V15.1.15

LED-BASIC

COMPONENTS, EDITOR, COMMAND SET

Erwin Reuß and Folker Bar
Unofficial English translation by Doug Paulley

Contents

LED-BASIC?

Behind the idea of LED-Basic is many years of experience in dealing with light emitting diodes and their control via PC or independent modules, such as the various LED players or LED controllers. All had the problem that the data for the LEDs had to be generated by a program - usually either JINX! or GLEDiator. These are mainly designed for LED matrices, and are sometimes quite cumbersome to configure and operate. To be able to use the data generated by the program on an LED player, they had to be copied to an SD card or USB stick, and inserted into the player. For small, simple projects, this process is too expensive, and limits one to the effects stored in the program. One's own creations are only possible to a limited extent, by changing the effect parameters.

LED-Basic is mainly designed for the control of single LEDs, or LED strips. The maximum number of LEDs that can be controlled depends largely on the performance of the microcontroller, and the memory size of the LED-Basic components used.

LED-Basic does not need to be used solely for applications with LEDs. It's also possible, for example, to implement controls that query sensors and emit signals on IO ports depending on the resulting measurements (see Temperature-Sensor-Interface).

Extensions?
LED-Basic should be easy to use, and the commands should be easy for every user to remember. For this reason, we deliberately avoided an elaborate user interface and BASIC commands with cumbersome syntax. Useful extensions proposed by users can be safely integrated into future versions. Your cooperation is therefore welcome. Send us your suggestions and any error messages to feedback@led-basic.de.

Installation and program start

Download the installation package "LedBasic_aa.b.c.exe" from our server and install it on your PC (aa.b.c = current version).

In the "SAMPLES" subdirectory of the installation path, there are some sample files for various LED-Basic Components. NB: you should not store your own programs in this directory because their contents may be deleted or overwritten during an update.

The installation directory also contains the USB driver file, and this manual in PDF format.

Windows 10 and Security

If there are any error messages such as "Access Denied" in Windows 10, please start the program with Administrator rights (right click on the LED-Basic icon and select "Run as Administrator").

Automatic and manual Updates

The LED-Basic Editor automatically checks if a new version is available when the program is started (if this option is enabled in the configuration). You can also check this at any time using Help menu > Check for update. If an updated version exists, it will be installed and started automatically. For this to work, of course, your PC must be connected to the Internet. You can also download the current version from our server and install it manually.

This PDF guide is also updated together with the latest software. This is also downloaded during an update and replaces the old version.

LED-Basic Editor

The LED-Basic Editor is also a so-called "Tokeniser". It translates BASIC code from plain text into "tokens". This saves space on the LED-Basic hardware and brings an enormous speed advantage during execution. The "Tokeniser" checks the syntax of the code entered, converts jump commands and labels to absolute addresses, checks for the correct use of variables and checks the number of parameters for LED and IO commands.

It is assumed that the user is familiar with the BASIC programming language, and understands the differences between variables, constants and expressions. If not, there is plenty of literature on the Internet about this rather simple programming language. The syntax and some special features of the LED-Basic are described below in the sections on keywords and additional commands. Given time, there will also be many sample programs for you to examine the programming techniques.

The version of the LED-Basic Editor (e.g. 15.1.9) is made up of the current BASIC version (V15) and the editor version (v1.9). If the editor version does not match the LED-Basic component's BASIC version, LED-Basic automatically performs a BIOS update.

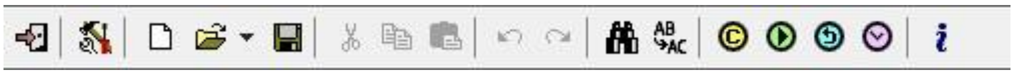
Editor functions

Text editor operation should be familiar. The LED-Basic Editor supports the standard cut, copy and paste functions, as well as undo and redo functions of up to 100 steps. When you save a program file, LED-Basic always creates a backup copy of the existing file (with the extension .bak).

The editor interface language can be selected using the menu item "language". Note that the language of the file or search dialogs is set depending on your installed version of Windows.

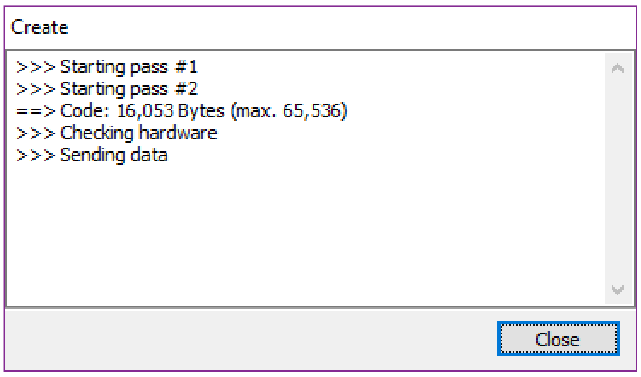
Because LED-Basic does not currently support "include" files, the editor can only open or edit one file at a time.

The editor's main functions can be accessed via the menu, the icon bar or via key shortcuts.



Ⓒ	F9	Translates the BASIC code and displays any errors in the Info window.
▶	SHIFT + F9 or F11	Translates the BASIC code and displays any error info windows. After successful translation, the code is transferred to the LED-Basic component.
↺	F12	Triggers the program on the LED-Basic component to restart.
⌄	Shift + F11	Updates real-time clock (if supported by the LED-Basic component).

Info window

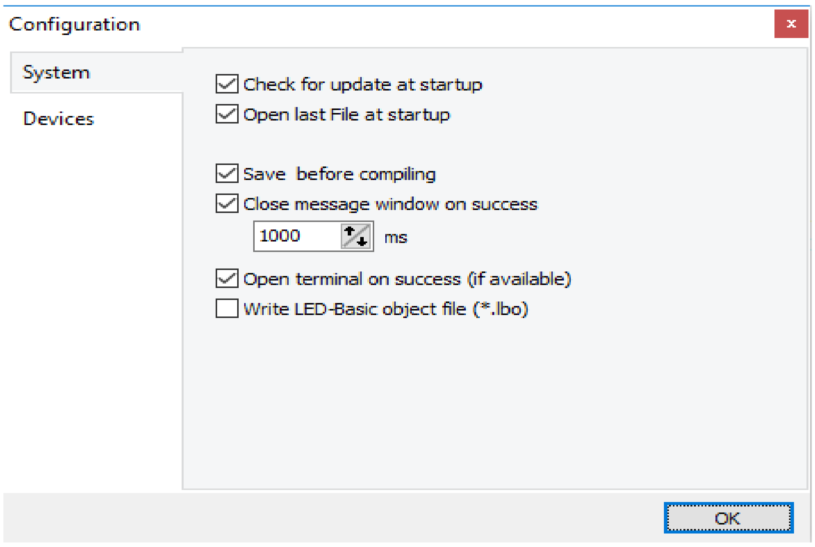


```
>>> Starting pass #1
>>> Starting pass #2
==> Code: 16,053 Bytes (max. 65,536)
>>> Checking hardware
>>> Sending data
```

The Info window displays any errors found during translation of the BASIC source code, or communication with the LED-Basic component. If the operation did not generate any errors, the info window will either automatically close after the time specified in Settings or can be closed manually. The window will not automatically close if an error has occurred and must be closed manually.

Configuration

System



- [✓] Check for update at startup
- [✓] Open last File at startup
- [✓] Save before compiling
- [✓] Close message window on success
  1000 ms
- [✓] Open terminal on success (if available)
- [ ] Write LED-Basic object file (*.lbo)

Check for new version at program start:

If there is an Internet connection when the LED-Basic Editor is started, it will automatically check whether there is a new version. This can then be downloaded and installed. In addition, a manual check can be made at any time using the Help menu.

Load last used file at program start:

The last edited file is opened again when the program starts. If this menu item is not selected, the LED-Basic Editor is opened with a blank document.

Save before compiling:

Before creating/compiling the BASIC code, the current file is automatically saved. In addition, a backup of the previous file is created (*. bak).

Close info window on success:

If the BASIC code is correctly translated, the info window automatically closes after the set time. If this option is not selected, the info window remains open until it is manually closed.

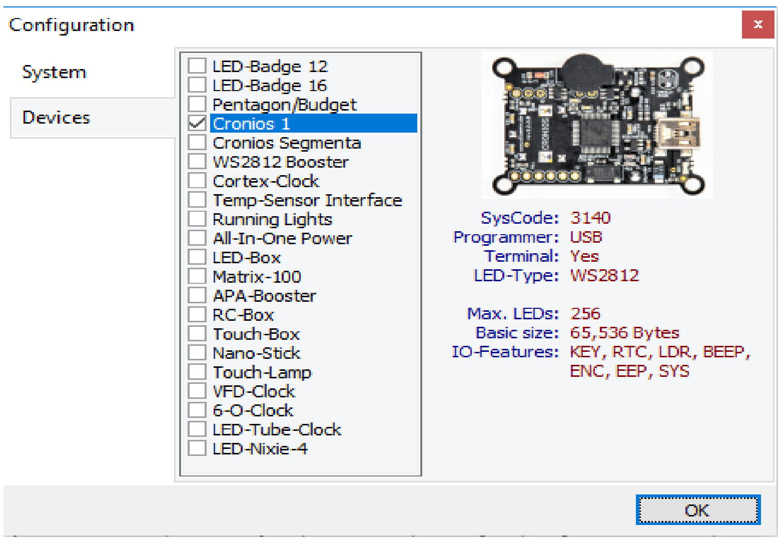
Open Terminal on success:

If the BASIC code is correctly translated and transferred to the controller without any errors, the terminal window will be opened to display print and error output (if this is supported by the LED-Basic component.)

Write LED-Basic object File (*. LBO):

This causes the object file generated by the Tokeniser to be stored in the same directory as the BASIC code. This can then be transferred to the LED-Basic component using an external programming tool (in development).

Components



- ☐ LED-Badge 12
- ☐ LED-Badge 16
- ☐ Pentagon/Budget
- ☑ Cronios 1
- ☐ Cronios Segmenta
- ☐ WS2812 Booster
- ☐ Cortex-Clock
- ☐ Temp-Sensor Interface
- ☐ Running Lights
- ☐ All-In-One Power
- ☐ LED-Box
- ☐ Matrix-100
- ☐ APA-Booster
- ☐ RC-Box
- ☐ Touch-Box
- ☐ Nano-Stick
- ☐ Touch-Lamp
- ☐ VFD-Clock
- ☐ 6-O-Clock
- ☐ LED-Tube-Clock
- ☐ LED-Nixie-4

SysCode:	3140
Programmer:	USB
Terminal:	Yes
LED-Type:	WS2812
Max. LEDs:	256
Basic size:	65,536 Bytes
IO-Features:	KEY, RTC, LDR, BEEP, ENC, EEP, SYS

Select the appropriate LED-Basic component from the list. You can view all the components for selection. Important! Only the component marked by a tick is used.

Under the component photo you will find its specifications:

System code:

This code is used to identify the LED-Basic Component. If the connected component does not match the system code, LED-Basic will refuse to transfer the BASIC code to it.

Programmer:

This shows whether a programming adapter (e.g. Prog_SB) must be used to transfer the BASIC code to the component.

Terminal:

Can the LED-Basic component show print and error output in the terminal?

LED Type:

Which led types are supported by the component?

RGB	Simple RGB LEDs without PWM, 7 different colours are possible.
WS2812	Serial LEDs with integrated PWM controller of type WS2811 (RGB or RGBW), also SK6812 or APA104/106. Characteristic: 3 Connection cables (+ 5V, DATA, GND)
APA102	Serial LEDs with an integrated PWM controller of type WS2801 (RGB). Characteristic: 4 Connection cables (+ 5V, DATA, CLOCK, GND)
7-SEGMENT	7-segment display, e.g. in the Cortex-Clock
SIMPLE LED	Single light emitting diodes, e.g. in Running Light with 16 LEDs

IO Functions:

What IO functions does the component support?

KEY	Keys and input ports (KEY_x)
PORT	Output ports (PORT_x)
RTC	Clock
LDR	Brightness sensor
TEMP	Temperature sensor (1 x DS18B20), only temperature sensing
XTEMP	Temperature sensor (max. 8 x DS18B20), temperature and parameter query
IR	Infrared Remote-Control sensor
BEEP	Speaker output
ENC	Encoder
POTI	Analogue Knobs (potentiometers)
ADC	Analog value queries (e.g. battery voltage)
EEP	EEPROM, stores data that remains even after the power supply is disconnected.
SYS	Calling system functions, reading or writing system variables

LED-Basic Main Instructions

The language of LED-Basic is limited to a few functions brought from the BASIC programming language. This is extended by special LED and IO commands. Converting the commands to "tokens" ensures a high processing speed of over 10000 lines per second (depending on the microcontroller used).

