice) wave$ "TRI", "SAW", "PULSE", "NOISE" pw pulse width (only for "PULSE") atk dec sus rel ADSR parameters (0..15)

## Esempi

```
10 SIDINIT 16000
20 SIDVOL 5
30 SFXDEF "SHOT" 4 "PULSE" 1200 0 3 0 2
40 GOTO 40
```

Definition of a shot

## Note

- Redefine the same name$ overwrites the effect.
- Produces no sound until called with SFXPLAY.

# SFXPLAY

## Sintassi

```
SFXPLAY name$
```

## Descrizione

Plays a sound effect defined with SFXDEF. Does not interrupt the music on voices 1..3.

## Esempi

```
10 SIDINIT 16000
20 SIDVOL 5
30 SFXDEF "COIN" 4 "TRI" 0 0 2 12 2
40 SFXSWEEP "COIN" "C6" "E6" 120
50 SFXPLAY "COIN"
60 GOTO 60
```

SFX playback

## Note

- If another SFX is already active on the same voice, it is stopped.
- Ideal for shooting, jumping, coins, drums.

# SFXSWEEP

## Sintassi

```
SFXSWEEP name$ start end dur
```

## Descrizione

Set a frequency sweep (pitch going up or down) for an effect. start, end note ("C7") or frequencies during sweep duration in ms

## Esempi

```
10 SIDINIT 16000
20 SIDVOL 5
30 SFXDEF "SHOT" 4 "PULSE" 1200 0 3 0 2
40 SFXSWEEP "SHOT" "C7" "C4" 120
50 SFXPLAY "SHOT"
60 GOTO 60
```

Descending shot

## Note

- Should be called after SFXDEF.
- If not set, the SFX uses fixed frequency.

# SIDINIT

## Sintassi

```
SIDINIT rate
```

## Descrizione

ONLY WORKS ON ESP32 DEV!!! Initialize the SID-like audio engine in the background and set the sample rate. For operation, a PAM8403 amplifier and a speaker must be connected to the gpio25 of the esp32.

## Esempi

```
10 SIDINIT 16000
20 SIDVOL 5
30 SIDVOICE 2 "TRI" 0 2 6 10 4
40 SIDMCLR
50 SIDMADD 2 "C4" 300
60 SIDMADD 2 "E4" 300
70 SIDMADD 2 "G4" 600
80 SIDMPLAY 0
90 GOTO 90
```

Audio startup

## Note

- 16000 = warmer C64 sound
- 22050 = brighter

# SIDMADD

## Sintassi

```
SIDMADD v noteOrHz dur
```

## Descrizione

Adds a musical event to the sequence. v 1..3 (music voices only) noteOrHz note ("A4"), frequency or "R" (rest) dur duration in ms

## Esempi

```
10 SIDINIT 16000
20 SIDVOL 5
30 SIDVOICE 1 "TRI" 0 2 6 12 4
40 SIDVOICE 2 "PULSE" 900 3 6 10 5
50 SIDVOICE 3 "TRI" 0 3 6 8 5
60 SIDMCLR
70 SIDMADD 1 "C3" 200
80 SIDMADD 2 "E4" 200
90 SIDMADD 3 "G4" 200
100 SIDMPLAY 1
110 GOTO 110
```

Arpeggio platformer

## Note

- SIDMADD 4 ... generates error (item reserved for SFX).

# SIDMCLR

## Sintassi

```
SIDMCLR
```

## Descrizione

Delete the music sequence loaded with SIDMADD.

## Esempi

```
10 SIDINIT 16000
20 SIDVOL 5
30 SIDVOICE 1 "TRI" 0 2 6 12 4
40 SIDMCLR
50 SIDMADD 1 "C3" 300
60 SIDMADD 1 "G2" 300
70 SIDMPLAY 1
80 WAIT 800
90 SIDMCLR
100 SIDMADD 1 "F2" 300
110 SIDMADD 1 "C3" 600
120 SIDMPLAY 1
130 GOTO 130
```

Pattern change

## Note

- Use it before uploading new music.

# SIDMPLAY

## Sintassi

```
SIDMPLAY loop
```

## Descrizione

Starts music in the background.

```
loop = 1 repeats
loop = 0 play once
```

## Esempi

```
10 SIDINIT 16000
20 SIDVOL 5
30 SIDVOICE 2 "PULSE" 900 2 5 9 4
40 SIDMCLR
50 SIDMADD 2 "C5" 150
60 SIDMADD 2 "E5" 150
70 SIDMADD 2 "G5" 300
80 SIDMPLAY 0
90 GOTO 90
```

Jingles

## Note

- Does not block the BASIC program.

# SIDMSTOP

## Sintassi

```
SIDMSTOP
```

## Descrizione

Stops the music and closes the active notes.

## Esempi

```
10 SIDINIT 16000
20 SIDVOL 5
30 SIDVOICE 2 "TRI" 0 2 6 10 4
40 SIDMCLR
50 SIDMADD 2 "C4" 300
60 SIDMADD 2 "E4" 300
70 SIDMPLAY 1
80 WAIT 1000
90 SIDMSTOP
100 GOTO 100
```

Stop music

## Note

- SFX can continue even after SIDMSTOP.

# SIDVOICE

## Sintassi

```
SIDVOICE v wave$ pw atk dec sus rel
```

## Descrizione

Configure timbre and ADSR of a voice.

```
v = 1..4
```

wave$ waveform pw pulse width atk dec sus rel envelope

## Esempi

```
10 SIDINIT 16000
20 SIDVOL 5
30 SIDVOICE 2 "TRI" 0 2 6 10 4
40 SIDMCLR
50 SIDMADD 2 "A4" 400
60 SIDMPLAY 0
70 WAIT 500
80 SIDVOICE 2 "SAW" 0 1 4 12 2
90 SIDMPLAY 0
100 GOTO 100
```

Tone comparison

## Note

- "TRI" = softer (C64)
- "SAW" / "PULSE" = more aggressive

# SIDVOL

## Sintassi

```
SIDVOL vol
```

## Descrizione

Set the global volume of the audio engine.

## Esempi

```
10 SIDINIT 16000
20 SIDVOICE 2 "TRI" 0 2 6 10 4
30 SIDMCLR
40 SIDMADD 2 "A4" 500
50 SIDVOL 3
60 SIDMPLAY 0
70 WAIT 600
80 SIDVOL 7
90 SIDMPLAY 0
100 GOTO 100
```

Volume comparison

## Note

- Recommended values with PAM8403: 3..6
- High volume without attenuation can distort.

# SIN(x)

## Sintassi

```
SIN(x)
```

## Descrizione

The SIN(x) function returns the sine of the angle x, where x is expressed in degrees (not radians). The returned value is a number between -1 and +1, as expected by the sine function. It is used in mathematical calculations, graphics, animations, simulations and logical conditions. Using with Radians If you have an angle in radians, you need to convert it to degrees before calling SIN: SIN(radians * 180 / PI)

## Esempi

```
10 PRINT "SIN(0) ="; SIN(0)

Expected output:
SIN(0) = 0
```

Sine of 0 degrees

```
10 PRINT "SIN(90) ="; SIN(90)

Expected output:
SIN(90) = 1
```

Sinus of 90 degrees

```
10 FOR A = 0 TO 90 STEP 15
20 PRINT "SIN("; A; ") = "; SIN(A)
30 NEXT A

Expected output (approximately):
SIN(0) = 0
SIN(15) = 0.2588
SIN(30) = 0.5
SIN(45) = 0.7071
SIN(60) = 0.8660
SIN(75) = 0.9659
SIN(90) = 1
```

Sinus table (0°–90°)

## Note

- The argument x is expressed in degrees
- The result is always between -1 and +1
- For cosine and tangent use COS(x) and TAN(x)

# SPC(n)

## Sintassi

```
SPC(n)
```

## Descrizione

The SPC(n) function generates a string of n spaces, useful for formatting output, aligning text, or creating visual margins in the terminal. It can be used: Inside PRINT to create spacing To construct rows with well-separated columns In combination with other strings

## Esempi

```
10 PRINT "CIAO" + SPC(5) + "MONDO"

Expected output:
HELLO WORLD
```

Space between two words

```
10 PRINT "NOME" + SPC(10) + "ETA'"
20 PRINT "LUCA" + SPC(11) + "12"
30 PRINT "MARCO" + SPC(9) + "15"

Expected output:
NAME AGE
LUKE 12
MARK 15
```

Multi-line alignment

```
10 FOR I = 1 TO 5
20 PRINT SPC(I) + "TEST"
30 NEXT I

Expected output:
TEST
  TEST
   TEST
    TEST
     TEST
```

Dynamic use

```
10 INDENT = 8
20 PRINT SPC(INDENT) + "FORMAT LINE"

Expected output:
FORMAT LINE
```

Fixed return

## Note

- If n = 0, no space is generated
- If n is greater than the terminal width, the text may wrap
- Can be combined with TAB(n) for more precise positioning

# SPIFREE

## Sintassi

```
PRINT SPIFREE
```

## Descrizione

The SPIFREE command returns the free space available in the SPIFFS file system (internal memory). Useful for checking the remaining space before saving files or logs.

## Esempi

```
10 PRINT SPIFREE

Expected output:
available space
```

View free space on SPIFFS

## Note

- The value is expressed in bytes
- Does not require parameters
- This only works if SPIFFS is initialized correctly

# SQR(x)

## Sintassi

```
SQR(x)
```

## Descrizione

The SQR(x) function returns the square root of a number. Accepts a numeric value x and returns √x. It is useful for mathematical calculations, geometry, trigonometry, conversions and in all situations where you need to extract a square root. Important condition: x must be greater than or equal to 0. If x < 0, a mathematical error is generated.

## Esempi

```
10 PRINT SQR(9)

Expected output:
3
```

Simple square root

```
10 A = 49
20 PRINT SQR(A)

Expected output:
7
```

Use with variable

```
10 X = 16
20 Y = SQR(X) + 2
30 PRINT Y

Expected output:
6
```

With combined function

```
10 PRINT SQR(-4)

Expected output:
RUNTIME ERROR: SQR domain error (x < 0)
```

Error on negative values

## Note

- x must be ≥ 0
- SQR(x) is calculated internally as sqrt(x)
- Ideal with TAN, SIN, COS, and for Euclidean distance
- You can also use it in complex expressions

# SR595 CLEAR

## Sintassi

```
SR595 CLEAR
```

## Descrizione

Sets all outputs of the 74HC595 chain to LOW (0). Equivalent to SR595 FILL 0.

## Esempi

```
10 SR595 INIT 23 18 5
20 SR595 CLEAR
```

Set all outputs in the chain to LOW

## Note

- Useful for quickly resetting all outputs.
- If you have more chips, all 8 × chips released will be set to 0.

# SR595 FILL

## Sintassi

```
SR595 FILL 0|1
```

## Descrizione

Set all outputs to 0 (LOW) or 1 (HIGH).

