



FEATURES

- 2.0" 320x240 / 2.8" 320x240 / 3.5" 480x320 / 4.3" 480x272
- AACS-Display (all angle color stability, optimized backlight and TFT-Panel over widest viewing angle)
- Superbright LED backlight over 800 cd/m²
- Object-oriented screen layout
- Change object during run-time: size, shape, color, content
- Animate and move objects, alpha-blending
- Fonts: ASCII and Unicode
- Single supply 3.3 V or directly through USB
- Serial Interfaces: USB, RS232, SPI, I²C
- 8 digital, freely definable I/Os built in, expandable up to 136, 4 analog inputs
- Time (RTC), battery-buffered
- Flash-memory as storage for pictures, fonts, menus and log-files
- Internal functions for calculation as well as programmability
- Tone feedback build-in

ORDERING CODES

2.0" TFT 320x240 dots, PCAP, white LED backlight (external dimensions: 65 x 43 mm)	EA uniTFTs020-ATC
2.8" TFT 320x240 dots, PCAP, white LED backlight (external dimensions: 84 x 58 mm)	EA uniTFTs028-ATC
3.5" TFT 480x320 dots, PCAP, white LED backlight (external dimensions: 100 x 65 mm)	EA uniTFTs035-ATC
4.3" TFT 480x272 dots, PCAP, white LED backlight (external dimensions: 114 x 84 mm)	EA uniTFTs043-ATC

ACCESSORIES

Set incl. 2.8" IPS Display, PCAP and test board with RGB LED and analogue input
 ZIF-connector 40 positions 0.5 mm pitch connector for FPC-cable
 FPC-cable 40 positions, 0.5 mm pitch, 102 mm long
 USB-Kabel Type A -> Mini USB around 1 m

EA DEMOPACK-RGBANA
EA WF050-40S
EA KF050-40
EA KUSB-MINI

Content

General	4
Software	5
Objects	6
Styles / StyleSheets	8
Coordinate system and angle	9
Multi language - String files	10
Boot-menu	11
Firmware-update	12
Protocol / Data transfer	13
Short protocol	15
Small protocol	19
Commands	23
Command syntax	25
Terminal window	27
Text output / strings	33
Pictures	47
Touch functions	49
Draw / graphic primitives	57
Bargraph / instruments	63
Keyboard	69
Input elements	72
Action / Animation	82
Object management	94
Styles	100
Macros	108
Variables/ Registers	124
I/O Port	138
Analogue Input	142
PWM output	143
Serial master interfaces	145
Sound	155
Time	156
Files / Memory	160
System commands	170
Answer / Feedback	179
Functions and Calculations	188
Hardware	198
Pin assignment	200
Power supply	203
Serial interfaces	204
RS232	204
SPI	205
I ² C	207
USB	207
Touch-panel	209
I/O	210
Analogue input	211
PWM output	212
Time	213
Memory	214
Electrical characteristics	215
Dimension EA uniTFTs020-ATC	217
Dimension EA uniTFTs028-ATC	221
Dimension EA uniTFTs035-ATC	225
Dimension EA uniTFTs043-ATC	229

uniTFTDesigner	233
Short cuts	235
Language Editor	237
Register Editor	238
Macro Editor	239
Tools	240
Revision	251

GENERAL

EA uniTFTs are a high-quality all-in-one implementation of the display, microcontroller unit, and touch screen. It is all users need to directly control their application and expedite development, prototyping, and deployment of their HMI/GUI. Design the HMI/GUI using the easy-to-use drag-and-drop uniTFTdesigner graphics development software

The EA uniTFTs series provides sophisticated graphical functions and intuitive menu control with its built-in instruction set. Thanks to the integrated instruction set and the Windows design software uniTFTDesigner, not only electronics specialists, but also experts in the field of design and user guidance are able to create the entire HMI.

The display modules are immediately ready for operation with 3.3 V, controlled via the built-in serial interfaces RS232, SPI, I²C or USB. The modules can be operated directly through the USB, too.

Object-oriented "programming", the wide set of commands, and the integrated but extensible Unicode fonts make "time-to-market" a breeze.

The EA uniTFT series, which forms the high-end market with larger modules, comes up with a very similar command set:

Currently are 3 different sizes available: 5" with 800x480 dots, 7" with 1024x600 dots and 10.1" with 1280x800 dots.