Fundamental rules for LED-Basic

- Case insensitive – upper and lower case are not distinguished
- There are 26 global variables A... z or A... Z (A is identical to a)
- LED-Basic numbers use a maximum of 16 bits -32768... + 32767
- Floating-point values are not supported.
- There are no string variables
- Supports a maximum of 4 nested GOSUB routines
- Supports a maximum of 4 nested FOR-NEXT loops
- All commands after END are ignored
- All lines starting ' (apostrophe) are ignored (as with REM)
- The terminal, e.g. in the LED-Basic Editor, displays print output and error messages if supported by the LED-Basic component

There are no line numbers in LED-Basic, unlike other basic variants. Line number labels are still used as jump marks for GOTO, GOSUB, or as an index for data values. A label consists of a number between 0 and 32766, followed by a colon.

Examples

```
1000:
  i = 123
  ...
  Return
```

```
30000: a = 0
  ...
  Return
```

The line number order is unimportant, but line numbers must be unique. The next command can be on the same or the following line.

Notes on using variables

The 26 variables a..z (A..Z) are globally valid. There are no local variables, if you are using variables and subroutines, be careful not to use the same variables as in the calling routine. Make a list or write the variables used as a comment in the code.

Entering number values:

Decimals:	0 1234 -2000
Hexadecimal numbers:	0x1234 0xAB
Binary numbers:	0b10101010 0b10

Arithmetic operators:

+	Addition, 3 + 4
-	Subtraction, 5 - 2
/	Division, 9 / 3
*	Multiplication, 10 * 5
%	Modulo (division remainder), 6 % 5

Bit operators:

|	bitwise OR operator, 0X05 | 0x80
&	bitwise AND operator, 0xAE & 0x07

Comparison operators:

<	less than
>	greater than
=	equal to
<>	not equal to
<=	less than or equal to
>=	greater than or equal to

Logical operators:

or	logical OR: if a=1 or a=3 then...
and	logical AND: if a=2 and b=3 then...

Syntax description

The following descriptors are used in the LED-Basic command manual:

- *VAL* = Numerical value (123, 0x100, 0b101010)
- *STR* = string, must be enclosed in quotation marks ("Hello World")
  (only possible in the PRINT command)
- *EXPR* = An expression with numerical result (1 + 2, a * 3)
- *VAR* = Variable (a, A, z, Z)
- *REL* = comparison operation (a > 4, x = y)
- *LABEL* = Jump target or data base (1234:)
- ... = arbitrary statement
- <xxx>= Input required</xxx>
- xxx | yyy = xxx or yyy are possible
- <VAL|EXPR|VAR> = here you can have a numerical value, an expression or a variable
- [XXX] = can be added optionally

REM

```
rem...
´ ...
```

Remark, Comment. All instructions after **rem** or ' (apostrophe) are ignored.

END

```
end
```

All commands behind **end** are ignored

LET

```
[let] <var>=<VAL|EXPR|VAR>
<var>=<VAL|EXPR|VAR>
```

Assigns a value to a variable.

let Is optional and can be omitted.

FOR-NEXT LOOP

```
for <VAR>=<VAL|EXPR|VAR> to|downto <VAL|EXPR|VAR> [step
<VAL|EXPR|VAR>]
...
next VAR
```

Please ensure that the value **to** is never smaller than the value before **to,** and that the value after **downto** is never greater than the value before **downto.**

step is always positive, even when using **downto**.

If **step** is not specified, the default increment is 1.

It is possible to nest a maximum of 4 FOR-NEXT loops. Always ensure that the loops are nested correctly.

IF-THEN-ELSE

```
if <VAL|EXPR|VAR> <REL> <VAL|EXPR|VAR> [then]... [else...]
```

then is not strictly necessary but must be present if **else** is used. **else** is optional.

After **then** or **else** only one command or expression is allowed. To execute multiple commands, you can use a sub-routine with **gosub/return**.

GOTO

```
goto <VAL>
```

<VAL> must be a numerical value and not an expression.

GOSUB/RETURN

```
gosub <VAL>
<LABEL>
Return
```

<VAL> must be a numerical value and not an expression.

It is possible to nest a maximum of 4 **gosub** routines.

RANDOM

```
random
```

Generates a positive random number between 0 and 32767.

It is not necessary to initialize the random generator (as other BASIC variants require via "randomize").

DELAY

```
delay <VAL|EXPR|VAR>
```

The program is halted for xx milliseconds.

The **delay** value cannot be 0, and the maximum is 32767 (equivalent to approximately 32.8 seconds)

PRINT

```
print <VAL|EXPR|VAR|STR> [;] [,] [<VAL|EXPR|VAR|STR>]
```

Used for status or debug output via terminal output (not available for every LED-Basic Component).

Multiple values or texts can be combined in one row. To do so use a comma or a semicolon. When using a comma a space is inserted, but when using a semicolon no space is inserted. The maximum length of the generated **print** text is 256 characters. Longer texts are truncated.

This command should be used sparingly for small amounts of data, because it reduces execution speed. The **print** function can be disabled globally using configuration parameter P0. If the data is sent too quickly, the terminal may be blocked due to a memory overflow, or the display may be corrupted. If the LED-Basic program hangs or crashes, briefly disconnect the USB connector from the component or Prog_SB.

DATA/READ

```
<LABEL>

data <VAL>[,<VAL>] [,<VAL>]

...

read <VAL>,<VAL|EXPR|VAR>
```

Attention! The syntax used here is not compatible with other basic variants and is especially optimized for LED-Basic.

A maximum of 127 table values can be defined following a label. These may only be numerical values and cannot be changed by the program. You may use several **data** lines in succession, in which case each label may only have 126 values in total. All values must be a maximum of 16-bit (-32768... 32767).

Example:

```
100:
data 10, 20, 30, 40
data 100, 200, 400, 800

200:
data 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80
```

Data lines should be defined at the beginning of the program and at the latest before their first use.

The **read** command accesses individual **data** values.

Example:

```
[1] a = read 100, 0

[2] x = read 200, i
```

[1] Reads the first data value after label 100 (counting always starts with 0)

[2] Reads the data value after label 200 with index i

If the program attempts to read past the last data value, no error is returned; **read** simply returns 0.

There must be no comment following the data line, so if necessary place comments before it.

Hardware Configuration

Various global settings can be set via this configuration line. Please note that not every LED-Basic component supports all the parameters. Unsupported parameters are ignored.

CONFIGURATION LINE

```
### L64 CRGB M100 P1 S3 T0 A3 F25
```

This line must be placed at the beginning of the BASIC code, otherwise the program will use the default values. The line must start with **###.** Several configuration parameters are possible:

Parameter	Description	Default
Lxxx	Number of LEDs connected (MAX_LED) Valid values: 8.. x (x = depending on the component used)	L256, L128, L64
Cxxxy	Colour configuration of the connected LEDs GRB = WS2812, RGB = SK6812, APA102, APA106 For RGBW LEDs, CRGBW must be entered here	CGRB
Mxxx	Master LED brightness in %. Valid values: 5.. 100	M100
Px	Print output on/off. Valid values: 0, 1 (0 = off, 1 = on) Note: Run-time error messages are always displayed and cannot be turned off.	P1
Sx	Turn system LEDs on/off. Valid values: 0.. 3 (0 = off, 1 = output LED on, 2 = function LED on, 3 = all LEDs on)	S3
Tx	Select LED type (for LED-Basic components that have multiple supported LED types) Valid values: see components section	T0
Ax	Bitrate for LEDs of type APA102 (SPI clock.) The higher the value, the lower the bit rate. Valid values: 0.. 7 (the bitrates are listed in the components section)	A3
Fxx	Frame rate for serial LEDs using the LED.show() command. If the frame rate is too low, the LEDs are unclear and flicker. If the frame rate is too high, the LED display may be corrupted and execution speed can be reduced. Valid values: 1.. 100	F25

LED auxiliary commands for LED-Basic components

The number of auxiliary commands and their parameters are dependent on the LED-Basic component in use and can change with LED-Basic versions. Please check and follow the updated instructions.

The number of parameters must be correct. If no parameter is required, empty parentheses must be used, as in C syntax. Ranges of parameter values are not validated. If incorrect values are used, faulty LED displays may occur.

In addition to numerical values, variables or calculated expressions may also be used as parameters.

It is best to learn the function of some commands via trial and error. There are examples in the installation directory, and on the Support home page (link at end).

The LED commands supported by each component are listed in its description.

LED commands for serial RGB LEDs with PWM (e.g. WS2812, APA102, SK6812)

LED.SHOW

```
LED.show()
```

The LEDs are displayed. Due to data transmission speeds, it takes 40ms (25 frames/second) for all to be displayed. Execution is therefore automatically limited to this speed. No LED display is possible without this command.

LED.LRGB

```
LED.lrgb (led, r, g, b)
```

LED at Position **led** is set with the values in **r**, **g** and **b**.

Values: led = [0.. MAX_LED-1], r/g/b = [0.. 255]

LED.LHSV

```
LED.lhsv (led, h, s, v)
```

LED at Position **led** is set with the values in **h** (HUE, colour value), s (SAT, saturation) and **v** (VOL, brightness).

Values: led = [0.. MAX_LED-1], h = [0.. 359], s = [0.. 255], v = [0.. 255]

LED.IRGB

```
LED.irgb (idx, r, g, b)
```

Up to 10 LED colour presets can be stored in index registers. Colour Index **idx** is set based on the values in **r**, **g** and **b**.

Values: idx = [0.. 9], r/g/b = [0.. 255]

LED.IHSV

```
LED.ihsv (idx, h, s, v)
```

Up to 10 LED colour presets can be stored in index registers. Colour Index **idx** is set based on the values in **h**, **s** and **v**.

Values: idx = [0.. 9], h = [0.. 359], s = [0.. 255], v = [0.. 255]

LED.ILED

```
LED.iled(idx, led)
```

LED at position **led** is displayed with the preset colour values in **idx**.

Values: LED = [0.. MAX_LED-1], idx = [0.. 9]

LED.IALL

```
LED.iall(idx)
```

All LEDs are shown with the preset colour values in **idx**.

Values: idx = [0.. 9]

LED.IRANGE

```
LED.irange(idx, beg, end)
```

The LEDs in the range of **beg** to **end** are shown with the preset colour value in **idx**.

Values: idx = [0.. 9], beg = [0.. MAX_LEDS-1], end = [beg.. MAX_LEDS-1]

LED.RAINBOW

```
LED.rainbow(h, s, v, beg, end, inc)
```

A rainbow effect is applied to a set of LEDs defined by (**beg..end**). The start colour is defined by **h**, **s** and **v. inc** specifies the variation or gradient in colour between LEDs. If the value of **h** is continuously changed, the rainbow appears to move. (see example on the Support homepage).

Values: h = [0.. 359], s = [0.. 255], v = [0.. 255], beg = [0.. MAX_LEDS-1], end = [beg.. MAX_LEDS-1], inc=[0.. 100]

LED.COPY

```
LED.copy(from, to)
```

The colour of a single LED in position **from** is copied to the LED in position **to**. The LED at position **from** remains unchanged.

Values: from = [0.. MAX_LEDS-1], to = [0.. MAX_LEDS-1]

LED.REPEAT

```
LED.repeat(beg, end, count)
```

The LED area **beg** to **end** is repeated **count** times.

Values: Beg = [0.. MAX_LEDS-1], end = [Beg.. MAX_LEDS-1], count = [1.. x]

Example:

```
LED.repeat(0, 5, 3)
LED arrangement before call:
0 1 2 3 4 5
LED arrangement after call (repeated 3 times):
0 1 2 3 4 5 0 1 2 3 4 5 0 1 2 3 4 5 0 1 2 3 4 5
```

The upper limit is automatically set at MAX_LEDS.

LED.SHIFT

```
LED.shift(beg, end, to)
```

The LED area **beg** to **end** is moved to after **to**.