## Esempi

```
10 SR595 INIT 23 18 5
20 SR595 FILL 1   'All ON
30 WAIT 1000
40 SR595 FILL 0   'All OFF
```

Set all outputs to 0 (LOW) or 1 (HIGH)

## Note

- Works on one or more chips at the same time.
- To quickly test LEDs/relays on all lines.

# SR595 INIT (SHIFT REGISTER 74HC595)

## Sintassi

```
SR595 INIT data clock latch [oe] [mr] [chips] [MSB|LSB]
```

## Descrizione

Initialize one or more 74HC595 registers and set the control pins.

```
data = DS (Data In) pin.
clock = SHCP pin (Shift Clock).
latch = STCP pin (Latch to update outputs).
oe (opt.) = OE pin (active-LOW; LOW = enabled).
mr (opt.) = MR pin (active-LOW; LOW = reset).
chips (opt.) = number of chips connected in cascade (default 1).
MSB|LSB (opt.) = order in which the bits are sent (default MSB).
```

## Esempi

```
10 SR595 INIT 23 18 5
```

Just one chip

```
10 SR595 INIT 23 18 5 -1 -1 2 LSB

Connection with multiple chips (cascade):
DATA (DS) of the first chip connected to the data pin of the micro.
Q7' of the first chip connected to the DS of the second.
CLOCK (SHCP), LATCH (STCP), OE, MR common to all.
Set chips in INIT to the total number of chips.
Practical advice for more 74HC595
Basic connection:
MCU → DS (DATA) → Q7'→ DS next chip
MCU → SHCP (CLOCK) → all chips
MCU → STCP (LATCH) → all chips
OE (active-LOW) and MR (active-LOW) common (OE=GND, MR=VCC if not used).
```

Two chips in cascade, LSB-first order

```
10 SR595 INIT 23 18 5 -1 -1 3
20 SR595 WRITE &HFFFFFF   'all ON
30 WAIT 1000
40 SR595 CLEAR
50 FOR I=0 TO 23
60   SR595 SET I 1
70   WAIT 100
80 NEXT I
90 GOTO 50
```

chip (24 outputs)

## Note

- After INIT, all outputs are LOW.
- The internal buffer (SR595 WRITE, SR595 SET) handles 8 × chip bits.

# SR595 SET

## Sintassi

```
SR595 SET index 0|1
```

## Descrizione

Set a single output of the chain to 0 or 1. index starts from 0 for Q0 of the first chip (near the micro),

```
7 = Q7 of the first chip,
8 = Q0 of the second chip, and so on.
```

## Esempi

```
10 SR595 INIT 23 18 5 -1 -1 2
20 SR595 SET 0 1    'Q0 of the first chip ON
30 SR595 SET 8 1    'Q0 of the second chip ON
40 WAIT 1000
50 SR595 CLEAR
```

Set a single output of the chain to 0 or 1

## Note

- Update the internal latch immediately.
- Useful for driving single relays, specific LEDs, control lines.

# SR595 SHOW

## Sintassi

```
SR595 SHOW
```

## Descrizione

Re-latch the internal state. Forces the exit of the last loaded state, even without modifications.

## Esempi

```
10 SR595 INIT 23 18 5
20 SR595 FILL 1
30 SR595 SHOW
```

Re-latch the internal state

## Note

- Only needed in particular situations (e.g. after MR hardware reset).
- SET, WRITE, FILL and CLEAR already latch automatically.

# SR595 WRITE

## Sintassi

```
SR595 WRITE value
```

## Descrizione

Writes a numeric value to all outputs (up to 8 × chip bits). The least significant bits (LSB) correspond to Q0..Q7 of the first chip. The next bits to the next chips in cascade.

## Esempi

```
10 SR595 INIT 23 18 5 -1 -1 2
20 SR595 WRITE &H00FF   'Chip1=FF (all ON), Chip2=00 (all OFF)
30 WAIT 1000
40 SR595 WRITE &HFF00   'Chip1=OFF, Chip2=ON
```

## Note

- Override all state: for partial changes use SET.
- For multiple chips, consider binary/hexadecimal representation for convenience.

# STARTFUNC / STOPFUNC

## Sintassi

```
STARTFUNC <nome>
STOPFUNC <nome>
```

## Descrizione

STARTFUNC starts a function defined with FUNC LOOP in non-blocking mode, which will continue to run in the background. STOPFUNC stops the continuous execution of the indicated function.

## Esempi

```
5 PINMODE 2 OUTPUT NOPULL
10 FUNC FLASH
20 DWRITE 2 1
30 DELAY 300
40 DWRITE 2 0
50 DELAY 300
60 ENDFUNC
70 CALLFUNC FLASH

Expected output:
The LED on pin 2 flashes once (on for 300 ms, then off).
```

Function performed only once (CALLFUNC)

```
5 PINMODE 2 OUTPUT NOPULL
10 FUNC BLINK LOOP
20 DWRITE 2 1
30 DELAY 500
40 DWRITE 2 0
50 DELAY 500
60 ENDFUNC
70 STARTFUNC BLINK

Expected output:
The LED flashes continuously every 500 ms, without blocking the rest of the program.
```

Background loop function (FUNC ... LOOP + STARTFUNC)

```
10 STOPFUNC BLINK

Expected output:
The LED stops flashing.
```

Stop a cyclic function

## Note

- The function name must be unique and without spaces
- Normal functions are called with CALLFUNC
- Functions with LOOP are executed in parallel with STARTFUNC and stopped with STOPFUNC
- Each FUNC must always be closed with ENDFUNC
- Standard BASIC commands (PRINT, IF, WAIT, DWRITE, etc.) can be used within the function.
- Can be used for multitasking (e.g. sensors, outputs, periodic checks)

# STOP, CONT, END

## Sintassi

```
STOP
CONT
END
```

## Descrizione

STOP Suspends the execution of the program voluntarily. The user can then use CONT to resume execution from the point where it was stopped. Useful for debugging or pausing execution. CONT Resumes execution from the point where a STOP was executed. It does not work if the program was interrupted with END, RUN, or after an error. If used without a previous STOP, it has no effect. END Completely terminates the current program. After END, CONT cannot be used. The END is optional: if not present, the program automatically ends at the last line.

## Esempi

```
10 INPUT "ENTER A NUMBER:"; N
20 IF N = 0 THEN STOP
30 PRINT "VALID NUMBER:"; N

Expected output:
ENTER A NUMBER: ? 0
(STOP)
Then:
CONT
Outputs:
Resumes execution from the line following the STOP.
```

Using STOP and CONT

```
10 PRINT "INIZIO"
20 END
30 PRINT "THIS IS NEVER DONE"

Expected output:
INIZIO
```

Use of END

```
10 FOR I = 1 TO 5
20 PRINT "PASSO"; I
30 IF I = 3 THEN STOP
40 NEXT I

Output after RUN:
STEP 1
STEP 2
STEP 3
(STOP)
After:
CONT
Final output:
STEP 4
STEP 5
```

Conditioned STOP + manual CONT

### Note

- STOP is useful for debugging or logical pauses
- CONT works only after STOP, not after END or RUN
- END closes the program cleanly

# STR$(x) o STR$(x, n)

## Sintassi

```
STR$(numero)
STR$(numero, decimali da visualizzare)
```

## Descrizione

The STR$ function converts a number (integer or decimal) to a text string. It is useful when you want to concatenate numbers to strings, or save/print values in text format. The converted number retains its sign and is transformed into a string compatible with other string functions (e.g. LEFT$, RIGHT$, LEN).

## Esempi

```
10 X = 123
20 PRINT STR$(X)

Expected output:
123
```

Basic conversion

```
10 V = 42
20 PRINT "THE VALUE IS" + STR$(V)

Expected output:
THE VALUE IS 42
```

Concatenation with text

```
10 A = -17
20 PRINT STR$(A)

Expected output:
-17
```

Negative numbers

```
10 T = 88
20 RIGA$ = "TOTALE=" + STR$(T)
30 SAVEINT "total.txt"

(Assuming the contents of the listing are saved with the value converted to text)
```

Use in data storage

```
10 N = 12345
20 S$ = STR$(N)
30 PRINT "FIGURES:"; LEN(S$)

Expected output:
FIGURES: 5
```

Combined use with LEN

## Note

- The result of STR$ is a numeric string, which can be converted back to a number with VAL(A$)
- Compatible with all string operators and with INPUT, PRINT, SAVE, etc.

# SYNCNTP

## Sintassi

```
SYNCNTP
```

## Descrizione

The SYNCNTP command synchronizes the system's internal clock with an NTP (Network Time Protocol) server via a Wi-Fi connection. Once executed, the system time is automatically updated with the current date and time obtained via the internet.

## Esempi

```
10 SYNCNTP
```

Clock synchronization

## Note

- An active Wi-Fi connection is required at time of execution
- The default time zone is UTC, but can be changed with the TIMEZONE command
- Does not require parameters
- Useful for applications that require exact time (e.g. data logging, scheduling)

# TAN(x)

## Sintassi

```
TAN(x)
```

## Descrizione

The TAN(x) function returns the tangent of the angle x, where x is in degrees (not radians). The tangent is defined as:

```
TAN(x) = SIN(x) / COS(x)
The returned value can range between -∞ and +∞ and is undefined for angles where COS(x) = 0
```

Use with radians If the angle is expressed in radians, convert it to degrees: TAN(radians * 180 / PI)

## Esempi

```
10 PRINT "TAN(0) ="; TAN(0)

Expected output:
TAN(0) = 0
```

Tangent of 0 degrees

```
10 PRINT "TAN(45) ="; TAN(45)

Expected output:
TAN(45) = 1
```

45 degree tangent

```
10 PRINT "TAN(60) ="; TAN(60)

Expected output (approximately):
TAN(60) = 1.732
```

60 degree tangent

```
10 PRINT TAN(89)
20 PRINT TAN(90)

Typical Output:
TAN(89) → very large value
TAN(90) → RUNTIME ERROR (domain error: cos ~ 0)
```

Critical values

## Note

- The argument x is expressed in degrees
- TAN(x) may generate error when COS(x) is close to zero
- For graphics and animations it is preferable to use SIN and COS
- TAN(x) is consistent with SIN(x) / COS(x)

# TCSFILTER

## Sintassi

```
TCSFILTER RED
TCSFILTER GREEN
TCSFILTER BLUE
TCSFILTER CLEAR
```

## Descrizione

The TCSFILTER command manually selects which color filter to activate on the sensor: RED → red filter GREEN → green filter BLUE → blue filter CLEAR → no filter, total light

## Esempi

```
10 TCSINIT 17 16 4 2 15 5
20 TCSFILTER RED
30 PRINT "RED filter active"
```

Set red filter

```
10 TCSINIT 17 16 4 2 15 5
20 TCSFILTER CLEAR
30 PRINT "CLEAR filter active"

Expected output:
CLEAR filter active
```

Set transparent filter

## Note

- Normally you don't need to use it, because the TCSREAD and TCSRGB commands automatically change the filters.
- Can be useful for debugging or measuring only one channel at a time.