Advantage	Standard TFT	EA uniTFT(s)
Quality	Consumer	Non-consumer
Brightness	250cd	1000cd (typ.)
Viewing angle	Limited +/-50° (typ.)	Up to 340°
Color	TN with Gray inversion effect	IPS: no color shift
Touch	resistive	PCAP incl. controller
Interface	8-/16 Bit data bus	I ² C or SPI or USB
Availability	Minimum order quantity	Ex stock
Longevity	1 year or more	Minimum of 8 years
Support	None unless high quantity	Bundled with product purchase
ce / EMC	None	Tested and certified
Software	Place dot by dot to create character or touch buttons	- High level commands included - Graphics development software f.o.c.

SOFTWARE - OPERATION OF EA UNITFTS-SERIES

The presentation on the display is based on the given commands. The commands can either be transmitted at runtime via one of the serial interfaces or combined on the internal memory in so-called macros and stored permanently. With the help of the commands, graphic objects are created. These objects have different properties, like color, position and built-in actions. These properties can be changed at any time, for example a string or the position of a touch-sensitive button can be changed.

All conceivable objects can be arbitrarily placed, moved and deleted. Windows font sets are stored directly in the display's memory. Thanks to automatic ASCII and Unicode switching, a wide variety of systems are supported flexibly, Chinese characters included. Elegant effects like fading in or out are already integrated. Style sheets can be used to create consistent designs. Images JPEG, PNG and many more (also transparent) can be integrated. Together with the integrated (EA uniTFTs035-ATC and EA uniTFTs043-ATC), battery-buffered time base, events can be documented together with a time stamp or processes can be controlled completely autonomously without an external computer.

Objects

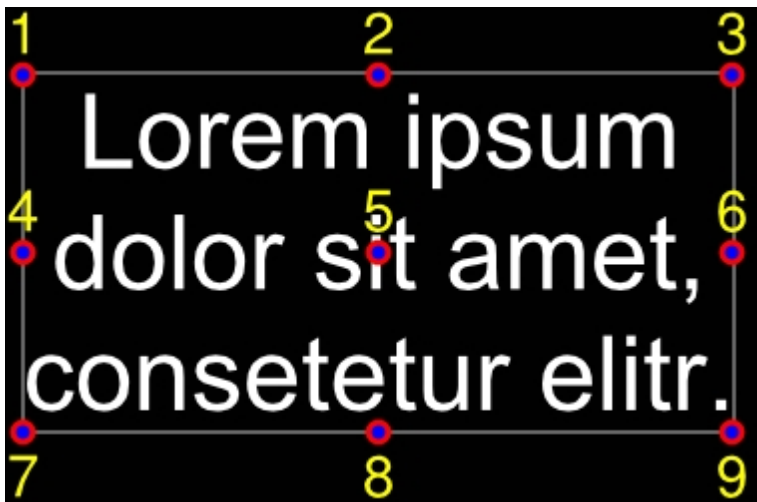
Every picture, text element and button is a so-called object. Each object got its own, individual object ID, which makes it uniquely identifiable. The object ID can be used to change the properties of an object at any time (size, position...). You can use 0 as ID for creating simple graphical objects. These objects are rendered directly to the background and aren't editable and manipulable any more. If you assign an already existing object ID to a new object, the previous object will be overwritten.

Commands for object management can be found [here](#).

Object position / Anchor

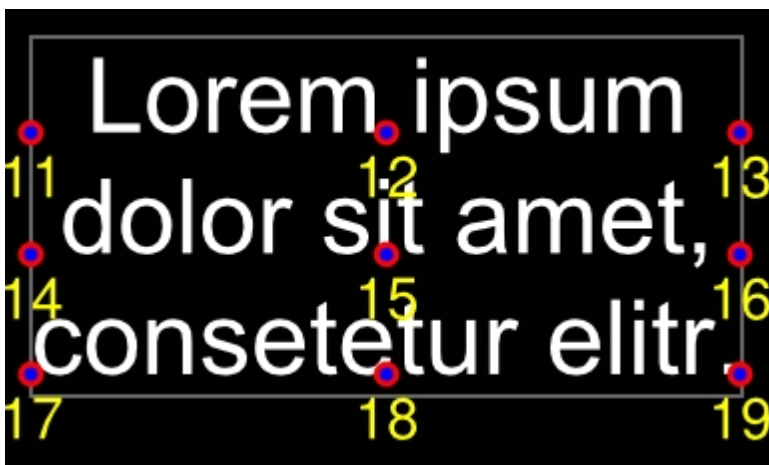
General anchors

The position of an object is based on the coordinates (origin: bottom left edge) related to the object anchor. Each object has 9 fixed anchors. Transformation on the object (e.g. rotation or shear) will be applied to the active anchor.