Values: beg = [0.. MAX_LEDS-1], end = [beg.. MAX_LEDS-1], to = [0.. MAX_LEDS-1]
Examples:

```
LED.shift(0, 4, 2)
LED arrangement before call:
0 1 2 3 4 x x
LED arrangement after call:
0 1 0 1 2 3 4
```

```
LED.shift(6, 9, 3)
LED arrangement before call:
x x x x x x 6 7 8 9
LED arrangement after call:
x x x 6 7 8 9 7 8 9
```

The LEDs in the vacated area retain their original value and must be set to new values if necessary.

LED.MIRROR

```
LED.mirror(beg, end, to)
```

The LED area **beg** to **end** is mirrored after **to**. Care should be taken to ensure that areas **beg...end** and **to** do not overlap, to avoid undesirable effects. Values: beg = [0.. MAX_LEDS-1], end = [beg.. MAX_LEDS-1], to = [0.. MAX_LEDS-1]

Example:

```
LED.mirror(0, 5, 6)
LED arrangement before call:
0 1 2 3 4 5
LED arrangement after call:
0 1 2 3 4 5 5 4 3 2 1 0
```

LED.BLACKOUT

```
LED.blackout()
```

All LEDs are switched off. LED.show() is not required.

LED commands for RGB LEDs without PWM and single LEDs

LED.SETLED

```
LED.setled(led, colour)
```

LED **led** will be set to colour **colour.**

Values: LED = 0... max (depending on the component), colour = 0... 7

LED.SETALL

```
LED.setall(colour)
```

All LEDs are set to colour **colour.**

Values: colour = 0... 7

Value	**Colour**	
0	Black	⬛
1	Red	🟥
2	Green	🟩
3	Yellow	🟨
4	Blue	🟦
5	Magenta	🟪
6	Cyan	🟦
7	White	⬜

LED commands for multiple LED types

LED.BRIGHT

```
LED.bright(value)
```

Sets brightness of the LEDs or the display. Note: Brightness changes in the upper range are barely noticeable by the human eye but have a disproportionate effect on the LED’s power consumption..

Values: value = [0..x], x depends on the component used and is specified in the list of LED commands.

LED commands for 7-SEGMENT-DISPLAY

LED.CLEAR

```
LED.clear()
```

The display is cleared, all segments off. This command should not be put in a loop or the display will flicker.

LED.PCHAR

```
LED.pchar(pos, char)
```

Puts the predefined character **char** in position **pos** (+ 1). Pos specifies the 7-segment position at which the character is to be output, from left to right (0... 3). The decimal point display remains unaffected.

Values: pos = 0... 3, char = 0... 29

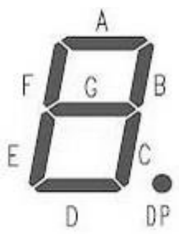
char	characters	char	characters	char	characters
0	0	10	A	20	-
1	1	11	B	21	SPACE
2	2	12	C	22	I
3	3	13	d	23	n
4	4	14	E	24	r
5	5	15	F	25	N
6	6	16	H	26	t
7	7	17	L	27	o
8	8	18	P	28	G
9	9	19	U	29	Y

LED.PRAW

```
LED.praw(pos, raw)
```

Outputs the character in **raw** in position **pos** (+ 1). POS specifies the 7-segment position at which the character is to be output, from left to right (0... 3). Decimal point display remains unaffected.

Values: pos = 0...3, raw = 0...127 (0x7F)



The value in raw indicates the segments to be turned on bitwise.

Bit 0 = A (0x01), Bit 1 = B (0x02), Bit 2 = C (0x04), Bit 3 = D (0x08),
Bit 4 = E (0x10), Bit 5 = F (0x20), Bit 6 = G (0x40)
The decimal point can only be moved using the LED.ADP command.

LED.ADP

```
LED.adp(dp)
```

Sets the decimal point location via bit-code. Bit 0 = Position 0, Bit 1 = Position 1, Bit 2 = Position 2, Bit 3 = Position 3. All other segments display remains untouched.

Values: dp = 0... 255 (0xFF) (Depending on the number of points)

LED.PHEX

```
LED.phex(pos, value, width)
```

Outputs the value in **value** as hexadecimal on the display at position **pos** (+ 1). The value in **width** (+ 1) specifies the width of the output. The hexadecimal display always has 0 prefixes.

For example, a 2-digit hexadecimal number (0x00.. 0xFF) should be displayed on the two right-hand displays: LED.phex (2, value, 1).

Values: (x = depends on the component):
pos = 0...3, value = 0...65536 (0xFFFF), width = 0..3

LED.PDEZ

```
LED.pdez(pos, value, width, lzero)
```

Outputs the value in **value** as a decimal number on the display at position **pos** (+ 1). The value in **width** (+ 1) specifies the width of the output. The value in lzero specifies whether prefix 0s should be suppressed (0 = yes, 1 = no). If a value greater than 9999 is specified, the display is blank.

For example, a 3-digit decimal number (0.. 999) should be displayed right-aligned with suppressed prefix 0s: LED.pdez (1, value, 2, 0).

Values (x = depends on the component):
pos=0...x, value=0...9999, width=0..3, lzero=0..1
pos=0...x, value=0...65535, width=0..4, lzero=0..1

LED Commands for 4-digit 7-SEGMENT-DISPLAY Only

LED.ACHAR

```
LED.achar(ch1, ch2, ch3, ch4)
```

Outputs all 4 characters at the same time. Values of all 4 characters must be specified. The values are identical to those in the table for LED.PCHAR. Decimal point display remains unaffected. This command is available only for 4-digit displays.

Values: ch1, ch2, ch3, ch4 = 0...29

LED.ARAW

```
LED.araw(raw1, raw2, raw3, raw4)
```

Outputs all 4 characters at the same time. Values of all 4 characters must be specified. The values are the same as for LED.PRAW. Decimal point display remains unaffected. This command is available only for 4-digit displays. Values: raw1, raw2, raw3, raw4 = 0... 127 (0x7F)

LED Commands for matrix displays

This area is currently being revised for future components.

IO Auxiliary commands for LED-Basic Components

The number and parameters of additional commands can change with LED-Basic versions. Please be sure to follow the instructions in each update. Please note that the LED-Basic component in use may not allow all functions.

The number of parameters must be correct. If no parameter is required, the parentheses must follow the command, as in C syntax.

In addition to numerical values, variables or calculated expressions can also be used as parameters.

See the list of supported IO commands in the description of the LED-Basic components.

IO.WAITKEY

```
IO.waitkey()
```

Program waits for any key press. The Wait LED flashes, unless disabled via SX in the configuration line.

IO.GETKEY

```
<VAR>=IO.getkey()
```

Whilst the program is running, the status of buttons can be queried. The number of buttons depends on the LED-Basic component in use.

The key press remains stored until a query has been made.

Button	Value
No	0
1	1 (Bit 0)
2	2 (Bit 1)
3	4 (Bit 2)
4	8 (Bit 3)
...	...

Wrong:

```
if IO.getkey() = 1 then goto 1000
if IO.getkey() = 2 then goto 2000
if IO.getkey() = 4 then goto 3000
```

The keyboard state is reset by the first **getkey** command. The second query will never be executed, even if key 2 had been pressed.

Right:

```
k = IO.getkey()
If k = 1 then goto 1000
if k = 2 then goto 2000
If k = 4 then goto 3000
```

Here the keyboard status is stored in a variable and can be queried for several different values.

IO.KEYSTATE

```
<VAR>=IO.keystate()
```

Unlike **getkey,** this function can be used to query whether one or more keys are pressed. Here, the keys are linked bitwise (see IO.getkey). The value returned remains as long as the key is pressed. The value returned when the button is released is NULL. The same value ranges apply as for **getkey**. This function is especially important for input ports if you want to process the signals (high, low). All input ports have integrated pull-up resistors and are debounced by software.

IO.SETPORT

```
IO.setport(port)
```

Sets the output ports to high level (approx. 3.3v). The value in **port** specifies which of the ports are affected**.** The number of ports that can be set is dependent on the LED-Basic component in use.

Port 1 = Bit 0, Port 2 = Bit 1, Port 3 = Bit 2, etc.

IO.CLRPORT

```
IO.clrport(port)
```

Sets the output ports to low level (0v). The value in **port** specifies which of the ports are affected**.**

Port 1 = Bit 0, Port 2 = Bit 1, Port 3 = Bit 2, etc.

IO.GETRTC

```
<VAR>=IO.getrtc(idx)
```

Reads the values of the real-time Clock (RTC) (if available). Depending on the LED-Basic component in use, the index values from 6 onwards may not be present.

Values: idx = [0.. 8]

Idx	Use	Range
0	Seconds	0 - 59
1	Minutes	0 - 59
2	Hours	0 - 23
3	Days	1 - 28/29/30/31
4	Months	1 - 12
5	Years	2000 - 20XX
6	Day of the Year	1 - 365 (366)
7	Weekday	0 - 6 (0 = Monday)
8	Leap year	0 = no, 1 = yes

IO.SETRTC

```
IO.setrtc(idx, val)
```

Sets the real-time clock (if available). If you want to write multiple values starting with the lowest **idx**, it is useful to first read the desired value, change it and then write it again.

Values: idx = [0.. 5] (as with **GETRTC,** 6.. 8 are calculated and cannot be changed)

IO.GETLDR

```
<VAR>=IO.getldr()
```

Reads the Brightness sensor (LDR) (if present). The resultant value is between 0 and 255 depending on the brightness. For technical reasons, the maximum and minimum values are never returned.

IO.GETIR

```
<VAR>=IO.getir()
```

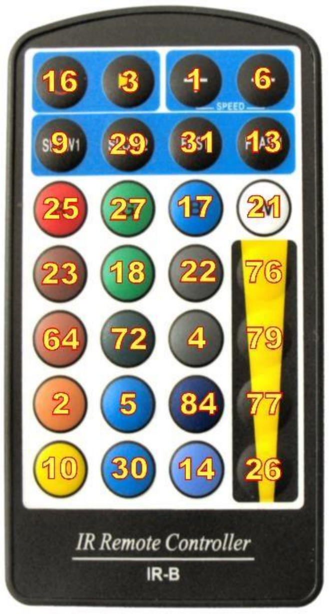
Reads the value of the infrared receiver (if present). 0 means that there is no data. The read value should be written to a variable (as in the GETKEY function). A 16-bit value is returned, but, the actual key value is in the lower 8 bits. The upper 8 bits returns the number of repetitions caused by long-pressing the button.

Example:

```
z = IO.getir()
k = z & 0xFF ' mask key value
n = z / 256 ' n = the number of repetitions
```

The decimal key values for the standard infrared remote control for all components with an infrared receiver can be found in the picture on the right. The values are set by the remote-control manufacturer and cannot be changed. All LED-Basic components with infrared sensors are only able to receive and process signals from this specific remote control.

Please note that colour differences may occur between the LEDs and the keys.



IO.GETTEMP

```
<VAR>=IO.gettemp()
```

Reads the value of the DS18B20 temperature sensor (if present). Result is a value in 0.1 °c resolution. Due to the relatively long duration of the scan, the query should not be made at shorter intervals than 1 second. The sensor's measurement process is only started by reading a value, so the first value read should be discarded.

Examples: 0 = 0 °c, 217 = 21.7 °c,-81 =-8.1 °c

Note: Higher resolutions than 0.1 °c are not possible due to the sensor used. The accuracy as stated by the manufacturer is ± 0.5 °c in the range -10 °c - +85 °c.

IO.XTEMPCNT

```
<VAR>=IO.xtempcnt()
```

For components that support several temperature sensors of type DS18B20 (e.g. the temperature sensor interface), this command displays the number of sensors that have been recognised. A maximum of 8 sensors can be connected. If the sensors are operated via a “parasitic” power supply, fewer than 8 sensors may be recognised (find out by trying!) due to the load on the data line. Please note that the sensors are only recognised during boot-up. You should only connect or disconnect sensors whilst switched off or damage may occur to the hardware and sensors.

IO.XTEMPVAL

```
<VAR>=IO.xtempval(nr, idx)
```

For components that support multiple temperature sensors of type DS18B20, this command can set different values (**idx**) for each sensor (**nr)**.