# TCSINIT (Sensore colore TCS230/TCS3200)

## Sintassi

```
TCSINIT S0 S1 S2 S3 OUT [LEDPIN]
```

## Descrizione

The TCSINIT command initializes the sensor by assigning the pins used: S0, S1, S2, S3 → digital pins for configuring the sensor OUT → input pin that reads the frequency from the sensor LEDPIN (optional) → pin that controls the module's LED lighting

## Esempi

```
10 TCSINIT 17 16 4 2 15 5
20 PRINT "Sensor initialized with LED"

Expected output:
Sensor initialized with LED
```

Complete initialization

```
10 TCSINIT 17 16 4 2 15
20 PRINT "Sensor initialized without LED"

Expected output:
Sensor initialized without LED
```

Initialization without LED

## Note

- Must be the first sensor command in the program.
- If you do not use the LED, the LEDPIN parameter can be omitted.
- After TCSINIT, the default scale is 20%.

# TCSLED

## Sintassi

```
TCSLED ON
TCSLED OFF
```

## Descrizione

The TCSLED command controls the LED lighting of the module: ON → turns on the integrated white LED OFF → turns off the LED

## Esempi

```
10 TCSINIT 17 16 4 2 15 5
20 TCSLED ON
30 PRINT "LED on"

Expected output:
LED on
```

Turn on the LED

```
10 TCSINIT 17 16 4 2 15 5
20 TCSLED ON
30 DELAY 500
40 TCSLED OFF
50 DELAY 500
60 TCSLED ON

Expected output:
The LED flashes twice.
```

LED blink

## Note

- This only works if you have specified a LEDPIN pin in TCSINIT.
- Useful for having constant light conditions during measurement.

# TCSREAD

## Sintassi

```
TCSREAD varR varG varB varC
```

## Descrizione

The TCSREAD command performs the complete reading of the 4 channels: varR → Red value varG → Green value varB → Blue value varC → Clear value (total light) The returned values are frequencies in Hz.

## Esempi

```
10 TCSINIT 17 16 4 2 15 5
20 TCSLED ON
30 TCSSCALE 20
40 TCSREAD R G B C
50 PRINT "R=";R;" G=";G;" B=";B;" C=";C

Expected output (variable values):
R=1450 G=1780 B=920 C=4200
```

Single reading

```
10 TCSINIT 17 16 4 2 15 5
20 FOR I=1 TO 3
30   TCSREAD ROSSO VERDE BLU CHIARO
40   PRINT I;") ";ROSSO;",";VERDE;",";BLU;",";CHIARO
50   DELAY 500
60 NEXT I

Expected output:
Three rows of different RGB+C values ■■for each iteration.
```

Repeated reading in loop

## Note

- The variables (R, G, B, C, etc.) are chosen by the user.
- If the values are too high or low, use TCSSCALE to adjust the sensitivity.
- It is the most used command for color analysis.

# TCSRGB

## Sintassi

```
TCSRGB varR varG varB
```

## Descrizione

The TCSRGB command reads only the three main channels (Red, Green and Blue), ignoring the Clear channel.

## Esempi

```
10 TCSINIT 17 16 4 2 15 5
20 TCSRGB R G B
30 PRINT "RGB: ";R;",";G;",";B

Expected output (variable values):
RGB: 1500,1800,950
```

RGB reading

```
10 TCSINIT 17 16 4 2 15 5
20 TCSRGB R G B
30 IF R>G AND R>B THEN PRINT "RED COLOR"
40 IF G>R AND G>B THEN PRINT "GREEN COLOR"
50 IF B>R AND B>G THEN PRINT "BLUE COLOR"

Expected output (sample dependent):
RED COLOR
```

Dominant color identification

## Note

- Faster than TCSREAD because it ignores the Clear channel.
- Useful for recognizing the dominant color of an object.

# TCSSCALE

## Sintassi

```
TCSSCALE n
```

## Descrizione

The TCSSCALE command sets the sensor output frequency scale:

```
0 → Power Down (sensor off)
2 → 2% scale (for very bright lights)
20 → Scale 20% (default)
100 → Scale 100% (for weak lights)
```

## Esempi

```
10 TCSINIT 17 16 4 2 15 5
20 TCSSCALE 2
30 PRINT "Scale set to 2%"

Expected output:
Scale set to 2%
```

Scale to 2%

```
10 TCSINIT 17 16 4 2 15 5
20 TCSSCALE 0
30 PRINT "Sensor on standby"

Expected output:
Sensor on standby
```

Turn off the sensor

## Note

- Use low values (2 or 20) in bright environments to avoid saturation.
- Use 100 in dark environments to increase sensitivity.
- If the sensor needs to be temporarily deactivated, set 0.

# TI

## Sintassi

```
TI
```

## Descrizione

```
TI is a system variable that represents the time elapsed since the ESP32 was turned on or re
```

It is read-only and is used to measure time intervals, delays, or timed events. It is particularly useful in combination with: DELAYMILLIS (to check the past time) WHILE/WEND for non-blocking pauses

```
IF for duration-based conditions
```

## Esempi

```
10 PRINT "TI ="; TI

Expected output:
TI = 1275
(Example value: 12.75 seconds from boot)
```

Printing the current timer

```
10 T0 = TI
20 FOR I = 1 TO 1000
30 NEXT I
40 DT = TI - T0
50 PRINT "ELAPSED TIME ="; DT; "(hundredths of a second)"

Expected output:
ELAPSED TIME = 15
```

Measurement of time between two points

```
10 T = TI
20 PRINT "I wait 3 seconds..."
30 WHILE TI < T + 300: WEND
40 PRINT "Done!"

Expected output:
I wait 3 seconds...
(Stopped for 3 seconds)
Done!
```

Non-blocking timer with WHILE

## Note

- TI will reset if the device is restarted
- It is expressed in hundredths of a second: multiply by 10 to get milliseconds, divide by 100 to get seconds
- To get formatted time, use TI$

# TIMEH

## Sintassi

```
TIMEH
```

## Descrizione

The TIMEH command returns the current time (0–23) according to the system's internal clock. It is useful for time-based controls (e.g. time automations).

## Esempi

```
10 PRINT TIMEH

Expected output:
14
```

Print the current time

## Note

- Returns an integer from 0 to 23
- Does not require parameters
- Useful in time precision checks

# TIMEM

## Sintassi

```
TIMEM
```

## Descrizione

The TIMEM command returns the current minutes (0–59) according to the internal clock.

## Esempi

```
10 PRINT TIMEM

Expected output:
38
```

Print the current minutes

## Note

- Returns an integer from 0 to 59
- Does not require parameters
- Useful in time precision checks

# TIMES

## Sintassi

```
TIMES
```

## Descrizione

The TIMES command returns the current seconds (0–59) of the internal clock. Allows per-second controls or short-term timing.

## Esempi

```
10 PRINT TIMES

Expected output:
05
```

Print the current seconds

## Note

- Returns an integer from 0 to 59
- Does not require parameters
- Useful in time precision checks

# TIMEZONE

## Sintassi

```
TIMEZONE codice
TIMEZONE "regola_POSIX"
TIMEZONE +H
TIMEZONE -H
TIMEZONE +H:MM
TIMEZONE -H:MM
TIMEZONE HELP
TIMEZONE ?
```

## Descrizione

TIMEZONE sets the time zone (timezone) used by the system to calculate local time.

TIMEZONE sets the time zone (timezone) used by the system to calculate local time. • code is a short identifier (IT, US, JP, …) that is automatically translated into a time zone string in POSIX format. • "POSIX_rule" allows you to directly set a complete POSIX rule (including daylight saving time if present). • +H, -H, +H:MM, -H:MM allow you to quickly manually set the offset from UTC: or +1 means UTC+1 or -5 means UTC-5 or +5:30 means UTC+5 hours and 30 minutes This is converted internally to the correct POSIX format (where the sign is reversed). The time zone setting is applied immediately and is also reused by the SYNCNTP command, so it remains valid after NTP synchronization. The internal hardware clock is not changed: it only changes how local date and time are interpreted relative to UTC. _____ Special Topics • TIMEZONE HELP • TIMEZONE ? They show the complete list of supported codes and their respective POSIX rules. This function replaces the old LISTTIMEZONES command. _____ Codes available These codes are already mapped internally: Code POSIX Rule Geographical area / Notes IT CET-1CEST,M3.5.0/2,M10.5.0/3 Italy FR CET-1CEST,M3.5.0/2,M10.5.0/3 France DE CET-1CEST,M3.5.0/2,M10.5.0/3 Germany UK GMT0BST,M3.5.0/1,M10.5.0/2 United Kingdom ES CET-1CEST,M3.5.0/2,M10.5.0/3 Spain US EST5EDT,M3.2.0/2,M11.1.0/2 United States (East coast, e.g. New York) CA EST5EDT,M3.2.0/2,M11.1.0/2 Canada (East area) JP JST-9 Japan (no daylight saving time) CN CST-8 China (no daylight saving time) IN IST-5.30am India (split offset, no DST) AU AEST-10AEDT,M10.1.0/2,M4.1.0/3 Australia (Sydney) BR BRT3BRST,M10.3.0/0,M2.3.0/0 Brazil RU MSK-3 Russia (Moscow) MX CST6CDT,M4.1.0,M10.5.0 Mexico AR ART3 Argentina ZA SAST-2 South Africa EG EET-2 Egypt KR KST-9 South Korea (no daylight saving time) SE CET-1CEST,M3.5.0/2,M10.5.0/3 Sweden CH CET-1CEST,M3.5.0/2,M10.5.0/3 Switzerland NL CET-1CEST,M3.5.0/2,M10.5.0/3 Netherlands BE CET-1CEST,M3.5.0/2,M10.5.0/3 Belgium PL CET-1CEST,M3.5.0/2,M10.5.0/3 Poland GR EET-2EEST,M3.5.0/3,M10.5.0/4 Greece FI EET-2EEST,M3.5.0/3,M10.5.0/4 Finland NO CET-1CEST,M3.5.0/2,M10.5.0/3 Norway PT WET0WEST,M3.5.0/1,M10.5.0/2 Portugal DK CET-1CEST,M3.5.0/2,M10.5.0/3 Denmark IE GMT0IST,M3.5.0/1,M10.5.0/2 Ireland UTC UTC0 Coordinated Universal Time (no offset) Note: POSIX rules also include automatic winter time/daylight saving time switching where applicable (e.g. CET-1CEST,...). _____ Use with manual offset You can set a custom time zone without using existing code. Valid examples: TIMEZONE +1 TIMEZONE -5 TIMEZONE +5:30 Meaning: • TIMEZONE +1 sets the local time to UTC+1. • TIMEZONE -5 sets the local time to UTC-5. • TIMEZONE +5:30 sets the local time to UTC+5 hours and 30 minutes. Internally, these values are automatically converted to an equivalent POSIX rule (e.g. UTC-1 to UTC+1), so the system can use them right away. This mode does not define daylight saving time: it is a fixed offset.