Strings and anchors

Strings have additional 9 anchors used to align objects (e.g. an underscore line) to the text base line.



Special case: Anchor 0

Each object has additionally a freely definable anchor. For circles, ellipses, and stars, the object anchor 0 is the

construction point.



Example: The pointer should rotate around the centre of the circle. The pointers 9 standard anchors (shown in dark grey) are not useful in this case because none of the defaults are located in the right position. The anchor 0 can be placed pixel-precise ([#OAS](#)) as shown, and this custom location marks the correct rotation point for the pointer object.

Styles / StyleSheets

Styles can be used to create consistent design. There are

- DrawStyles
- TextStyles
- ButtonStyles

Before placing any graphic object or text object, a DrawStyle or a TextStyle need to be defined. A DrawStyle defines the pen type and a fill color and the TextStyle the font and it's size.

DrawStyle:

Color, gradients, pattern and pen for (out)lines are defined in a DrawStyle.

TextStyle:

The appearance of a string is defined in a TextStyle. A TextStyle is based on a DrawStyle for color and some font specification for size, alignment and spacing.

ButtonStyle:

Touch buttons and switches are defined by a ButtonStyle, which consists of a TextStyle for labeling and DrawStyles for background painting.

ColorRamp:

Filling an object can be done with solid color or with some gradient. Those gradient and its colors are defined in ColorRamps and can be used linear or radial.

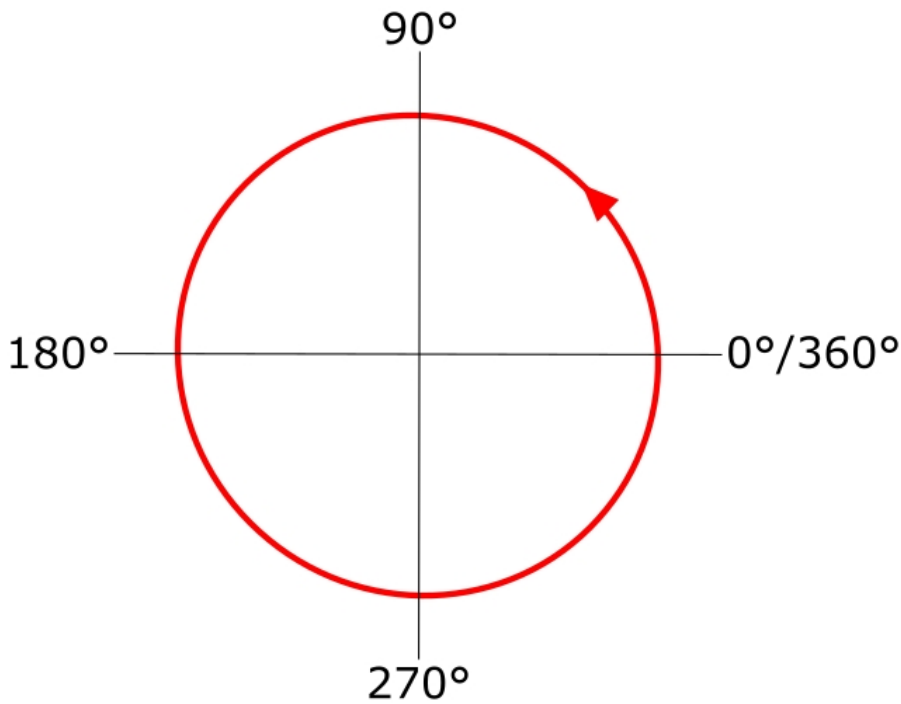
The Windows tools uniTFTDesigner supports StyleSheets that contain a collection of several Draw, Text, and ButtonStyles and also ColorRamps.

The commands related to styles and colorramps can be found [here](#).