Values: nr = [0.. x] (x = value from IO.xtempcnt – 1), idx = 0.. 10

idx	Meaning
0	0 = temperature value not valid, < > 0 = temperature value valid
1	Temperature in 0.1 °c resolution (see IO.gettemp)
2	0 = Parasitic power supply, 1 = External power supply
3.. 10	ROM-ID of the sensor

IO.BEEP

```
IO.beep(val)
```

Generates a sound from the loudspeaker (if available).

val	Meaning
0	Mute
1.. 36	Note
From 200	frequency = val Hz

Frequencies below 200 Hz cannot be generated due to system conditions.

The following note values are allowed:

Val	Note	Val	Note	Val	Note
1	C2	13	C3	25	C4
2	C2#	14	C3#	26	C4#
3	D2	15	D3	27	D4
4	D2#	16	D3#	28	D4#
5	E2	17	E3	29	E4
6	F2	18	F3	30	F4
7	F2#	19	F3#	31	F4#
8	G2	20	G3	32	G4
9	G2#	21	G3#	33	G4#
10	A2	22	A3	34	A4
11	A2#	23	A3#	35	A4#
12	H2	24	H3	36	H4

IO.GETENC

```
<VAR>=IO.getenc()
```

Reads the value of the encoder (if present). The value range is set using the following command **IO.SETENC**.

IO.SETENC

```
IO.setenc(pos, max, stop)
```

Sets the parameters for the encoder (if present).

Values: pos = [0.. max], max = [1.. 65535], stop = [0/1]

The encoder provides values between 0 and **max**. The current value, **pos,** must always be set to < = **max**. If **stop** = 1, the maximum value will remain even if the encoder is turned further. If **stop** = 0 the counter will start again at 0 if the maximum value is exceeded and is **max** again if reduced below 0.

IO.GETPOTI

```
<var>=IO.getpoti(idx)
```

Reads the value of a potentiometer, or its converted value (if present). Which values, and how many, depends on the component.

Values: idx = [0.. x], x depends on the component used

IO.GETADC

```
<var>=IO.getadc(idx)
```

Reads an analogue value. The values that can be read depend on the component.

Values: idx = [0.. x], x depends on the component used

IO.EEREAD

```
<VAR>=IO.eeread(adr)
```

Reads the contents of the EEPROM at address **adr** (if available). Returns a 16-bit value. On manufacture, all memory locations of the EEPROM are 0xFFFF (-1). Always returns 0 if **adr** is an invalid address.

Values: adr = [0.. x], x depends on the component used

IO.EEWRITE

```
IO.eewrite(adr, data)
```

Writes the 16-bit value in **data** to the EEPROM with address **adr** (if available). Please note that the write operation may take several milliseconds and thus delay the program's execution. To prolong the life of the EEPROM, write operations should be avoided where possible, and values saved only if they have changed

Values: ADR = [0.. x], x depends on the component used and is specified in the list of IO commands. data = [-32768.. 32767]

IO.SYS

```
<VAR>=IO.sys(par1, par2)
```

Universal command to set or read system parameters (if present). Both parameters must always be passed, even if they are not needed.

The common values for PAR1 are listed here. If a component has other values, they are described there separately.

[COM] Communication

These commands can be used to send single or multiple characters to the component. It can use LED-basic commands to evaluate the data and possibly return results via the Print command.

<var> = IO.sys(0, 0)

Reads the number of characters received through the virtual COM port.

0 = No characters received

1..64 = Number of characters in the receive memory.

IO.sys(0, 1)

Sets the number of characters received to zero.

<var> = IO.sys(n, 0)

Reads the receive memory at position n.

Valid values for n: 1..x (x = number of characters received, Max. 64).

Using the terminal (from version 15.1.14), the ASCII values of key presses is transmitted. One character is written to the receive memory.

Example: Key A is pressed. IO.sys(0.0) returns 1 = 1 value in memory. IO.sys(1, 0) returns the value 65 (0x41) = ASCII code for "A". After that, the receive counter should be set to zero using IO.sys(0, 1).

[CALIB] Calibration

IO.sys(99, c)

Calibration of the real-time clock. If the integrated real-time clock is to be used, this can be calibrated here.

Valid values for c: -510 to 510

A negative value makes the clock slower, a positive value makes the clock run faster.

Change the values in 10s, 20s or 50s and then synchronize the clock with the PC. Since the changes have only a slight effect on the speed of the clock, it can take several hours or even days to detect a change in speed.

[DFP] DFPlayer Mini

The DFPlayer Mini plays MP3 files from a micro SD card.

<val>=IO.sys(1000, 0)	Read status. **val**: 0 = busy
IO.sys(1000, 1)	Reset status
IO.sys(1000 + cmd, par)	**cmd:** matches CMD 0x01 to 0x4D of DFPlayer Mini **par:** 16 Bit Parameter (0 if not needed)
IO.sys(1100, 0)	Delete queue
IO.sys(1100, 1)	Play queue
IO.sys(1101, par)	Add file to queue (max. 16 entries)

Examples:

```
IO.sys(1012, 0)              Reset DF-Player (use twice)
IO.sys(1006, 20)             set volume at 20 (valid values 0.. 30)
```

After these commands, the busy status must be queried.

```
IO.sys(1015, (2 * 256) + 45)     Play MP3 file [045. mp3] in directory [02]
```

Busy status can be queried to wait for the end of the file.
Playback is terminated immediately when a new command is sent.

```
IO.sys(1100, 0)                  Delete queue
IO.sys(1101, (2 * 256) + 1)      Add MP3 file [001.mp3] in directory [02]
IO.sys(1101, (2 * 256) + 3)      Add MP3 file [003.mp3] in directory [02]
IO.sys(1101, (2 * 256) + 24)     Add MP3 file [024.mp3] in directory [02]
IO.sys(1101, (2 * 256) + 31)     Add MP3 file [031.mp3] in directory [02]
IO.sys(1101, (1 * 256) + 1)      Add MP3 file [001.mp3] in directory [01]
IO.sys(1100, 1)                  Play queue
```

Read Busy Status Example:

```
1000:
        s = IO.sys(1000, 0)          Read status
        if s = 0 goto 1000           If busy = 0, repeat read status
        IO.sys(1000, 1)              Reset Status
        return                       and return
```

Link to the DF-player Guide (English):
https://www.dfrobot.com/wiki/index.php/DFPlayer_Mini_SKU:DFR0299

Runtime error messages

Runtime errors are displayed on through the terminal (if supported by the LED-Basic component).

```
?ERROR xx IN LINE zzzzz
```

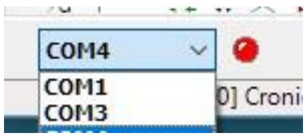
Error	Indication
11	Unknown token (should not occur)
12	Wrong address (should not occur)
13	Too many nested GOSUB commands
14	RETURN without GOSUB
15	Value cannot be 0
16	Too many nested FOR-NEXT loops
17	Incorrect values at TO/DOWNTO
18	Next variable is invalid
19	Wrong value in LED command
20	Wrong value in IO command

For LED-Basic components without terminal (e.g. LED-button 12), the error code is indicated by the number of red flashing LEDs (+ 9) (6 flashing LEDs means error 15).

LED-Basic Components

Driver installation

Each LED-Basic component with a USB connection, and the programming adapter "Prog-SB", are addressed via a virtual COM-port. On Windows 7 and 8. x, you must install the driver file "LedBasic. inf". If you are still using Windows 8. x, you may need to "Disable driver Signature enforcement". To find how to do this is easy via a Google search.
There is no need to install a driver on Windows 10.
Each component appears as a "serial USB device (COMX)". The best way to test which COM port is correct for your component is to click on the list of COM ports in the LED-Basic Editor before and after you have connected the component. The COM port that appears on insertion should be correct.



Different Components

Various components are available for use with LED-Basic. These can be distinguished from one another by the ability to control different types and sets of LEDs, connections for sensors and switches, and internal peripherals such as Real-Time Clock or IO ports.

All the LED-Basic Components use the same basic command set. Only the LED and IO auxiliary commands are different. Please refer to the following descriptions to find out which commands your LED-Basic component understands. LED-Basic is universal and needn’t be used solely with LEDs. LED-Basic can also be used in components that only query sensors and trigger switching operations.

There are several LED-Basic Components that feature an USB port. This connection is used both for programming and for terminal output. If the component does not have a USB port, the USB programming adapter "Prog-SB" is needed. This connects the PC to the 6-pin programming interface of the LED-Basic component.

List of components supported by LED-Basic

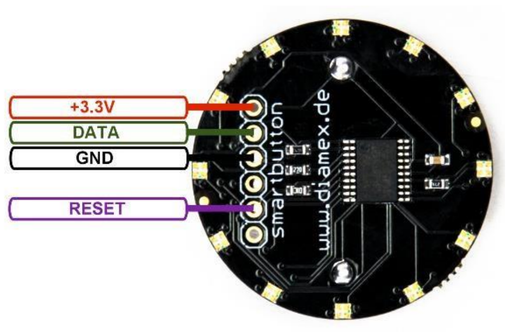
The following is a description of the LED-Basic Components supported by the LED-Basic Editor. Please note that there are common components across many of the controllers. The description lists the connection arrangement for each LED-Basic component, a list of LED and IO commands supported by the controller, and the unit’s configuration line parameters.

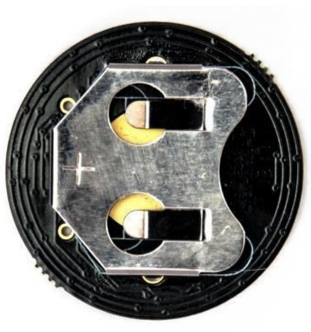
NOTES: All input/output pins have a maximum level of 3.3 volts. To prevent hardware damage, if higher voltages are involved you should use resistors or level converters. Input pins have integrated pull-up resistors. Push buttons should be simply connected from the input pin to GND. Output pins are at low level (0 volts) in the idle state. At high level, you can only drive a few milliamps, e.g. an LED can be connected directly via a suitable resistor. If you need a higher current (e.g. to control a relay), it is necessary to use transistors or IC drivers.

LED outputs for serial RGB LEDs (WS2812, APA102) always have a level of 5 volts.

All un-labelled pins are unused or are used solely for programming and testing during manufacture.

LED-Badge (12 LEDs)





BADGE = badge, sticker, brooch

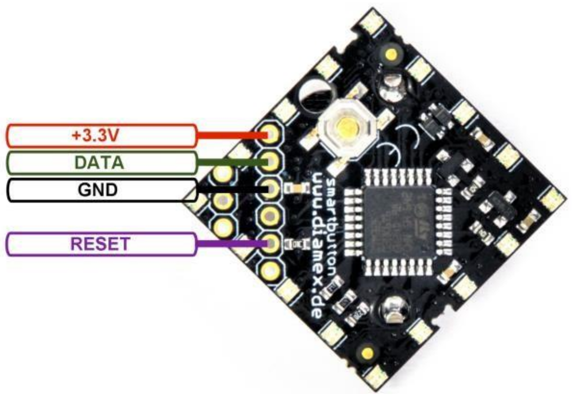
12 RGB LEDs without PWM. Can be switched on and off by inserting and removing the battery.

Program using a Prog-SB (+ 3.3 V, DATA, GND, RESET).

System code: 3110

Configuration line	LED commands	IO commands
NONE	SETLED, SETALL	NONE

LED-Badge (16 LEDs)



Different designs are available. The pin assignment for the programming adapter is identical for all designs.

BADGE = badge, sticker, brooch

16 RGB LEDs without PWM. Switch on by pressing the button, turn off by pressing the button for more than 2 seconds. It is possible to query the button state using a BASIC command. Terminal display and error output is shown via Prog-SB.

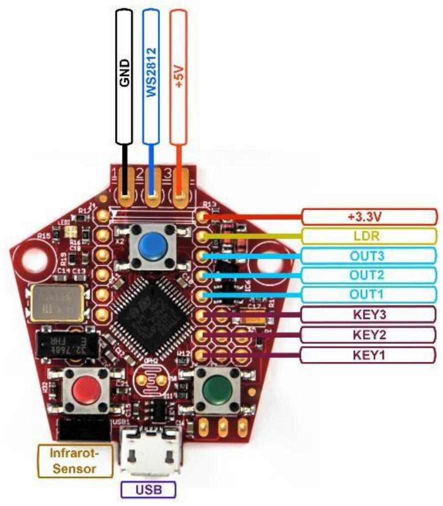
Program via Prog-SB (+ 3.3 V, DATA, GND, RESET, TERMINAL).