## Esempi

```
10 TIMEZONE IT
```

```
20 SYNCNTP

Expected output: current date and time in local Italian format, with automatic management of
```

Set the Italian time zone and synchronize

```
TIMEZONE +1
SYNCNTP
```

Manually set UTC+1 without table

```
TIMEZONE HELP
oppure
TIMEZONE ?

Output atteso: elenco codici (IT, US, JP, …) con la relativa definizione POSIX.
```

View the list of supported time zones

## Note

- TIMEZONE HELP / TIMEZONE ? replaced LISTTIMEZONES.
- Changing the timezone is immediate.
- The internal time (RTC/system counter) remains in UTC; only the conversion to local time changes.
- After choosing the correct time zone, using SYNCNTP updates the network clock and gives you the correct local date/time.

# TO STEP NEXT

## Sintassi

```
FOR variabile = inizio TO fine [STEP incremento]
  ...
NEXT [variabile]
```

## Descrizione

The FOR...NEXT structure is used to create counter loops, in which a numeric variable is automatically incremented or decremented until it reaches a final value. variable: name of the counter (e.g. I) start: initial value end: final value

```
STEP: increment (positive or negative, optional — default = 1)
```

The body of the loop can contain any statement, including IF, PRINT, LET, GOTO, etc.

## Esempi

```
10 FOR N = 0 TO 10 STEP 2
20 PRINT "N ="; N
30 NEXT N

Expected output:
N = 0
N = 2
N = 4
N = 6
N = 8
N = 10
```

Cycle with customized increment (STEP 2)

```
10 FOR X = 10 TO 1 STEP -3
20 PRINT X
30 NEXT X

Expected output:
10
7
4
1
```

Decreasing cycle (negative STEP)

```
10 SUM = 0
20 FOR I = 1 TO 10
30 SUM = SUM + I
40 NEXT I
50 PRINT "SUM ="; SUM

Expected output:
SUM = 55
```

Calculate the sum of the numbers from 1 to 10

# TONE

## Sintassi

```
TONE pin freq durata
```

## Descrizione

Outputs a sound on the specified pin, with a frequency expressed in Hertz (freq) or as a note name in quotation marks (e.g. "A4"), for a specified duration in milliseconds (duration).

Common frequencies Note Frequency (Hz) DO 262 RE 294 MI 330 FA 349 SOL 392 LA 440 YES 494 Map notes available (C1 to B7) Eighth Note (frequency in Hz) C1–B1 C1 33, C#1 35, D1 37, D#1 39, E1 41, F1 44, F#1 46, G1 49, G#1 52, A1 55, A#1 58, B1 62 C2–B2 C2 65, C#2 69, D2 73, D#2 78, E2 82, F2 87, F#2 92, G2 98, G#2 104, A2 110, A#2 117, B2 123 C3–B3 C3 131, C#3 139, D3 147, D#3 156, E3 165, F3 175, F#3 185, G3 196, G#3 208, A3 220, A#3 233, B3 247 C4–B4 C4 262, C#4 277, D4 294, D#4 311, E4 330, F4 349, F#4 370, G4 392, G#4 415, A4 440, A#4 466, B4 494 C5–B5 C5 523, C#5 554, D5 587, D#5 622, E5 659, F5 698, F#5 740, G5 784, G#5 831, A5 880, A#5 932, B5 988 C6–B6 C6 1047, C#6 1109, D6 1175, D#6 1245, E6 1319, F6 1397, F#6 1480, G6 1568, G#6 1661, A6 1760, A#6 1865, B6 1976 C7–B7 C7 2093, C#7 2217, D7 2349, D#7 2489, E7 2637, F7 2794, F#7 2960, G7 3136, G#7 3322, A7 3520, A#7 3729, B7 3951

## Esempi

```
10 TONE 26 440 500
```

Outputs a sound at 440 Hz (note A4) for 500 ms on pin 26.

```
20 TONE 26 "C5" 300
```

Outputs a fifth octave C for 300 ms.

## Note

- The pin must be connected to a piezoelectric buzzer or speaker.
- If you use a note in quotes, the system will automatically convert the note to its frequency.
- Quotation marks must be double (") and mandatory to indicate a note (e.g. "A4"), they must not be used for numerical frequencies.
- Common frequencies
- Map notes available
- (from C1 to B7)

# ULTRA INIT (SENSORE ULTRASUONI HC-SR04)

## Sintassi

```
ULTRA INIT <trig> <echo> [timeout_us]
```

## Descrizione

Configure the TRIG and ECHO pins of the HC-SR04 sensor. The optional parameter sets the read timeout in microseconds.

```
Produces no output; enable subsequent readings with ULTRA READ.
```

## Esempi

```
10 ULTRA INIT 5 18
20 ULTRA READ

Expected output:
a number in cm (e.g. 42.15)
```

Basic initialization

```
10 ULTRA INIT 5 18 40000
20 ULTRA READ

Expected output:
a number in cm (e.g. 57.82)
```

Custom timeout

## Note

- Use ULTRA INIT before ULTRA READ
- Default timeout ≈ 30000 µs (~5 m)
- The ECHO pin of the HC-SR04 is 5 V: use a divider towards the ESP32; TRIG at 3.3V is fine

# ULTRA READ

## Sintassi

```
ULTRA READ [samples] [var]
```

## Descrizione

Measure the distance in centimeters. Without var it directly prints the distance (in cm). With var, assign the value to a variable (numeric or string if it ends with $). The optional samples parameter indicates the number of samples to average for a more stable reading.

## Esempi

```
10 ULTRA INIT 5 18
20 ULTRA READ 3

Expected output:
a number in cm per line (e.g. 41.97)
```

Direct print (average 3)

```
10 ULTRA INIT 5 18
20 ULTRA READ 5 D
30 PRINT D
40 IF D < 20 THEN PRINT "TOO CLOSE"

Expected output:
18.34
TOO CLOSE
```

Assignment to variable and threshold

```
10 ULTRA INIT 5 18
20 ULTRA READ 3 S$
30 PRINT S$

Expected output:
e.g. 33.20
```

Assignment to string

## Note

- Works after ULTRA INIT
- In case of timeout: print “NaN” or assign NaN to the numeric variable
- Recommended samples 3–7 for stability
- Compatible with PRINT, LET, IF (using a variable)

# ULTRA TIMEOUT

## Sintassi

```
ULTRA TIMEOUT <timeout_us>
```

## Descrizione

Set the global timeout (in microseconds) for ultrasonic readings.

```
Does not print anything; the new value applies to subsequent ULTRA READ.
```

readings

## Esempi

```
10 ULTRA INIT 5 18
20 ULTRA TIMEOUT 40000
30 ULTRA READ

Expected output:
a number in cm
```

Increase the flow rate

```
10 ULTRA INIT 5 18
20 ULTRA TIMEOUT 20000
30 ULTRA READ 3 D
40 PRINT D

Expected output:
averaged cm (e.g. 35.44)
```

Reduce latency

## Note

- Required value > 0
- To be used after ULTRA INIT; influences all subsequent readings
- Choose larger timeout for longer distances, smaller for faster response

# VAL(A$)

## Sintassi

```
VAL(stringa$)
```

## Descrizione

The VAL function converts a numeric string to a numeric value (integer or with decimals). It is useful for: Interpreting textual input as numbers Performing calculations on data received as strings Reading numerical values saved in files or from serial input If the string does not begin with a valid number, the result will be 0.

## Esempi

```
10 S$ = "123"
20 N = VAL(S$)
30 PRINT N + 1

Expected output:
124
```

Simple conversion

```
10 INPUT "ENTER A NUMBER AS A STRING:"; T$
20 V = VAL(T$)
30 PRINT "DOUBLE NUMBER:"; V * 2

Example input:
ENTER A NUMBER AS A STRING: ? "50"
Expected output:
DOUBLE NUMBER: 100
```

Direct use with INPUT

```
10 S$ = "3.14"
20 PRINT VAL(S$) * 2

Expected output:
6.28
```

Converting decimals

```
10 A$ = "ABC123"
20 PRINT VAL(A$)

Expected output:
0
```

Initial part non-numeric

```
10 X = 77
20 S$ = STR$(X)
30 Y = VAL(S$)
40 PRINT Y + 1

Expected output:
CopiaModifica
78
```

Comparison with STR$

### Note

- Reads only the starting number in the string
- If the string is empty or non-numeric, returns 0
- Logical inverse of STR$

# VERIFY

## Sintassi

```
VERIFY "nomefile"
VERIFY variabile$
VERIFY SD "nomefile"
VERIFY SD variabile$
VERIFY SPIFFS "nomefile"
VERIFY SPIFFS variabile$
```

## Descrizione

VERIFY compares the program currently in memory with the contents of the indicated file. Without specifying memory, check in SPIFFS (internal memory). With SD, check the file on the SD card. You may explicitly state SPIFFS for clarity. The file name can be passed in quotes or via a string variable (e.g. F$). Possible outcome: File equal to the listing in memory → matches File different from the listing in memory → different File not found → error The operation is for control purposes only: it does not modify anything.

## Esempi

```
VERIFY "main.bas"

Output atteso (se identico):
VERIFY successful (SPIFFS): content matches.
```

Verify a file in SPIFFS (default)

```
VERIFY SD "backup.bas"

Output atteso (se differente):
VERIFY FAILED: line mismatch.
EXPECTED: 10 PRINT "CIAO"
FOUND:    10 PRINT "HELLO"
```

Check a file on SD

```
10 LET F$ = "setup.bas"
20 VERIFY F$
```

Use a string variable

## Note

- If the file contains extra lines compared to the in-memory listing, it is reported as different.
- If the SD card is not present/mounted, VERIFY SD ... will return a read error.
- It is recommended to use the .bas extension for consistency with other commands.

# VREAD(p)

## Sintassi

```
VREAD(p)
VREAD(p) <R1> <R2>
```

## Descrizione

VREAD returns the voltage in Volts: With a single form: it measures the voltage present at the pin (Vpin). With divider: reconstructs the voltage upstream of a resistive divider R1–R2 connected to the pin (central node). Formula:

```
V_in = V_pin * (R1 + R2) / R2
```

## Esempi

```
10 V = VREAD(34)
20 PRINT "Vpin=", V

Shows the voltage measured at the pin node.
```

Direct reading on pin 34

```
10 V = VREAD(34) 100000 100000
20 PRINT "VBAT=", V

Rebuilds battery voltage (≈ 2 × Vpin).
```

Battery reading with divider (100k/100k) on pin 34

```
10 PIN=34: R1=100000: R2=100000
20 VB = VREAD(PIN) R1 R2
30 PRINT "VBAT=", VB

Expected output:
Vpin= 0.012345
VBAT= 3.274910
```

With variables

## Note

- VREAD uses ADC CAL parameters (REF/GAIN/OFFSET).
- With the divider, R2 must be >0 (otherwise error).
- If the pin is floating you may read random values.
- Do not exceed 3.3 V on the ADC pin of the ESP32.