Coordinate system and angle

The coordinate system refers directly to the display resolution of the module with the origin 0|0 placed in the lower left corner of the display. For example the EA uniTFTs028-A has a drawing field of 320 x 240 dots. Valid coordinates for this display are 0..319 and 0..239 hence.

Angles are given in the mathematical sense of rotation (counter-clockwise). 0° is horizontally right. Besides instruments rotation is available in 90° steps:



Multi language - String files

In an increasingly interdependent world of international assignments, supporting multiple languages is a must. The EA uniTFTs-Series with its unicode support is part of the solution. Without unicode it's basically impossible to work with Chinese characters e.g..

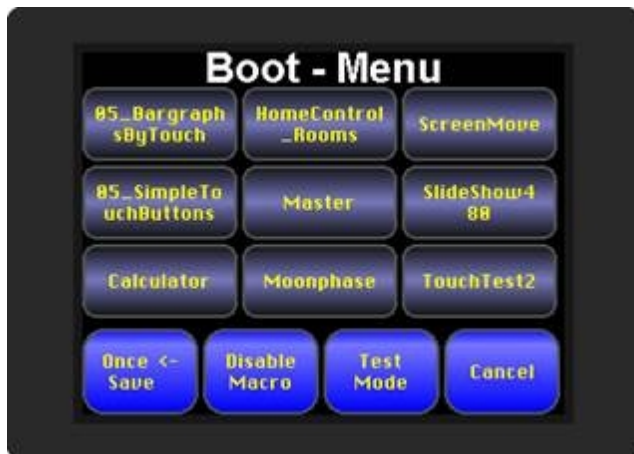
The second part of supporting internationalization are string files: these text files provide a database of strings to be displayed. In macro files, strings are referenced by an index, then at runtime this index is replaced with the corresponding text taken from the string file.

Further details can be found by looking at the command description under [#VFL](#) or the [examples](#).

Boot menu

Multiple projects can be stored on the integrated memory. The project which is started automatically is defined using the "start.emc" file. To load a different project, the start file need to be updated, or on touch enabled panels, a project can be selected via the boot menu:

When switching on the device (or after hardware reset), wipe over the touch panel several times in short interval.



To avoid mis-use by the operator, the boot menu can be deactivated. For this purpose, an empty file named "bootmenu.off" must be placed in the root directory of the memory. This can be done using mass storage mode and Windows Explorer to transfer the file, or directly via uniTFTs commands:

```
#FWO</bootmenu.off>
#FWC
```

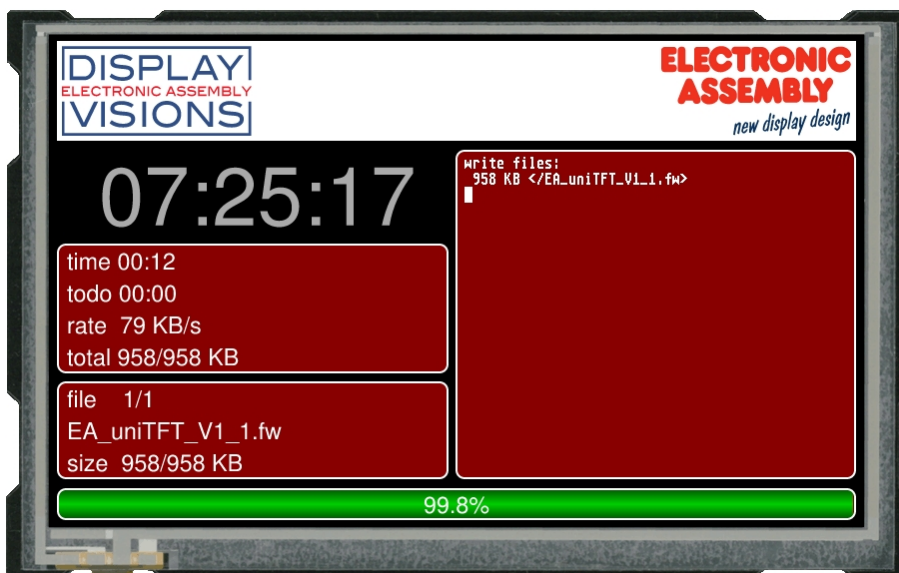
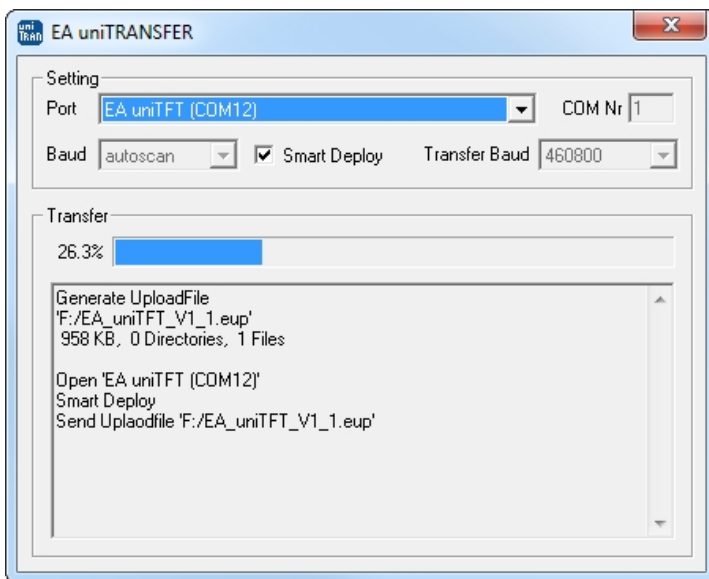
In addition to project selection, the boot menu offers the option to start test mode, or to display information about the module. It provides the version, protocol status, baudrate, SPI mode, I²C bus address etc