System code: 3120

Configuration line	LED commands	IO commands
P...	SETLED, SETALL	WAITKEY, GETKEY CLRPORT

The LED button is switched off by IO.CLRPORT(0). This can be used, for example, in BASIC programming to switch off the component and thus save energy after a set number of loops.

Basic-Pentagon-Board



The main unit for LED-Basic. Controls a maximum of 256 LEDs with integrated PWM (WS2812, SK6812 and compatible). Also suitable for RGBW LEDs. Optionally controllable via Infra-red remote-control. Terminal display and error output via USB. 3 buttons or input pins, 3 programmable output pins.

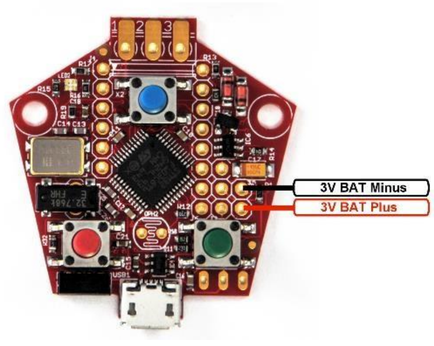
Can be powered via the USB or LED connections.

An LDR (photosensitive resistor) can be soldered directly onto the PCB (OPH2) or connected to the LDR and +3.3v pins.

Program via USB connection.

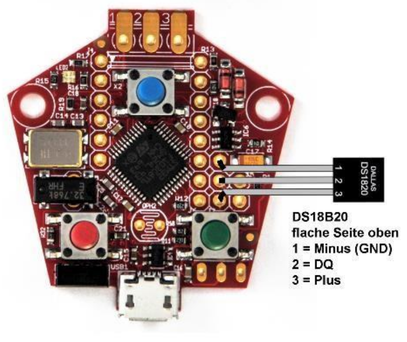
System code: 3130

Configuration line	LED commands	IO commands
L... C... M... P... S... Q... LXXX: Maximum = 256	All commands for RGB LEDs with PWM	WAITKEY, GETKEY, KEYSTATE GETRTC, SETRTC GETLDR GETIR SETPORT, CLRPORT GETTEMP

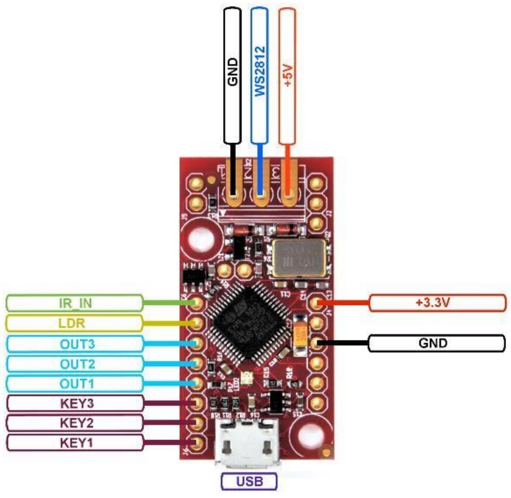


If the real-time clock is to be used, a 3 volt battery (e.g. CR2032 or 2 x AAA cells) can be connected to the designated pins.

Connection of a DS18B20 temperature sensor to the Pentagon board.



Basic-Budget-Board



The main unit for LED-Basic in a lightweight design (e.g. suitable for model aeroplanes). Software for the basic Pentagon board is also compatible with this board. Controls a maximum of 256 LEDs with integrated PWM (WS2812, SK6812 and compatible). Also suitable for RGBW LEDs. Terminal display and error output via USB. 3 buttons or input pins, 3 programmable output pins.

Can be powered via the USB or LED connections.

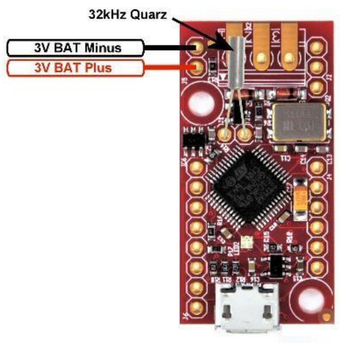
An LDR (photosensitive resistor) can be connected to the LDR and +3, 3v pins.

An infrared receiver can be connected to the IR_IN pin(see wiring diagram for Basic-Booster).

Program via USB connection.

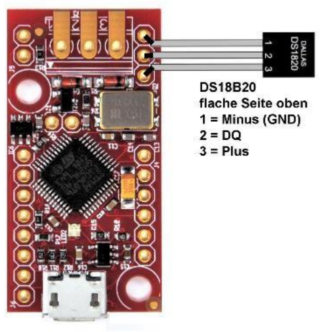
System code: 3130

Configuration line	LED commands	IO commands
L... C... M... P... S... Q... LXXX: Maximum = 256	All commands for RGB LEDs with PWM	WAITKEY, GETKEY, KEYSTATE GETRTC, SETRTC GETLDR GETIR SETPORT, CLRPORT GETTEMP

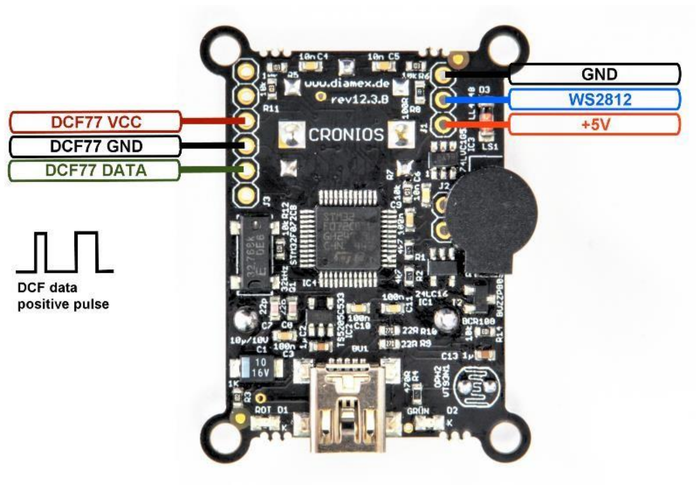


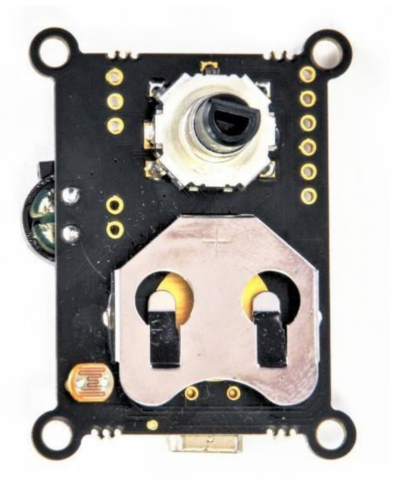
If the real-time clock is to be used, a 3 volt support battery (e.g. CR2032 or 2 x AAA cells) can be connected to the designated pins. In addition, a 32.768 kHz Quartz crystal must be soldered onto the board.

Connecting a DS18B20 temperature sensor to the Budget-Board.



Cronios 1





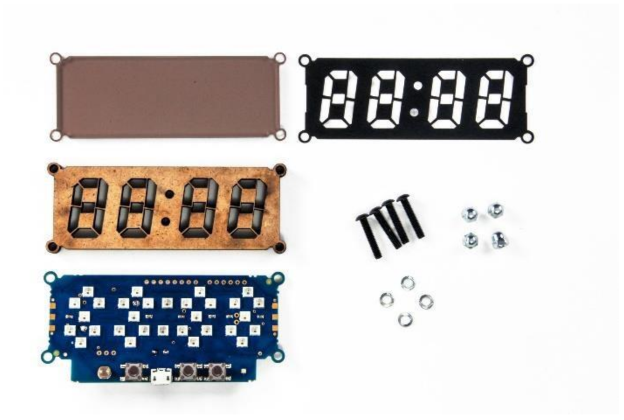
The Cronios 1 Component works with the LED-Basic software. Controls a maximum of 256 LEDs with integrated PWM (WS2812, SK6812 and compatible). Also suitable for RGBW LEDs. Terminal display and error output via USB. Integrated real-time clock with support battery. The board includes a rotary encoder (spinner) with button functionality, LDR and loudspeaker. EEPROM with 1024 memory locations. Can be powered via the USB or LED connections. As of LED-Basic-version 15.1.12, a DCF77 module can be connected for time synchronisation.

Program via USB connection.

System code: 3140

Configuration line	LED commands	IO commands
L... C... M... P... S... Q... LXXX: Maximum = 256	All commands for RGB LEDs with PWM	WAITKEY, GETKEY, KEYSTATE GETRTC, SETRTC GETLDR BEEP GETENC, SETENC EEREAD, EEWRITE SYS [COM, CALIB]

Cronios-Segmenta



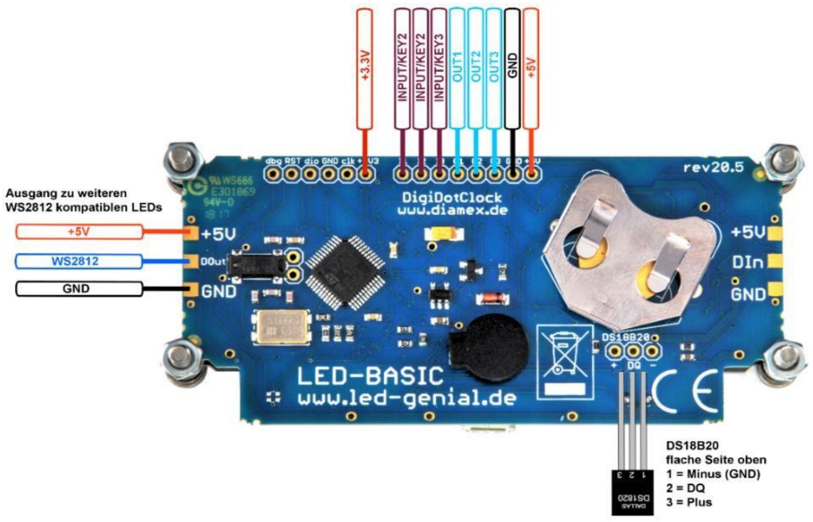
Kit for a 4-digit 7-segment clock with coloured numerals consisting of SK6812 mini LEDs. Terminal display and error output via USB. The board includes an integrated real-time clock with support battery. 3 keys or input pins, 3 output pins for switching signals, LDR and loudspeakers. Has suitable connection for optional temperature sensor DS18B20. Power supply via USB port. In addition to the 30 hardwired LEDs, the output (Dout) can be used to control up to 256 LEDs.

Program via USB connection.

System code: 3150

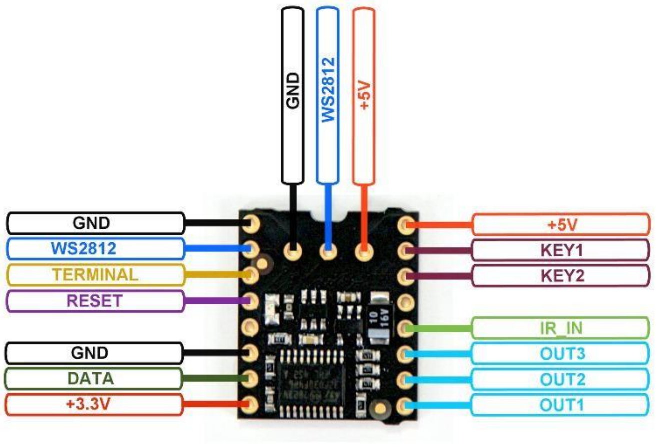
Configuration line	LED commands	IO commands
L... M... P... S... Q... LXXX: Maximum = 256	All commands for RGB LEDs with PWM	WAITKEY, GETKEY, KEYSTATE GETRTC, SETRTC GETLDR SETPORT, CLRPORT GETTEMP BEEP EEREAD, EEWRITE

Note: Because the board uses a high current, the display may not work if connected to a low-power USB port. If so, try another USB port or use a powered USB hub.



Basic-Booster

Mini-interface for LED-Basic. Controls a maximum of 64 LEDs with integrated PWM (WS2812, SK6812 and compatible). Also for RGBW LEDs. Terminal output and error output via Prog-SB. Functions can be triggered via push buttons connected to two input pins, 3 output pins (OUT1, 2.3) are available for switching operations. Optionally, an infrared receiver can be connected to IR_IN, the required components are available in the set that includes the remote control. Powered via one of the + 5v connections.