# WAIT n

## Sintassi

```
WAIT n
```

## Descrizione

The WAIT command suspends program execution for n milliseconds. During the delay, the microcontroller does not execute any other code: it is a blocking pause. It is useful for: Waiting between two operations Creating animations or flashes Simulating loading times

## Esempi

```
10 PRINT "CIAO"
20 WAIT 1000
30 PRINT "MONDO"

Expected output (with 1 second pause between the two lines):
CIAO
(monitor pause)
MONDO
```

Pause between two messages

```
10 CLS
20 PRINT "■"
30 WAIT 500
40 CLS
50 WAIT 500
60 GOTO 10

Expected output:
An infinite flash of the "■" symbol every half second.
```

Simulated flashing

```
10 INPUT "ENTER YOUR NAME:"; N$
20 PRINT "ATTENDI..."
30 WAIT 2000
40 PRINT "BENVENUTO "; N$

Expected output:
After input, a 2 second pause before the welcome message.
```

Delay after reading

```
10 FOR I = 5 TO 1 STEP -1
20 PRINT I
30 WAIT 1000
40 NEXT I
50 PRINT "GO!"

Expected output:
A countdown from 5 to 1 with 1 second between each number.
```

Countdown

## Note

- WAIT completely blocks execution (including INPUT, GET, etc.)

- The unit of measurement is the millisecond (1000 = 1 second)

# WIFI

## Sintassi

```
WIFI <ssid_expr> <password_expr>
```

## Descrizione

Connects the ESP32 to a Wi-Fi network in station mode (WIFI_STA). Useful for displaying web pages on existing network infrastructure or synchronizing settings (e.g. date/time) via the network.
Outputs/Effects If successful, the ESP connects to Wi-Fi. You can read the IP with:

```
PRINT IP (station IP)
```

Useful variables exposed: ESPMAC$ → MAC address of the card

## Esempi

```
10 WIFI “MySSID” “MyPassword”

Expected output:
Wi-Fi connection activated.
```

Simple connection

```
10 LET SSID$="My house"
20 LET PASS$="qwerty123"
30 WIFI SSID$ PASS$

Expected output:
Wi-Fi connection activated.
```

Connection with variables

```
10 LET BASE$="Ufficio"
20 WIFI BASE$+"_GUEST" "passguest"
```

Connection with expression

```
10 WIFI “MySSID” “MyPassword”
20 HTML DEFAULT
30 HTMLOBJ "<h2>Example of web page</h2>"
60 HTMLSTART

Expected output:
Wi-Fi connection activated and web page created on the IP address (check with PRINT IP).
```

Web page reachable via IP

```
10 PINMODE 2 OUTPUT NOPULL
20 WIFI “MySSID” “MyPassword”
30 HTML DEFAULT
40 HTMLOBJ "<h2>LED control on board</h2>"
50 HTMLOBJ "<button onclick='fetch(\\"/exec?cmd=DWRITE 2 1\\")'>Turn on LED</button>"
60 HTMLOBJ "<button onclick='fetch(\\"/exec?cmd=DWRITE 2 0\\")'>Turn off LED</button>"
70 HTMLSTART
```

Web page that controls GPIO

## Note

- Connection timeout ~10 s.

• If PRINT IP returns 0.0.0.0, the connection was unsuccessful.

• For SSID/password with spaces, use quotes or variables/expressions (e.g. SSID$="Network with spaces" → WIFI SSID$ PASS$).

# WIFIAP

## Sintassi

```
WIFIAP <ssid_expr> <password_expr>
```

## Descrizione

Start a Wi-Fi Access Point on the ESP32 (WIFI_AP mode). Useful for creating an autonomous local network for configuration pages (e.g. time/date) or for sending BASIC commands (GPIO, etc.). Outputs/Effects If successful, the AP is active. You can read the AP's IP with:

```
PRINT IPAP
```

## Esempi

```
10 WIFIAP MyAP myap_pass

Expected output:
Local Wi-Fi network created.
```

Simple AP

```
10 LET APSSID$="ConfigESP"
20 LET APPASS$="config123"
30 WIFIAP APSSID$ APPASS$
```

AP with variables

```
10 LET BASE$="Demo"
20 WIFIAP BASE$+"_AP" "demopass1"
```

AP with expression

```
10 WIFIAP MyAP myap_pass
20 HTML DEFAULT
30 HTMLOBJ "<h2>Example of web page</h2>"
60 HTMLSTART

Expected output:
AP activated and web page created on the local IP (check with PRINT IPAP).
```

AP web page

```
10 PINMODE 2 OUTPUT NOPULL
20 WIFIAP MyAP myap_pass
30 HTML DEFAULT
40 HTMLOBJ "<h2>LED control on board</h2>"
50 HTMLOBJ "<button onclick='fetch(\\"/exec?cmd=DWRITE 2 1\\")'>Turn on LED</button>"
60 HTMLOBJ "<button onclick='fetch(\\"/exec?cmd=DWRITE 2 0\\")'>Turn off LED</button>"
70 HTMLSTART
```

Web page that controls GPIO on AP

## Note

- Password < 8 characters → error.
- If you need to allow open AP (without password), a variant can be provided (WIFIAP SSID "") – tell me if you want to enable it.
- To know the IP of the AP: PRINT IPAP.

# WIFIAPCONNECTED

## Sintassi

```
WIFIAPCONNECTED
```

## Descrizione

Returns 1 if the device created an Access Point network via the WIFIAP command, otherwise 0. Can be used within PRINT, IF, or variable assignments.

## Esempi

```
10 WIFIAP "Basic32" "password"
20 IF WIFIAPCONNECTED = 1 THEN PRINT "NETWORK CREATED"

Expected output:
NETWORK CREATED
```

Monitor the connection to an AP

```
10 PRINT WIFIAPCONNECTED

Expected output:
0 oppure 1
```

View AP status

## Note

- Works after WIFIAP
- Returns 1 if the Access Point network was created correctly
- Useful for activating actions upon connection

# WIFIAPDISCONNECT

## Sintassi

```
WIFIAPDISCONNECT
```

## Descrizione

Disable the active Access Point mode, created with WIFIAP. Always returns 0, compatible with IF and PRINT OR LET.

## Esempi

```
10 WIFIAPDISCONNECT
20 PRINT "AP OFF:"; WIFIAPDISCONNECT

Expected output:
AP OFF: 0
```

Disable the AP Wi-Fi network

```
10 IF WIFIAPDISCONNECT = 0 THEN PRINT "AP DEACTIVATED"

Expected output:
AP DEACTIVATED
```

Condition for verification

```
10 LET A = WIFIAPCONNECTED

Expected output:
0
```

LET

## Note

- Turn off the AP network
- Returns 0 as confirmation
- Useful for saving energy or safety

# WIFICONNECTED

## Sintassi

```
WIFICONNECTED
```

## Descrizione

Returns 1 if the device is connected to a Wi-Fi network via the WIFI command, otherwise 0. Can be used within PRINT, IF, or variable assignments.

## Esempi

```
10 WIFI "ssid" "password"
20 WAIT 2000
30 PRINT WIFICONNECTED

Expected output:
1
```

Print the connection status

```
10 IF WIFICONNECTED = 0 THEN PRINT "NOT CONNECTED"

Expected output:
NOT CONNECTED
```

Condition with IF

```
10 LET A = WIFICONNECTED
20 PRINT A

Expected output:
IF CONNECTED 1 IF NOT CONNECTED 0
```

LET

## Note

- Works after WIFI
- Returns 1 (connected) or 0 (not connected)
- Compatible with PRINT, LET, IF

# WIFIDISCONNECT

## Sintassi

```
WIFIDISCONNECT
```

## Descrizione

Disconnects the device from the Wi-Fi network previously connected with WIFI. It always returns 0 and can also be used in PRINT or IF OR LET.

## Esempi

```
10 WIFIDISCONNECT
20 PRINT "DISCONNECTED:"; WIFIDISCONNECT

Expected output:
DISCONNECTED: 0
```

Disconnect the Wi-Fi

```
10 IF WIFIDISCONNECT = 0 THEN PRINT "OK"

Expected output:
OK
Example 3 – LET use
10 LET A = WIFICONENCTED
RUN
Expected output:
0
```

Conditional use

## Note

- Only disconnects the WIFI network, not WIFIAP
- Returns 0 for compatibility with IF and PRINT and LET
- Useful for switching networks or restarting the connection

# WIRE

## Sintassi

```
WIRE <sda> <scl>
WIRE AUTO <sda> <scl>
WIRE OFF
```

## Descrizione

The WIRE command configures the I2C bus of the BASIC system and manages both immediate initialization and automatic activation when the ESP32 starts. 1. WIRE Sets the SDA and SCL pins. Initialize the I2C bus immediately. The configuration is not automatically saved for startup. All BASIC I2C commands require WIRE to have been executed at least once. 2. WIRE AUTO Sets the SDA and SCL pins. Initialize the I2C bus immediately. Save the configuration, which will be automatically reactivated each time the ESP32 is started. It remains valid even after a reset or power cycle. 3. WIRE OFF Disables automatic initialization of the I2C bus at startup. If the bus is active at that time, it remains active until restarted. It does not change the currently configured pins, it only eliminates autostart.

## Esempi

```
10 WIRE 21 22
20 SCANI2C

Possible output:
Scanning I2C bus...
Found device at 0x3C (60)
Total I2C devices: 1
```

Minimum configuration (runtime)

```
10 WIRE 21 22
20 INITRTC 0x68
30 PRINT "RTC ok!"
```

Configure and initialize the RTC

```
10 WIRE AUTO 21 22
20 PRINT "I2C ready!"

After a reboot of the ESP32, the bus will be automatically initialized with the same pins.
```

I2C auto-configuration at boot

```
10 WIRE OFF
20 PRINT "Auto-I2C disabled."

The next time you turn on WIRE it will NOT run automatically.
```

Disable automatic I2C

```
10 INITRTC 0x68

Expected output:
RUNTIME ERROR: INITRTC: WIRE must be called first
```

Error if WIRE is missing

## Note

- The command does not use commas → only spaces between arguments.

• SDA and SCL must be valid pin numbers on the ESP32 board.

• WIRE must be run before SCANI2C, INITRTC or any I2C sensor/actuator.

• WIRE AUTO saves the configuration in /prefs.cfg and automatically reapplies it at boot.

• WIRE OFF does not remove configured pins, only disables automatic mode.

• It is recommended to call WIRE (or WIRE AUTO) at the beginning of the program to avoid unexpected behavior.

# WS BRIGHTNESS

## Sintassi

```
WS BRIGHTNESS value
```

## Descrizione

Set the overall brightness (0–255) of the selected strip. It is composed with WS LBRIGHT (per-LED): the final value of each channel is:

```
output = raw * (LBRIGHT / 255) * (BRIGHTNESS / 255)
```

## Esempi

```
10 WS NEW 0 13 8
20 WS SELECT 0
30 WS FILL 255 0 0
40 WS BRIGHTNESS 40
50 WS SHOW
```

Dimmer on red

```
10 WS NEW 0 13 8
20 WS SELECT 0
30 WS FILL 0 120 255
40 B = 0
50 WS BRIGHTNESS B
60 WS SHOW
70 B = B + 5
80 IF B > 255 THEN GOTO 120
90 WAIT 20
100 GOTO 50
120 WS BRIGHTNESS B
130 WS SHOW
140 B = B - 5
150 IF B < 0 THEN GOTO 40
160 WAIT 20
170 GOTO 120
```

“Breathing” effect

## Note

- 0 = off, 255 = maximum overall brightness.
- The per-LED brightness (WS LBRIGHT) and the global brightness (WS BRIGHTNESS) are multiplied by each other:
- output = raw * (LBRIGHT/255) * (BRIGHTNESS/255)
- WS BRIGHTNESS does not send the data immediately: a WS SHOW (or WS SHOW ALL) is needed.