Program via Prog-SB (+ 3.3 V, DATA, GND, RESET, TERMINAL).

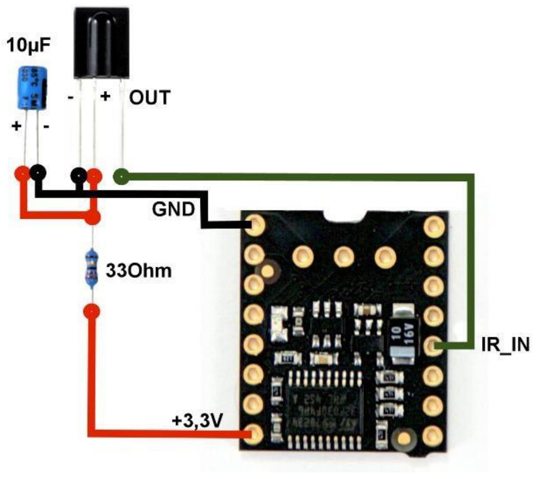
System code: 3160

Configuration line	LED commands	IO commands
L... C... M... P... S... Q... LXXX: Maximum = 64	All commands for RGB LEDs with PWM	WAITKEY, GETKEY, KEYSTATE SETPORT, CLRPORT GETIR

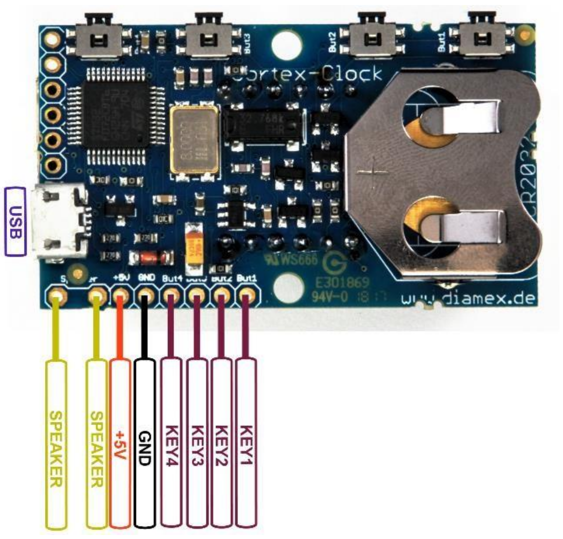
Connecting an infrared receiver to the Basic-Booster

In addition to the infrared remote control, the infra-red capability of this board requires the appropriate infrared receiver TSOP31436, a 33 ohm resistor and a 10µF electrolytic capacitor. The resistor and capacitor are used to suppress interference from the supply voltage, and thus to ensure clear infrared signal reception. Please do not mistake the polarity of the capacitor. The shorter leg (black mark on the housing) should be connected to GND. Connect the parts to the basic booster according the following diagram

Tip: The Basic-Booster PCB can be placed on a breadboard with matching pin strips. The components for the infrared receiver can then be wired easily without soldering.



Cortex-Clock



4-digit 7-segment Clock module. Terminal display and error output via USB. Integrated real-time clock with support battery. 4 push buttons can be used as input pins, via solder pads. Board includes a connector for a miniature speaker. Power supply via USB connection, or via the + 5v and GND pads.

This module is not limited just to displaying the time. It can be programmed for other applications, such as short-term alarm clocks or egg timer, stopwatch, pedometer and more.

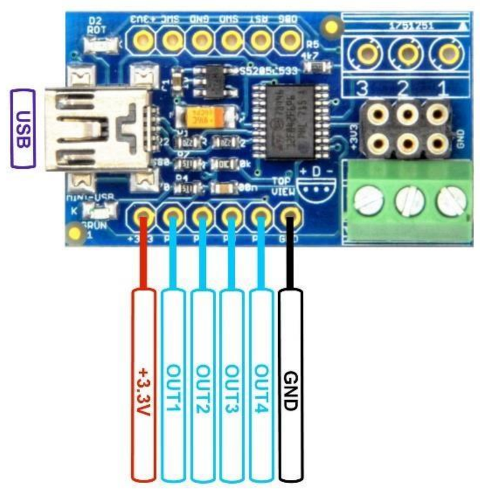
Program via USB connection.

System code: 3170

Configuration line	LED commands	IO commands
P...	All commands for 7-SEGMENT-DISPLAY, BRIGHT [0..15]	WAITKEY, GETKEY, KEYSTATE GETRTC, SETRTC BEEP EEREAD, EEWRITE [0..15]

Temperature-Sensor Interface

Interface for a maximum of 8 temperature sensors of type DS18B20 by Dallas. Switching operations are triggered using 4 output ports. Terminal display and error output via USB. Power supplied via the USB port. Plug-in board-compatible with matching pin strips.



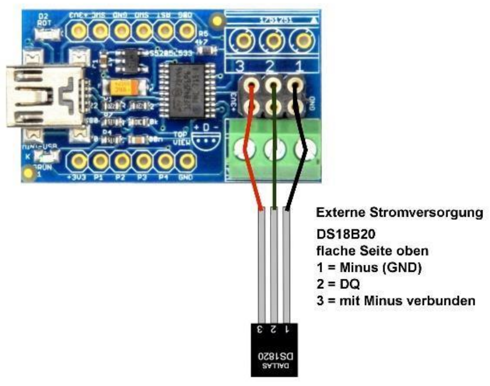
Program via USB connection.

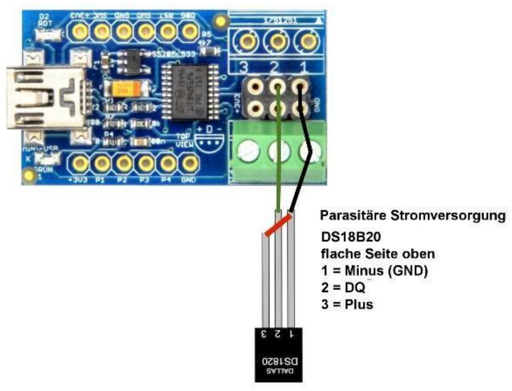
System code: 3180

Configuration line	LED commands	IO commands
P... S...	None	XTEMPCNT, XTEMPVAL SETPORT, CLRPORT

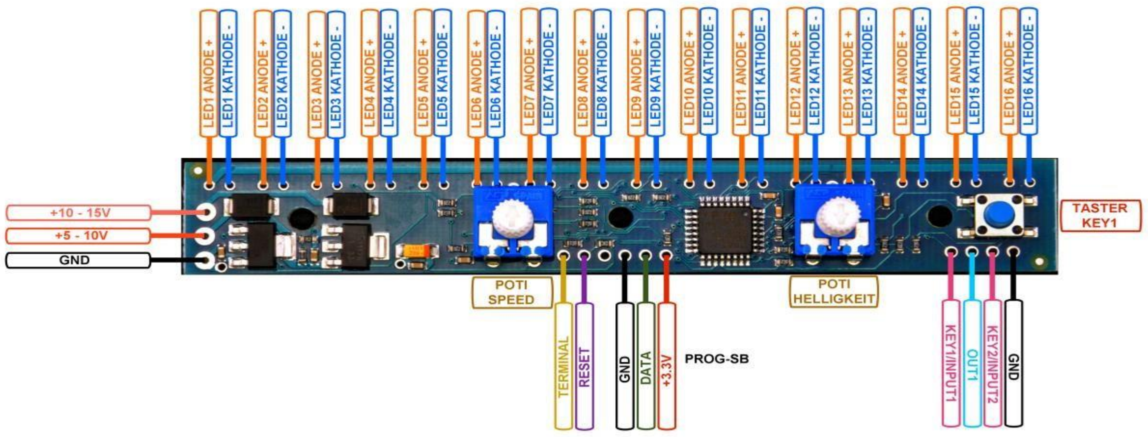
Notes on Use:

Do not connect or remove any sensors whilst unit is connected to power.
Do not clean the sensors, this damages them.
If connected using a “parasitic” power supply using only 2 lines, it may not be possible to use the maximum number of sensors (use trial and error).
All sensors are queried sequentially. Each one takes approximately 1 second, so it can take up to 8 seconds for the temperature to be updated (with 8 sensors connected).
The order of sensors can vary when sensors are exchanged. All sensor connections on the board are connected in parallel, so the order of the sensors is independent of the pins to which they are connected When reading a sensor, the green LED flashes briefly - this can be switched off by specifying S0 in the configuration line.
The red LED indicates that a run-time error has occurred. The exact error message is displayed via the terminal.





Running Light with 16 LEDs



Control for 16 monochrome LEDs. 2 potentiometers for Speed and Brightness control (or programmable for other functions). Includes 1 Push button, and an additional input port is available. Switching operations can be triggered via an output port. Terminal display and error output via Prog-SB. The power supply (5-10v/10-15v) is connected via solder pads.

Program via Prog-SB (+ 3.3V, DATA, GND, RESET, TERMINAL).

System code: 3190

Configuration line	LED commands	IO commands
P...	SETLED, SETALL BRIGHT	GETKEY, WAITKEY, KEYSTATE SETPORT, CLRPORT, GETPOTI EEREAD, EEWRITE [0..15]

LED.setled (pos, val)

val: 0 = switches off LED at position **pos**, 1 = switches on LED at position **pos**
pos: 0.. 15

LED.setall(val)

val: 0 = all LEDs off, 1 = all LEDs on

LED.bright(val)

Val: 0.. 255 = all LEDs set to brightness **val**, > 255 = brightness controlled by potentiometer (default)

IO.getpoti(idx)

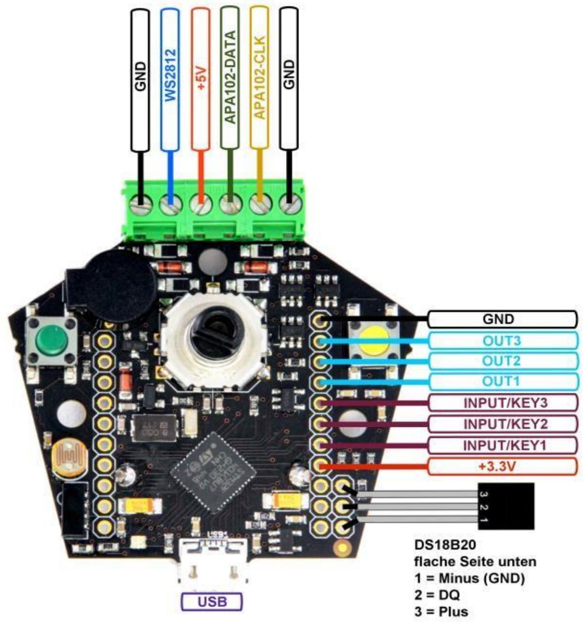
Reads the values of the potentiometers, or their converted values from a table.

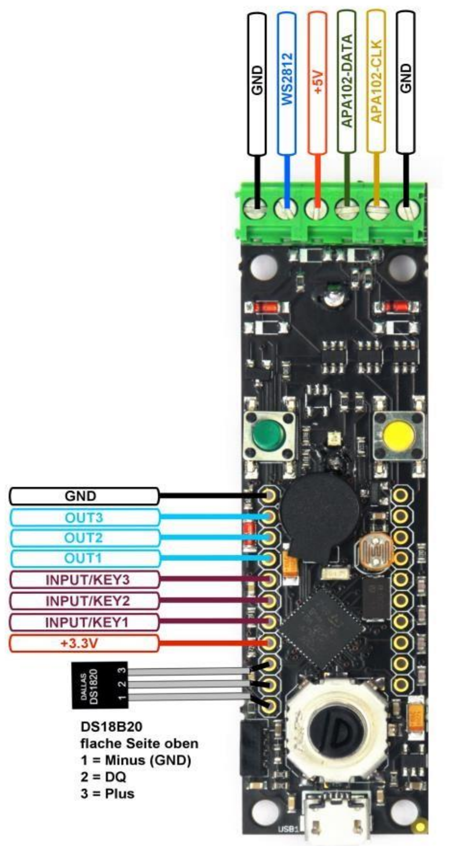
Valid IDX values:
0 = Speed-read potentiometer value directly (0.. 255)
1 = Read converted speed-potentiometer value from table (0.. 15)
2 = Brightness potentiometer value (0.. 255)- read directly
3 = converted brightness potentiometer value from logarithmic table (0.. 255)

Response in case of an invalid **idx** value is null.