# WS BUS BRIGHTNESS

## Sintassi

```
WS BUS BRIGHTNESS value
```

## Descrizione

Set the global brightness (0–255) the same for all stripes in the BUS. Each strip of the BUS receives the same BRIGHTNESS, maintaining its own LBRIGHT per-LED.

## Esempi

```
10 WS NEW 0 13 5
20 WS NEW 1 27 5
30 WS BUS DEFINE 0 1
40 WS BUS FILL 0 255 0
50 WS BUS BRIGHTNESS 50
60 WS BUS SHOW
```

Global bus dimmer

```
10 WS NEW 0 13 5
20 WS NEW 1 27 5
30 WS BUS DEFINE 0 1
40 B = 0
50 WS BUS BRIGHTNESS B
60 WS BUS SHOW
70 B = B + 10
80 IF B > 180 THEN B = 180
90 WAIT 30
100 IF B < 180 THEN GOTO 50
```

Fade all over the bus

## Note

- Requires that the BUS has been defined with WS BUS DEFINE.
- Compatible with WS BUS LBRIGHT: both LBRIGHT and the BRIGHTNESS of the strip are valid for each virtual LED.

# WS BUS COUNT

## Sintassi

```
WS BUS COUNT var
```

## Descrizione

Writes in var the total number of LEDs in the BUS, i.e. the sum of the LEDs of all the strips included in WS BUS DEFINE.

## Esempi

```
10 WS NEW 0 13 5
20 WS NEW 1 27 7
30 WS BUS DEFINE 0 1
40 WS BUS COUNT N
50 PRINT "BUS LED=";N
```

Print bus count

```
10 WS NEW 0 13 5
20 WS NEW 1 27 7
30 WS BUS DEFINE 0 1
40 WS BUS COUNT N
50 POS = 0
60 WS BUS FILL 0 0 0
70 WS BUS SET POS 255 255 255
80 WS BUS SHOW
90 POS = POS + 1
100 IF POS >= N THEN POS = 0
110 WAIT 40
120 GOTO 60
```

Runners all over the bus

## Note

- If you recreate/delete strips (WS NEW, WS DELETE, WS DELETE ALL), run WS BUS DEFINE again before using the BUS again.

# WS BUS DEFINE

## Sintassi

```
WS BUS DEFINE id1 id2 [id3 ...]
```

## Descrizione

Defines the order of the strips connected in the BUS. The "virtual" indices of the BUS range from 0 to N-1, where N is the total number of LEDs (WS BUS COUNT). The order is given by the sequence of passed ids (id1, id2, ...).

## Esempi

```
10 WS NEW 0 13 6
20 WS NEW 1 27 6
30 WS BUS DEFINE 0 1
40 WS BUS FILL 0 0 255
50 WS BUS SHOW
```

Two strips in series

```
10 WS NEW 0 13 4
20 WS NEW 1 27 4
30 WS NEW 2 26 4
40 WS BUS DEFINE 2 0 1
50 WS BUS COUNT N
60 I = 0
70 IF I >= N THEN GOTO 120
80 IF I < N/3 THEN WS BUS SET I 0 255 0
90 IF I >= N/3 THEN IF I < (2*N)/3 THEN WS BUS SET I 0 0 255
100 IF I >= (2*N)/3 THEN WS BUS SET I 255 0 0
110 I = I + 1
115 GOTO 70
120 WS BUS SHOW
```

Three colored blocks

## Note

- All indicated ids must exist (WS NEW) and the strip must still be valid.
- You can redefine the BUS whenever you want: it overwrites the previous definition.

# WS BUS FILL

## Sintassi

```
WS BUS FILL R G B
```

## Descrizione

Set all virtual BUS LEDs to color (R,G,B) on internal buffers (without sending). The actual update takes place with WS BUS SHOW.

## Esempi

```
10 WS NEW 0 13 8
20 WS NEW 1 27 8
30 WS BUS DEFINE 0 1
40 WS BUS FILL 255 255 255
50 WS BUS SHOW
60 WAIT 300
70 WS BUS FILL 0 0 0
80 WS BUS SHOW
```

On/Off throughout the bus

```
10 WS NEW 0 13 8
20 WS NEW 1 27 8
30 WS BUS DEFINE 0 1
40 WS BUS FILL 255 120 0
50 WS BUS SHOW
60 WAIT 300
70 WS BUS FILL 0 80 255
80 WS BUS SHOW
90 WAIT 300
100 GOTO 40
```

Alternate between two palettes

## Note

- Useful for initializing the BUS to a base color before applying effects with WS BUS SET and WS BUS LBRIGHT.

# WS BUS LBRIGHT

## Sintassi

```
WS BUS LBRIGHT idx value
```

## Descrizione

Set the per-LED brightness (0–255) of the idx virtual LED in the BUS. The LED is chosen on the concatenated index, based on the definition of WS BUS DEFINE.

## Esempi

```
10 WS NEW 0 13 6
20 WS NEW 1 27 6
30 WS BUS DEFINE 0 1
40 WS BUS COUNT N
50 HEAD = 0
60 WS BUS FILL 255 0 0
70 I = 0
80 IF I >= N THEN GOTO 110
90 WS BUS LBRIGHT I 10
100 I = I + 1
105 GOTO 80
110 WS BUS LBRIGHT HEAD 255
120 T = 1
130 IF T > 5 THEN GOTO 170
140 K = HEAD - T
150 IF K < 0 THEN K = K + N
160 WS BUS LBRIGHT K 255 - T * 40
165 T = T + 1
167 GOTO 130
170 WS BUS SHOW
180 HEAD = HEAD + 1
190 IF HEAD >= N THEN HEAD = 0
200 WAIT 50
210 GOTO 60
```

Pointer with tail on BUS

```
10 WS NEW 0 13 6
20 WS NEW 1 27 6
30 WS BUS DEFINE 0 1
40 WS BUS COUNT N
50 LEVEL = 0
60 WS BUS FILL 0 60 0
70 I = 0
80 IF I >= N THEN GOTO 120
90 IF I < LEVEL THEN WS BUS LBRIGHT I 255
100 IF I >= LEVEL THEN WS BUS LBRIGHT I 20
110 I = I + 1
115 GOTO 80
120 WS BUS SHOW
130 LEVEL = LEVEL + 1
140 IF LEVEL > N THEN LEVEL = 0
150 WAIT 100
160 GOTO 60
```

Bargraph on BUS

### Note

- It is made up with WS BUS BRIGHTNESS:
- output = raw * (LBRIGHT/255) * (BRIGHTNESS/255) on the corresponding virtual LED.

# WS BUS SET

## Sintassi

```
WS BUS SET idx R G B
```

## Descrizione

Set the color of the virtual LED idx in the BUS (linked index), on the internal buffers. The actual update takes place with WS BUS SHOW.

## Esempi

```
10 WS NEW 0 13 5
20 WS NEW 1 27 7
30 WS BUS DEFINE 0 1
40 WS BUS COUNT N
50 POS = 0
60 WS BUS FILL 0 0 0
70 WS BUS SET POS 255 255 255
80 WS BUS SHOW
90 POS = POS + 1
100 IF POS >= N THEN POS = 0
110 WAIT 40
120 GOTO 60
```

Runner on the BUS

```
10 WS NEW 0 13 8
20 WS NEW 1 27 8
30 WS BUS DEFINE 0 1
40 WS BUS COUNT N
50 LEN = 4
60 POS = 0
70 WS BUS FILL 0 0 0
80 I = 0
90 IF I >= LEN THEN GOTO 130
100 IDX1 = POS + I
110 IF IDX1 >= N THEN IDX1 = IDX1 - N
120 WS BUS SET IDX1 255 0 0
125 I = I + 1
127 GOTO 90
130 I = 0
140 IF I >= LEN THEN GOTO 190
150 IDX2 = POS + 8 + I
160 IF IDX2 >= N THEN IDX2 = IDX2 - N
170 WS BUS SET IDX2 0 0 255
180 I = I + 1
185 GOTO 140
190 WS BUS SHOW
200 POS = POS + 1
210 IF POS >= N THEN POS = 0
220 WAIT 60
230 GOTO 70
```

Alternating segments

## Note

- It is the equivalent of WS SET, but works on the virtual index of the BUS, not on the local index of the single strip.

# WS BUS SHOW

## Sintassi

```
WS BUS SHOW
```

## Descrizione

Send the buffers of all the strips present in the BUS, applying for each LED: raw color (from WS BUS FILL / WS BUS SET) per-LED brightness (WS BUS LBRIGHT) global brightness (WS BUS BRIGHTNESS)

## Esempi

```
10 WS NEW 0 13 8
20 WS NEW 1 27 8
30 WS BUS DEFINE 0 1
40 WS BUS FILL 0 255 0
50 WS BUS SHOW
```

Enter after FILL

```
10 WS NEW 0 13 4
20 WS NEW 1 27 4
30 WS BUS DEFINE 0 1
40 WS BUS SET 0 255 0 0
50 WS BUS SET 7 0 0 255
60 WS BUS SHOW
```

Sending after many SETS

## Note

- In BUS mode it is preferable to use WS BUS SHOW instead of WS SHOW ALL.

# WS CLEAR

## Sintassi

```
WS CLEAR
```

## Descrizione

Immediately turns off all the LEDs of the selected strip:

```
resets the colors in the buffers (raw R,G,B = 0)
```

immediately sends the update to the strip

## Esempi

```
10 WS NEW 0 13 8
20 WS SELECT 0
30 WS CLEAR
```

Reset strip

```
10 WS NEW 0 13 8
20 WS SELECT 0
30 WS FILL 255 255 255
40 WS SHOW
50 WAIT 80
60 WS CLEAR
```

Flash

## Note

- After WS CLEAR the strip remains off until you set the colors again and run WS SHOW.
- WS CLEAR does not change WS BRIGHTNESS or WS LBRIGHT values.

# WS COUNT

## Sintassi

```
WS COUNT var
```

## Descrizione

Writes the number of LEDs of the selected strip in var.

## Esempi

```
10 WS NEW 0 13 10
20 WS SELECT 0
30 WS COUNT N
40 PRINT "LED strip 0 =";N
```

Count strip printing

```
10 WS NEW 0 13 10
20 WS SELECT 0
30 WS COUNT N
40 I = 0
50 WS FILL 0 0 0
60 WS SET I 255 255 255
70 WS SHOW
80 I = I + 1
90 IF I >= N THEN I = 0
100 WAIT 50
110 GOTO 50
```

Safe runner

## Note

- If no strip is selected (WS SELECT), the command generates an error at runtime.

# WS DELETE

## Sintassi

```
WS DELETE id
```

## Descrizione

Removes the id strip and frees the associated memory: if it was in the BUS, it is removed from the BUS (which is recalculated) if it was the selected strip, the selection is reset

## Esempi

```
10 WS NEW 0 13 8
20 WS NEW 1 27 8
30 WS DELETE 1
```

Delete a strip

```
10 WS NEW 0 13 8
20 WS DELETE 0
30 WS NEW 0 13 16
40 WS SELECT 0
50 WS FILL 0 0 255
60 WS SHOW
```

Recreate with another count

## Note

- If you want to be sure to turn off the real strip first, use:
- WS SELECT id + WS CLEAR, then WS DELETE id.

# WS DELETE ALL

## Sintassi

```
WS DELETE ALL
```

## Descrizione

Removes all the created strips, empties the BUS and resets the selected strip.

## Esempi

```
10 WS NEW 0 13 8
20 WS NEW 1 27 8
30 WS DELETE ALL
```

Total reset

```
10 WS DELETE ALL
20 WS NEW 0 13 12
30 WS SELECT 0
40 WS FILL 255 0 0
50 WS SHOW
```

Start from scratch

## Note

- After WS DELETE ALL you must recreate the strips with WS NEW and, if you need the bus, redefine it with WS BUS DEFINE.

# WS FILL

## Sintassi

```
WS FILL R G B
```

## Descrizione

Set all LEDs of the selected strip to the color (R,G,B) in the internal buffers. To physically update the strip you need a WS SHOW (or WS SHOW ALL).

## Esempi

```
10 WS NEW 0 13 8
20 WS SELECT 0
30 WS FILL 255 0 0
40 WS BRIGHTNESS 30
50 WS SHOW
```

Pale red

```
10 WS NEW 0 13 8
20 WS SELECT 0
30 WS FILL 255 255 255
40 WS SHOW
50 WAIT 150
60 WS FILL 0 0 0
70 WS SHOW
80 WAIT 150
90 GOTO 30
```

Blink

## Note

- To work on individual LEDs use WS SET.

# WS LBRIGHT

## Sintassi

```
WS LBRIGHT idx value
```

## Descrizione

Set the per-LED brightness (0–255) of the idx LED on the selected strip (buffer only). It is a scaling factor that does not change the color, only the relative intensity of the LED.

## Esempi

```
10 WS NEW 0 13 16
20 WS SELECT 0
30 WS FILL 0 120 255
40 I = 0
50 IF I >= 16 THEN GOTO 80
60 WS LBRIGHT I I*16
70 I = I + 1
75 GOTO 50
80 WS SHOW
```

Intensity scale

```
10 WS NEW 0 13 12
20 WS SELECT 0
30 COUNT = 12
40 POS = 0
50 WS FILL 255 0 0
60 I = 0
70 IF I >= COUNT THEN GOTO 100
80 WS LBRIGHT I 10
90 I = I + 1
95 GOTO 70
100 WS LBRIGHT POS 255
110 T = 1
120 IF T > 5 THEN GOTO 160
130 K = POS - T
140 IF K < 0 THEN K = K + COUNT
150 WS LBRIGHT K 255 - T * 40
155 T = T + 1
157 GOTO 120
160 WS SHOW
170 POS = POS + 1
180 IF POS >= COUNT THEN POS = 0
190 WAIT 50
200 GOTO 50
```

Pointer with tail

## Note

- It does not change the “raw” color, only its intensity.
- It is composed with WS BRIGHTNESS according to:
- output = raw * (LBRIGHT/255) * (BRIGHTNESS/255)

# WS NEW (NeoPixel WS2812B)

## Sintassi

```
WS NEW id pin count
```

## Descrizione

Create/initialize a NeoPixel strip with id identifier, connected to pins and with LED count. If id already exists, the strip is recreated with the new parameters.

## Esempi

```
10 WS NEW 0 13 8
20 WS NEW 1 27 12
```

Two strips

```
10 WS NEW 0 13 8
20 WS NEW 0 13 16
```

Recreate with more LEDs

## Note

- Stable 5 V power supply, GND in common with the ESP32.
- Recommended resistance ~330 Ω in series on the DATA line.
- Recommended 470–1000 µF capacitor near the 5 V input of the strip.
- For long strips, inject 5 V at multiple points along the strip.

# WS SELECT

## Sintassi

```
WS SELECT id
```

## Descrizione

Select the strip on which all WS ... non-BUS commands operate (FILL, SET, SHOW, BRIGHTNESS, LBRIGHT, COUNT, CLEAR, etc.).

## Esempi

```
10 WS NEW 0 13 8
20 WS NEW 1 27 8
30 WS SELECT 1
40 WS FILL 0 0 255
50 WS SHOW
```

Work on id=1

```
10 WS NEW 0 13 8
20 WS NEW 1 27 8
30 WS SELECT 0
40 WS FILL 255 0 0
50 WS SHOW
60 WS SELECT 1
70 WS FILL 0 0 255
80 WS SHOW
```

Alternate two strips

## Note

- If no strip is selected (WS SELECT not called or strip removed), WS ... commands that require a strip fail with error.

# WS SET

## Sintassi

```
WS SET idx R G B
```

## Descrizione

Set the “raw” color of the idx LED on the selected strip (buffer only). To update the strip you need a WS SHOW (or WS SHOW ALL).

## Esempi

```
10 WS NEW 0 13 8
20 WS SELECT 0
30 WS SET 0 255 0 0
40 WS SET 1 255 127 0
50 WS SET 2 255 255 0
60 WS SET 3 0 255 0
70 WS SET 4 0 180 255
80 WS SET 5 0 0 255
90 WS SET 6 148 0 211
100 WS SHOW
```

Manual rainbow 7 LEDs

```
10 WS NEW 0 13 12
20 WS SELECT 0
30 PH = 0
40 WS FILL 0 0 0
50 I = 0
60 IF I >= 12 THEN GOTO 110
70 T = I + PH
80 IF T >= 3 THEN T = T - 3
90 IF T = 0 THEN WS SET I 255 100 0
100 I = I + 1
105 GOTO 60
110 WS SHOW
120 PH = PH + 1
130 IF PH >= 3 THEN PH = 0
140 WAIT 80
150 GOTO 40
```

Theater chase (period 3)

## Note

- The idx index goes from 0 to count-1, where count is the number of LEDs in the strip (WS COUNT).

# WS SHOW

## Sintassi

```
WS SHOW
WS SHOW ALL
```

## Descrizione

WS SHOW → sends the buffers of the selected strip. WS SHOW ALL → sends the buffers of all the created strips (not BUS). In both cases, per-LED WS LBRIGHT and global WS BRIGHTNESS are applied.

## Esempi

```
10 WS NEW 0 13 8
20 WS SELECT 0
30 WS FILL 0 255 0
40 WS SHOW
```

Current strip

```
10 WS NEW 0 13 8
20 WS NEW 1 27 8
30 WS SELECT 0
40 WS FILL 255 0 0
50 WS SELECT 1
60 WS FILL 0 0 255
70 WS SHOW ALL
```

All the strips

## Note

- In BUS mode, prefer WS BUS SHOW instead of WS SHOW ALL.

# Indice (comandi)

ABS(x) — pag. 7

ACS CALIB SETOFFSET — pag. 8

ACS CALIB SHOW — pag. 9

ACS CALIB ZERO — pag. 10

ACS INIT (acs712) — pag. 11

ACS READ — pag. 12

ACS RMS — pag. 13

ACS SAMPLES — pag. 14

ACS SENS — pag. 15

ACS VREF — pag. 16

ACS WATCH — pag. 17

ADC CAL GAIN — pag. 19

ADC CAL MEASURE — pag. 20

ADC CAL OFFSET — pag. 21

ADC CAL REF — pag. 22

ADC CAL RESET — pag. 23

ADC CAL STATUS — pag. 24

ALEXABRIGHT — pag. 25

ALEXACALL — pag. 27

ALEXASTATE — pag. 28

AND, OR, NOT — pag. 29

AREAD(p) — pag. 31

ASC(A$) — pag. 32

AUTORUN — pag. 33

AWRITE(p, v) — pag. 34

BREAKPIN — pag. 35

BT AT — pag. 36

BT ATMODE OFF — pag. 37

BT ATMODE ON — pag. 38

BT AVAILABLE — pag. 39

BT END — pag. 40

BT FLUSH — pag. 41

BT INIT (HC-05/HC-06) — pag. 42

BT PRINT — pag. 43

BT READ — pag. 44

BT READLN — pag. 45
BT SEND — pag. 46
BT SETBAUD — pag. 47
BT SETNAME — pag. 48
BT SETPIN — pag. 49
BT TIMEOUT — pag. 50
BT VERSION — pag. 51
CALLFUNC — pag. 52
CARDKB — pag. 54
CHAIN — pag. 55
CHR$(x) — pag. 57
CLS — pag. 58
CLSANSI — pag. 59
COPY — pag. 60
COS(x) — pag. 61
DATA — pag. 62
DATED — pag. 63
DATEM — pag. 64
DATEY — pag. 65
DEBUGMEM — pag. 66
DEF FN — pag. 67
DEL — pag. 68
DELAY n — pag. 69
DELN — pag. 71
DELVAR — pag. 73
DEV AUTO ON/OFF — pag. 74
DEV CLEAR — pag. 75
DEV DISPLAY — pag. 76
DEV FONT — pag. 77
DEV ILI SET — pag. 78
DEV MAPCOLS — pag. 79
DEV MAPROWS — pag. 80
DEV ON / OFF — pag. 81
DEV PS2 LAYOUT — pag. 82
DEV PS2 OFF — pag. 83
DEV PS2 ON — pag. 84
DEV PS2 SET — pag. 85

DEV SERIAL ON / OFF — pag. 86
DEV STATUS — pag. 87
DHTCALIBRESET — pag. 88
DHTCALIBSET — pag. 89
DHTCALIBSHOW — pag. 91
DHTINIT (DHT11-DHT22) — pag. 92
DHTREAD — pag. 93
DIM — pag. 94
DIR — pag. 97
DLEVEL(p) — pag. 98
DO — pag. 99
DO BLOCK — pag. 100
DREAD(p) — pag. 101
DWRITE(p v) — pag. 102
ELSE — pag. 103
EXP(x) — pag. 104
FCLOSE — pag. 105
FNname(...) — pag. 106
FOR/NEXT — pag. 107
FORMAT — pag. 109
FPRINT — pag. 110
FREAD$ — pag. 111
FREEMEM — pag. 112
FSDEFAULT — pag. 113
FUNC / ENDFUNC — pag. 114
GET — pag. 116
GOSUB n — pag. 117
GOTO n — pag. 119
HTMLOBJ — pag. 121
HTMLSTART — pag. 123
HTTP GET — pag. 124
HTTP HEADERS — pag. 125
HTTP JSON GET — pag. 126
HTTP POST — pag. 127
IF ... THEN [ELSE] — pag. 128
ILI CIRCLE — pag. 130
ILI CLEAR — pag. 131