The potentiometer value is read by an analogue-digital converter in the unit's microcontroller. To determine a consistent result, The unit takes several consecutive values and compares them in order to get a consistent result. Therefore, when the **getpot** function is run for the first time, the result may be invalid (value = 0).

All-in-One Power-M4-Board





A powerful circuit with many functions, based upon the Cortex-M4. For a maximum of 1024 LEDs with integrated PWM (WS2812, SK6812 and compatible, as well as APA102 and compatible). Also suitable for RGBW LEDs. Optionally controllable with infrared remote control. Terminal display and error output via USB. 2 buttons, 3 input pins, 3 programmable output pins. Rotary encoder with button, loudspeaker and LDR (photo resistor), Real Time Clock (RTC) with battery. Connections for optional temperature sensor (DS18B20) and beeper. EEPROM with 1024 memory locations. Power supply via USB or LED connection.

Program via USB connection.

System code: 3210

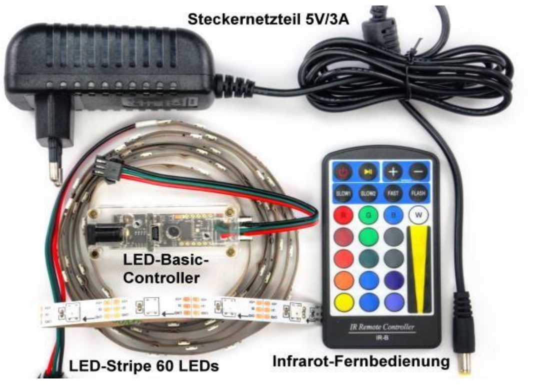
Configuration line	LED commands	IO commands
L... C... M... P... S... T... A... Q... LXXX: Maximum = 1024 TX: 0 = WS2812, 1 = APA102	All commands for RGB LEDs with PWM	WAITKEY, GETKEY, KEYSTATE GETRTC, SETRTC GETLDR GETIR SETENC, GETENC SETPORT, CLRPORT GETTEMP BEEP EEREAD, EEWRITE [0..1023]

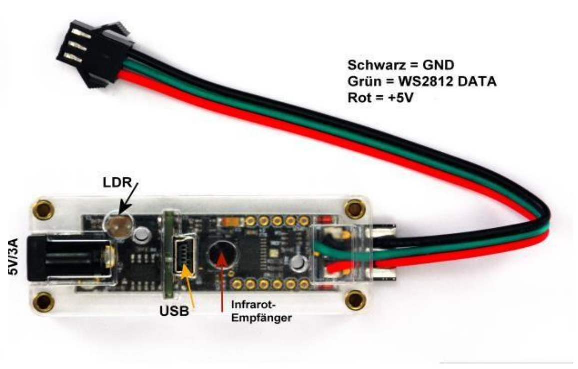
In the configuration line, optional value **Ax** sets the bit rate for the control of APA102 LEDs. The higher the value, the lower the bit rate. For longer LED connection cables, if interference occurs it may be necessary to set a lower bit rate. This parameter has no meaning when using WS2812 LEDs.

A0 = 42 Mbit	A1 = 21 Mbit	A2 = 10.5 Mbit	A3 = 5.25 Mbit
A4 = 2.6 Mbit (default)	A5 = 1.3 Mbit	A6 = 656 kbit	A7 = 328 kbit

The LEDs may flicker when using bitrates above 10 Mbit.

LED-Box





Special circuit for controlling WS2812 strips via infrared remote control. The LED box includes the LED-Basic controller, an LED strip with 60 LEDs, infrared remote control and power supply. Ideal for apartment effect lighting. Using the LDR, the strip brightness can be adjusted dependent on ambient light. EEPROM with 8 memory locations. Terminal display and error output via USB. Power supply via 5v/3a adapter.

Program via USB connection.

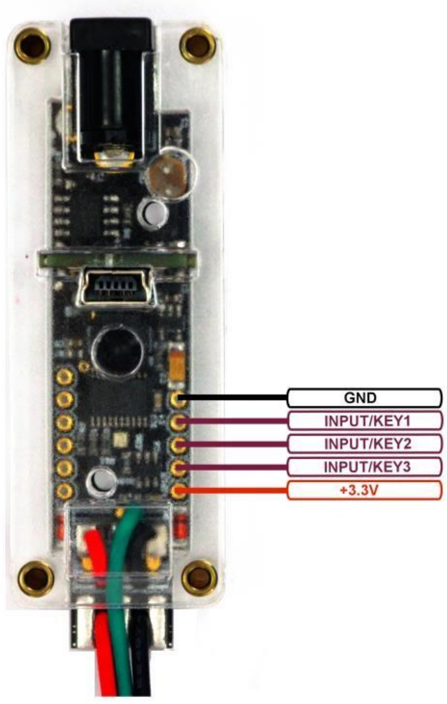
System code: 3220

Configuration line	LED commands	IO commands
L... M... P... S... Q... LXXX: Maximum = 64	All commands for RGB LEDs with PWM	WAITKEY, GETKEY, KEYSTATE SETPORT, CLRPORT GETIR GETLDR EEREAD, EEWRITE [0..7]

The circuit has a quiescent current even when off. Therefore, the LED power supply can be controlled by **IO.setport(1)** to turn on and command **IO.clrport(1)** to turn off.

LEDs may flicker or switch on after their power is connected. Therefore, a command to delete all LEDs should be sent immediately after switching on (e.g. **LED.blackout()**)

Three input pins (input/key) can be queried in LED-Basic, for example, to trigger certain LED sequences via an external signal.



LED-Matrix 10x10

AVAILABLE SOON

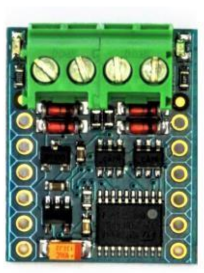
Status: Due to technical difficulties in production, the availability of this component is delayed.

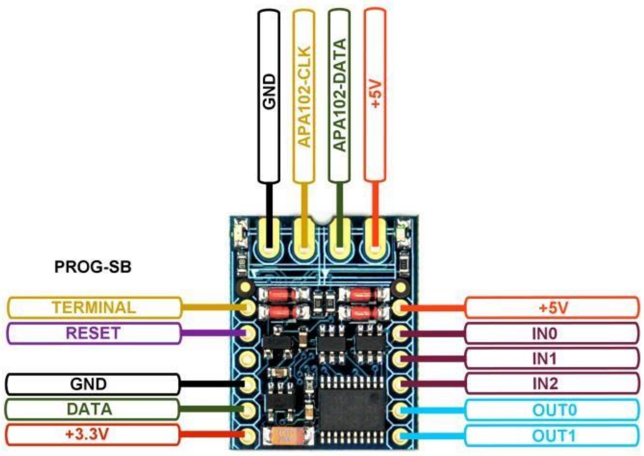
PICTURES TO FOLLOW

Mini 10x10 matrix with RGB LEDs. Real time clock with battery, 2 buttons. Terminal display and error output via USB. Power supply and program via USB connection.
System code: 3200

Configuration line	LED commands	IO commands
P...	MATRIX commands (still in development)	WAITKEY, GETKEY, KEYSTATE GETRTC, SETRTC

APA Booster





Special mini-interface for LED-Basic. For a maximum of 256 RGB LEDs with integrated PWM of type APA102 (DATA- and CLOCK-line). Terminal display and error output via Prog-SB. Functions can be triggered via 3 input pins (IN0..3). 2 output pins (OUT0..1) are available for switching operations. Power supplied via the + 5v LED connector.

This component is not suitable for WS2812/SK6812 LEDs!

Program via Prog-SB (+ 3.3 V, DATA, GND, RESET, TERMINAL).

System code: 3230

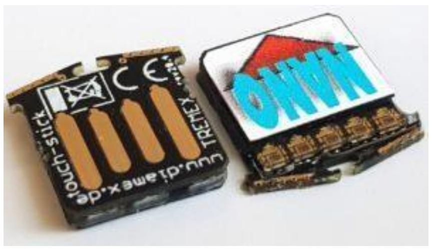
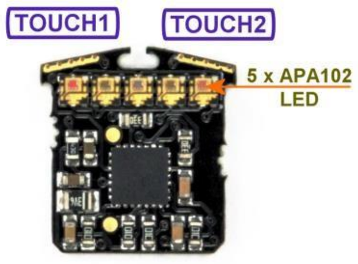
Configuration line	LED commands	IO commands
L... C... M... P... S... A... Q... LXXX: Maximum = 256	All commands for RGB LEDs with PWM	WAITKEY, GETKEY, KEYSTATE SETPORT, CLRPORT

In the configuration line, optional value **Ax** sets the bit rate for the control of APA102 LEDs. The higher the value, the lower the bit rate. For longer LED connection cables, if interface occurs it may be necessary to set a lower bit rate.

A0 = 14 Mbit	A1 = 12 Mbit	A2 = 65 Mbit	A3 = 3 Mbit
A4 = 1.5 Mbit (default)	A5 = 750 kbit	A6 = 375 kbit	A7 = 187.5 kbit

The LEDs may flicker when using bitrates above 10 Mbit.

Touch-Stick (Nano)

Mini USB stick with LED-Basic programming capability. 5 Mini-APA102 LEDs, 2 touch sensors. EEPROM with 16 memory locations. Output via terminal. Display and error output via USB. USB power supply.

Program via USB connection.

System code: 3260

Configuration line	LED commands	IO commands
M... P... CRGB	All commands for RGB LEDs with PWM	WAITKEY, GETKEY, KEYSTATE EEREAD, EEWRITE [0..12] SYS [COM]

Touch-Lamp

AVAILABLE SOON

Status: Development completed, waiting for production material.

PICTURES TO FOLLOW

Effect lamp with sensor buttons and remote control. Colour and brightness of 8 Integrated WS2812 LEDs can be changed and can illuminate a Plexiglass display. EEPROM with 8 memory locations. Terminal display and error output via USB. USB power supply.

Program via USB connection.

System code: 3270

Configuration line	LED commands	IO commands
M... P...	All commands for RGB LEDs with PWM	WAITKEY, GETKEY, KEYSTATE SETPORT, CLRPORT GETIR EEREAD, EEWRITE [0..7]

As the LEDs pass a quiescent current in the off state, the LED power supply can be turned on using the command **IO.setport(1)** and off by command **IO.clrport(1)**.

VFD-Clock

AVAILABLE SOON

Status: Still in development.

PICTURES TO FOLLOW

8-digit clock with 7-segment IV-6 VFD tubes. Terminal display and error output via USB. Integrated real-time clock with support battery. DCF-77 connection with socket. 4 buttons, LDR, beeper, DS18B20 temperature sensor. Power supply via USB, current consumption approx. 450mA.

Program via USB connection.

System code: 3280

Configuration line	LED commands	IO commands
P... M... F...	One command for 7-SEGMENT DISPLAY, BRIGHT [0..256]* All commands for RGB LEDs with PWM	WAITKEY, GETKEY, KEYSTATE GETRTC, SETRTC BEEP GETLDR GETTEMP EEREAD, EEWRITE [0..999]* SYS [COM, CALIB, DFP]

*LED.bright(val)
val: 0.. 256 = set VFD display brightness at **val**.
At 0, tube heating and high voltage are switched off.

*IO.eeread(ADR), IO.eewrite(adr, data)
adr: [0..99]

The EEPROM addresses 0.. 7 are reserved for calibration of the brightness of the VFD tubes, which can have considerable differences due to production. Please do not use these addresses for user programs. There is a brightness correction value for each VFD tube. 0 = no correction, positive value = brighter (maximum 4000), negative value = darker (minimum -1900). If the positive values are too high, the display may flicker. After changing the values in the EEPROM, a restart must be performed to accept the values (press F12). Of course, individual segments cannot be adjusted; variation between such can only be solved by exchanging the tube.

*IO.eeread(adr), IO.eewrite(adr, data)
adr: [1000..1007]

These "virtual addresses" are intended for temporary brightness changes of the VFD tubes. Writing to one of these addresses causes an immediate change in the brightness of the associated tube. However, these addresses are lost when you remove the power supply. For permanent storage, they must be copied to the EEPROM addresses 0..7.

In the configuration line, the number, type and colour of the “LEDs” are ignored. The large, central LEDs are addressed by numbers 0 to 7, starting from the left. The small LEDs are addressed by numbers 8 to 39, starting from the left (4 under each tube).