ILI DRAW — pag. 132
ILI FILLCIRCLE — pag. 133
ILI FILLRECT — pag. 134
ILI INIT (ILI9341) — pag. 135
ILI INIT TOUCH (XPT2046) — pag. 136
ILI LED — pag. 138
ILI LINE — pag. 139
ILI PIXEL — pag. 140
ILI RECT — pag. 141
ILI SETBGCOLOR — pag. 142
ILI SPEED — pag. 143
ILI SPRITE CLEAR — pag. 144
ILI SPRITE COLLISION — pag. 145
ILI SPRITE COLLISIONC — pag. 147
ILI SPRITE DELETE — pag. 149
ILI SPRITE DRAW — pag. 150
ILI SPRITE FLIP — pag. 151
ILI SPRITE HIDE — pag. 152
ILI SPRITE MOVE — pag. 153
ILI SPRITE NEW — pag. 154
ILI SPRITE PAL — pag. 156
ILI SPRITE ROTATE — pag. 157
ILI SPRITE SETBITMAP — pag. 158
ILI SPRITE SETCHAR — pag. 159
ILI SPRITE SETNUM — pag. 160
ILI SPRITE SETTEXT — pag. 161
ILI SPRITE SHOW — pag. 162
ILI TEXT — pag. 163
ILI TOUCH AREA — pag. 164
ILI TOUCH CALIBRATE — pag. 165
ILI TOUCH SPRITE — pag. 167
INCLUDE — pag. 168
INITRTC — pag. 169
INITSD — pag. 170
INPUT — pag. 172
INT(x) — pag. 174
IP — pag. 175

IPAP — pag. 176
IR HALT — pag. 177
IR INIT — pag. 178
IR SEND — pag. 179
IR WAIT — pag. 180
IRDB ADD — pag. 181
IRDB CLEAR — pag. 182
IRDB CODE — pag. 183
IRDB DELETEFILE — pag. 184
IRDB INIT — pag. 185
IRDB LIST — pag. 186
IRDB LOAD — pag. 187
IRDB REMOVE — pag. 188
IRDB SAVE — pag. 189
IRDB SEND — pag. 190
KEYPOLL — pag. 191
KPAVAILABLE — pag. 192
KPFLUSH — pag. 193
KPINIT (Matrix keypad) — pag. 194
KPMAP — pag. 195
KPMODE — pag. 196
KPREAD — pag. 197
KPWAIT — pag. 198
LCD BACKLIGHT (solo I²C) — pag. 199
LCD CLEAR — pag. 200
LCD CURSOR — pag. 201
LCD HOME — pag. 202
LCD I2C INIT (16X2 – 16X4) — pag. 203
LCD PAR INIT (16X2 – 16X4) — pag. 204
LCD PRINT — pag. 205
LCD ROW — pag. 206
LCD SCROLL — pag. 207
LCD SETCUR — pag. 208
LEFT$(A$, N) — pag. 209
LEN(A$) — pag. 210
LET — pag. 211
LIST — pag. 212

LISTVARS — pag. 214
LOAD — pag. 215
LOADVAR — pag. 216
LOG(x) — pag. 217
MAC — pag. 218
MEMCLEAN — pag. 220
MID$(A$, start, len) — pag. 222
MILLIS — pag. 224
MQTTAUTOPOLL — pag. 226
MQTTCONNECT — pag. 227
MQTTPUB — pag. 228
MQTTSUB — pag. 229
NEW — pag. 230
NOTONE — pag. 231
NOWCLR — pag. 232
NOWINIT (ESP-NOW) — pag. 233
NOWPOLL — pag. 234
NOWSEND — pag. 236
NRF AVAILABLE — pag. 237
NRF CONFIG — pag. 238
NRF FLUSH — pag. 239
NRF INIT (nRF24L01) — pag. 240
NRF POWERDOWN — pag. 241
NRF POWERUP — pag. 242
NRF READ — pag. 243
NRF SEND — pag. 244
NRF SET RXADDR — pag. 245
NRF SET TXADDR — pag. 246
NRF START RX — pag. 247
NRF STOP RX — pag. 248
OLED CAMERA CLAMP — pag. 249
OLED CAMERA FOLLOW SPRITE — pag. 251
OLED CAMERA GET — pag. 253
OLED CAMERA SET — pag. 255
OLED CIRCLE — pag. 257
OLED CLEAR — pag. 258
OLED DRAW — pag. 259

OLED DRAWRECT — pag. 260
OLED FILLCIRCLE — pag. 262
OLED FILLRECT — pag. 263
OLED INIT (DISPLAY SSD1306-SSD1309) — pag. 264
OLED INVERT ON / OFF — pag. 265
OLED LINE — pag. 266
OLED PIXEL — pag. 267
OLED RECT — pag. 268
OLED SPRITE CLEAR — pag. 269
OLED SPRITE COLLISION — pag. 270
OLED SPRITE COLLISIONC — pag. 272
OLED SPRITE DELETE — pag. 274
OLED SPRITE DRAW — pag. 275
OLED SPRITE DRAWWORLD — pag. 277
OLED SPRITE HIDE — pag. 278
OLED SPRITE MOVE — pag. 279
OLED SPRITE NEW — pag. 280
OLED SPRITE SETBITMAP — pag. 282
OLED SPRITE SETCHAR — pag. 283
OLED SPRITE SETNUM — pag. 284
OLED SPRITE SETTEXT — pag. 285
OLED SPRITE SHOW — pag. 286
OLED TEXT — pag. 287
OLED TILE ANIM NEW — pag. 288
OLED TILE ANIM TICK — pag. 290
OLED TILE CLEAR — pag. 292
OLED TILE FLAGS — pag. 294
OLED TILE HITBOX — pag. 296
OLED TILE NEW — pag. 298
OLED TILEMAP ANIM — pag. 300
OLED TILEMAP ANIMFILL — pag. 302
OLED TILEMAP CLEAR — pag. 304
OLED TILEMAP COLLIDE — pag. 305
OLED TILEMAP DELETE — pag. 307
OLED TILEMAP DRAW — pag. 308
OLED TILEMAP DRAWRECT — pag. 310
OLED TILEMAP FILL — pag. 312

OLED TILEMAP GET — pag. 313
OLED TILEMAP HITINFO — pag. 315
OLED TILEMAP LOAD — pag. 317
OLED TILEMAP MOVE — pag. 319
OLED TILEMAP NEW — pag. 321
OLED TILEMAP RAYCAST — pag. 322
OLED TILEMAP SET — pag. 324
OLED TILEMAP TILEAT — pag. 326
OLED TILEMAP TRIGGER — pag. 328
ON x GOTO ... — pag. 331
OPEN — pag. 333
PEEK — pag. 334
PINMODE — pag. 335
POKE — pag. 336
PRINT — pag. 338
PRINTUSING — pag. 339
READ — pag. 341
REBOOT — pag. 343
RENAME — pag. 344
RESTORE — pag. 345
RETURN — pag. 347
RFID HALT — pag. 349
RFID INIT (MFRC522) — pag. 350
RFID PRESENT — pag. 351
RFID READBLOCKABS — pag. 352
RFID READUID — pag. 353
RFID WAITUID — pag. 354
RFID WRITEBLOCKABS — pag. 355
RFIDDB ADD — pag. 356
RFIDDB CLEAR — pag. 357
RFIDDB COUNT — pag. 358
RFIDDB DELETEFILE — pag. 359
RFIDDB EXISTS — pag. 360
RFIDDB INIT — pag. 361
RFIDDB LABEL — pag. 362
RFIDDB LIST — pag. 363
RFIDDB LOAD — pag. 364

RFIDDB REMOVE — pag. 365
RFIDDB SAVE — pag. 366
RIGHT$(A$, N) — pag. 367
RND(x) / RND(a, b) — pag. 368
RREAD — pag. 370
RUN — pag. 371
SAVE — pag. 372
SAVEVAR — pag. 373
SCANI2C — pag. 374
SDFREE — pag. 375
SERVOATTACH — pag. 376
SERVODETACH — pag. 377
SERVOMAX — pag. 378
SERVOREAD — pag. 379
SERVOSTATUS — pag. 380
SERVOWRITE — pag. 381
SERVOWRITEMICROS — pag. 382
SETDATE — pag. 383
SETTIME — pag. 384
SFXDEF — pag. 385
SFXPLAY — pag. 386
SFXSWEEP — pag. 387
SIDINIT — pag. 388
SIDMADD — pag. 389
SIDMCLR — pag. 390
SIDMPLAY — pag. 391
SIDMSTOP — pag. 392
SIDVOICE — pag. 393
SIDVOL — pag. 394
SIN(x) — pag. 395
SPC(n) — pag. 396
SPIFREE — pag. 397
SQR(x) — pag. 398
SR595 CLEAR — pag. 399
SR595 FILL — pag. 400
SR595 INIT (SHIFT REGISTER 74HC595) — pag. 401
SR595 SET — pag. 402

SR595 SHOW — pag. 403
SR595 WRITE — pag. 404
STARTFUNC / STOPFUNC — pag. 405
STOP, CONT, END — pag. 406
STR$(x) o STR$(x, n) — pag. 408
SYNCNTP — pag. 409
TAN(x) — pag. 410
TCSFILTER — pag. 411
TCSINIT (Sensore colore TCS230/TCS3200) — pag. 412
TCSLED — pag. 413
TCSREAD — pag. 414
TCSRGB — pag. 415
TCSSCALE — pag. 416
TI — pag. 417
TIMEH — pag. 418
TIMEM — pag. 419
TIMES — pag. 420
TIMEZONE — pag. 421
TO STEP NEXT — pag. 423
TONE — pag. 424
ULTRA INIT (SENSORE ULTRASUONI HC-SR04) — pag. 425
ULTRA READ — pag. 426
ULTRA TIMEOUT — pag. 427
VAL(A$) — pag. 428
VERIFY — pag. 430
VREAD(p) — pag. 431
WAIT n — pag. 432
WIFI — pag. 434
WIFIAP — pag. 436
WIFIAPCONNECTED — pag. 437
WIFIAPDISCONNECT — pag. 438
WIFICONNECTED — pag. 439
WIFIDISCONNECT — pag. 440
WIRE — pag. 441
WS BRIGHTNESS — pag. 443
WS BUS BRIGHTNESS — pag. 444
WS BUS COUNT — pag. 445

WS BUS DEFINE — pag. 446
WS BUS FILL — pag. 447
WS BUS LBRIGHT — pag. 448
WS BUS SET — pag. 450
WS BUS SHOW — pag. 451
WS CLEAR — pag. 452
WS COUNT — pag. 453
WS DELETE — pag. 454
WS DELETE ALL — pag. 455
WS FILL — pag. 456
WS LBRIGHT — pag. 457
WS NEW (NeoPixel WS2812B) — pag. 458
WS SELECT — pag. 459
WS SET — pag. 460
WS SHOW — pag. 461