It is normal for the board to get hot, due to the heating and high voltage.

The high voltage generator, and the PWM of the LEDs, can interfere with reception of DCF77 signals. Tip: Switch off the VFD tubes and the LEDs for a few minutes during the night so that the DCF77 signal is received without interference and the clock can synchronize.

Caution! The high voltage in the circuit can be up to 60 volts. Please only touch the components in the circuit when power is disconnected.

6-O-Clock

AVAILABLE SOON

Status: Development completed, waiting for production.

PICTURES TO FOLLOW

6-digit clock with 7-segment display. Terminal display and error output via USB. Integrated real-time clock. 2 push buttons. Power supply via USB connection or 3v battery (only for short-term operation via button press).

Program via USB connection.

System code: 3290

Configuration line	LED commands	IO commands
P...	Some commands for 7-segment display, BRIGHT [0..9]*	WAITKEY, GETKEY, KEYSTATE GETRTC, SETRTC EEREAD, EEWRITE [0..1023] GETADC SYS [COM, CALIB]

*LED.bright(val)
val: 0..9 = Set display brightness. If powered using battery, set no more than 5 in order to not overtax the battery.

There are colons at position 1 and 3 and a decimal point at position 4.

IO.getadc(0)
Battery voltage. Returns voltage value in 0.01 volt resolution, so 290 corresponds to 2.90v. For example, a warning message can be shown on the display when the battery is almost exhausted. Note that this value varies depending on how many segments are lit and may need to be read multiple times.

<VAR> = IO.sys(100, 0)
Query whether powered by battery (var = 0) or by USB (var = 1)

IO.sys(101, 0)
Puts the clock into sleep mode. The microcontroller is placed in advanced power save mode (current consumption approx. 3µA), the display is switched off, the program execution is terminated. Only the quartz oscillator is still in operation. Waking from this state can only be achieved by pressing Button 1.

LED-Tube-Clock

AVAILABLE SOON

Status: Development completed, waiting for production.

PICTURES TO FOLLOW

8-digit clock with 7-segment LED displays in VFD tube design. Terminal display and error output via USB. Integrated real-time clock with support battery. DCF-77 connection with Jack. 4 buttons, beeper. Power supply via USB connection, current consumption approx. 30mA.

Program via USB connection.

System code: 3300

Configuration line	LED commands	IO commands
P...	Commands for 7-SEGMENT-DISPLAY, BRIGHT	WAITKEY, GETKEY, KEYSTATE GETRTC, SETRTC BEEP EEREAD, EEWRITE [0..15] SYS [COM, CALIB]

LED-Nixie-4

AVAILABLE SOON

Status: In development.

PICTURES TO FOLLOW

4-digit LED Nixie clock. Terminal display and error output via USB. Integrated real-time clock with support battery. DCF-77 connection with Jack. Rotary Encoder, 1 push button, beeper, LDR and connector for DS18B20 temperature sensor. Power supply via USB port.

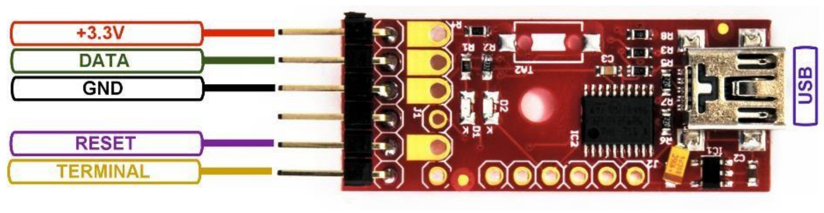
Program via USB connection.

System code: 3320

Configuration line	LED commands	IO commands
P...	All commands For RGB LEDs with PWM	WAITKEY, GETKEY, KEYSTATE GETRTC, SETRTC BEEP GETLDR GETTEMP EEREAD, EEWRITE [0..1023] SYS [COM, CALIB, DFP]

Accessories, Enhancements

Prog-SB, Serial-Basic programming and terminal adapter



LED-Basic components (e.g. Button12, Button16, headlights or basic boosters) that do not have their own USB port require this programming adapter. The adapter enables the same functionality as in components with their own USB port – programming with LED-Basic, BIOS update, and print and error output via the terminal (except Button12).

Prog-SB is available as a single programming adapter with pin strips for connecting the LED-Basic component with jumper cables, or as programming “pliers”, in which the LED-Basic component is connected by means of spring contacts.

When connecting to jumper cables, you need 4 or 5 connections. 4 lines are required to upload the LEDBasic code and to update the BIOS:

- **(PIN1) +3.3v**
- **(PIN2) DATA**
- **(PIN3) GND**
- **(PIN5) RESET**

(PIN 4 and 6 are not used)

If the print and error output is to be displayed via the terminal, PIN6 must also be connected (NB: not available on Button12).

- **(PIN6) Terminal**

(PIN 4 is not used)

The LED-Basic components are supplied with power via the programming adapter. Please remove any battery before programming.

Notes:

- TA2 on the Prog-SB board is not required for the serial basic function and is therefore not available.
- All 6 lines may be connected, the unused cable(s) do not affect the programming.

For Windows 7 and 8.x, Install the driver in the LED-Basic installation package, as described in the "Driver Installation" chapter. No driver installation is required for Windows 10.

DCF77-Time module

AVAILABLE SOON

Status: Waiting for production.

PICTURES TO FOLLOW

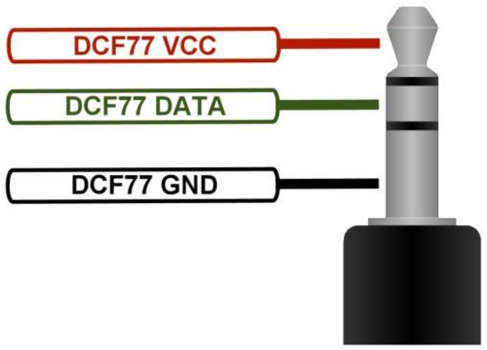
This DCF77 module is suitable for all LED-Basic components with a clock function, provided they have a BIOS version with DCF77 support.

LEDs with integrated PWM (e.g. WS2812, APA102) interfere with DCF77 reception. Place the receiver as far away from these LEDs as possible. The LED in the DCF77 receiver module must blink once per second and must not flicker.

If the reception signal is clean, the LED-Basic component should synchronize to the current time within a maximum of 3 minutes.

Power supply: 1.5-3.5v (approx. 50 µA)

- Output signal: Positive pulses at operating voltage level (no pullup resistance required)
- LED display for reception indication
- Power supply and data return over 3.5mm stereo jack



GPS-> DCF-Time module

AVAILABLE SOON

Status: Development completed, waiting for production

PICTURES TO FOLLOW

If a dcf77 receiver does not receive a signal because, for example, there is too much interference in the surrounding environment, this module can help. As soon as there is a clear view of the GPS satellites, this module receives the accurate time UTC via GPS and converts it into a DCF77 compatibles signal.

- Output signal: Positive or negative pulses at operating voltage (switchable)
- Adjustable time difference, +/- 7 hours relative to UTC
- Optional automatic summer and winter time detection
- 2 LEDs indicate signal and data reception
- Power supply and data over 3.5 mm stereo jack (as in the DCF77 module)
- Power supply: 3.3V (approx. 60mA)

Please note that it may take 3 to 5 minutes for the module to obtain GPS reception, and for the reception LED to remain steady.

WLAN-> DCF-Time module

AVAILABLE SOON

Status: Under development

PICTURES TO FOLLOW

If it is not possible to receive DCF77 or GPS signals, this module may be the answer. Via WLAN, the current time is received from an NTP server, and converted into a compatible DFF77 signal. a smartphone with WLAN reception and web browser is required to configure this module's WLAN access,

- Output signal: Positive or negative pulses at operating voltage level (switchable)
- LED for Data indication
- Power supply and data over 3.5 mm stereo jack (as in the DCF77 module)
- Power supply: 3.3V (approx.??? mA)

Versions

A list of all published LED-Basic versions. This documentation is updated for each version and is therefore not provided separately.

V15.1.15

New components:
LED-Tube, the LED clock in the VFD tube design.
LED-Nixie-4, 4-digit LED Nixie clock
Update: Real-time clock calibration via IO. SYS command on Cronios controller.
Fixed an error with IO.setenc.

V15.1.14

Update: 16-Channel Running Light: fixed bug in BIOS. Added connection pins for Prog-SB.
APA-Booster: Images added and updated.
New components:
VFD clock, a nostalgic clock with VFD tubes.
Touch lamp: effect lamp with sensor buttons and remote control.
6-O-Clock, the 6-digit LED clock.
Nano-stick, the programmable mini-USB stick with 5 RGB LEDs.
New command: IO.sys(a, b) for reading and setting system parameters, for some new components.

V15.1.13

Unpublished trial version.

V15.1.12

Update: New version of the 16-channel light. New and updated sample files for Cortex-clock, Cronixie, light
New: Support for DFF77- time synchronisation module for CHRONIOS1 (other components to follow).

V15.1.11

Update: Cortex-Clock and Cronios-Segmenta now have an EEPROM function, such that brightness, colours and alarm times can be stored.
New and Correction: Added the LED-box connection image. The EEPROM in the LED box has 8 memory locations (0.. 7).
Changes: Button 12 and Button 16 are now called Badge 12 and Badge 16 (Badge = badge, sticker)

V15.1.10

Bugfix: No negative values could be used in data rows.

V15.1.9

Bugfix: Following a label with REM resulted in a run-time error.
New: Added APA-Booster.
Update: New configuration value **Ax** to set the bitrate for APA102 LEDs.

V15.1.8

New: Added LED Matrix 10x10, All-in-One Cortex-M4-Power-Board and LED-Box.
Update: CRONIOS1: EEPROM read/write added (1024 addresses).

V15.1.7

Bugfix: Fixed bug in BIOS flashing.

V15.1.6

New: Added Budget-Board, Cortex-Clock, Temperature-Sensor Interface and Running Light.

V15.1.5

First published version

Errors, Bugs

This is a list of known errors.

V15.1.12
Error in the 16-channel Running Light's BIOS. Please use V15.1.14.

V15.1.11
If an IO function with parameters is used in an if statement, it generates a run-time error 11.

Example:
If IO.eeread(0) < > 10 then...

Solution 1:
t = IO.eeread(0)
if T < > 10 then...

Solution 2:
```
if (IO.eeread(0)) < > 10 then...
```

Notes

Adam Dunkel's µBasic Interpreter was an inspiration for LED-Basic. However, moving the tokeniser to the PC with a self-developed "token-code", as well as the addition of LED-and IO-routines, resulted in almost entire replacement with our own code.

Although this manual has been revised several times and checked, it may still contain factual or grammatical errors. We are very grateful for user error reports or suggestions for improvement. E-mail: feedback@led-basic.de

Links

LED-Basic Homepage
http://www.led-basic.de

LED-Genial Online Shop
http://www.led-genial.de

Diamex-Shop
http://www.diamex.de

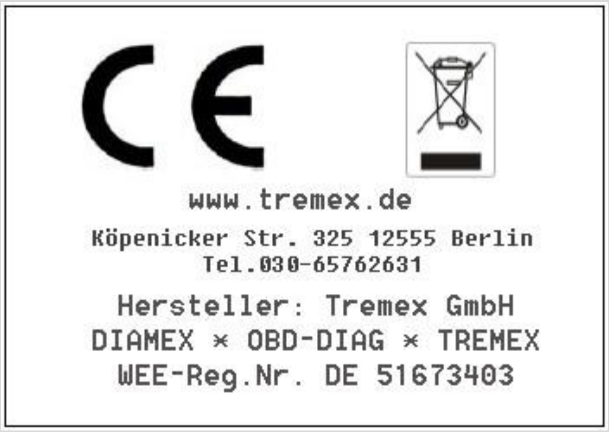
µBasic by Adam Dunkel
http://dunkels.com/adam/ubasic/

SALES

DIAMEX Produktion und Handel GmbH
Innovationspark Wuhlheide Köpenicker Straße 325, Haus 41 12555 Berlin
Phone: 030-65762631

E-mail: Info@diamex.de
Homepage: Http://www.diamex.de

MANUFACTURE


www.tremex.de
Köpenicker Str. 325 12555 Berlin
Tel.030-65762631
Hersteller: Tremex GmbH
DIAMEX ✕ OBD-DIAG ✕ TREMEX
WEE-Reg.Nr. DE 51673403