Firmware-update

To use the latest features of the EA uniTFTs-Series, it might be necessary to update the internal firmware of the module.

Firmware-update via serial interface and Windows PC:

- Save the firmware file (e.g. EA_uniTFTs_V1_1.fw) to your local drive
- Connect the EA uniTFT with your PC
- Start uniTRANSFER.exe (found in the Simulator_and_Tools folder of the uniTFTDesigner installation) and select the correct serial interface to the EA uniTFTs.
- Drag'n'Drop the firmware file to the EA uniTRANSFER window.



- After transferring the data, a manually reset needs to be performed, then the firmware will be loaded automatically after restart. **Attention: Please do not switch of the module while updating.**

Firmware update via serial interface

The firmware file also can be transferred to EA uniTFT with any system. To do this, transfer the contents of the * .fw file 1:1 (with protocol in packets) to the EA uniTFT. The transfer progress will become visible on the display module. After successful transfer, a data check will be done automatically. If the data is correct, the update starts automatically.

Attention: Please do not switch of the module while updating.

PROTOCOL / DATA TRANSFER

The transmission protocol is identical regardless of which of the 4 serial interfaces is used to transfer data from the higher-level controller. The hardware circuit for each interface varies, which is described under the chapter "[serial interfaces](#)".

The data transfer is embedded in a fixed frame with checksum. The EA uniTFTs-Series acknowledges this packet with the character <ACK> (= 0x06) on successful reception or <NAK> (= 0x15) when it detects a faulty checksum or encounters a buffer overflow. In case of a <NAK>, the complete packet is discarded and has to be sent again. An <ACK> only confirms the correct transmission. A syntax check does not take place.

Two different protocols are implemented, the "[Short Protocol](#)" and the "[Small Protocol](#)". The short protocol works with a CRC16 checksum and allows the transfer of larger data packets. The Small protocol was implemented mainly for compatibility with the EA eDIPxxx series.

The maximum amount of user data per packet is 2042 bytes or 255 bytes, respectively. Commands that are larger (for example, image or file transfers, [#FWD](#) ...) must be split into several packets. The data in the individual packets is reassembled by the display module after receiving them successfully.

Remark:

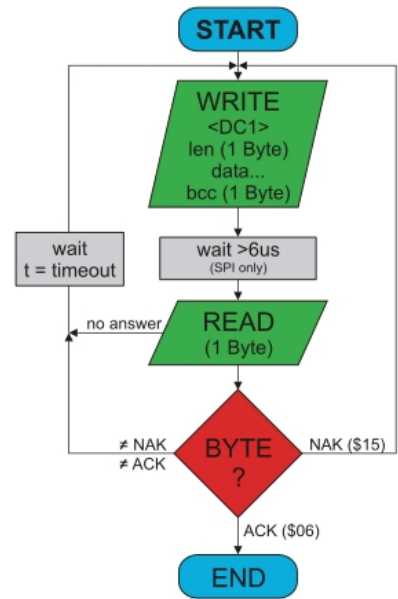
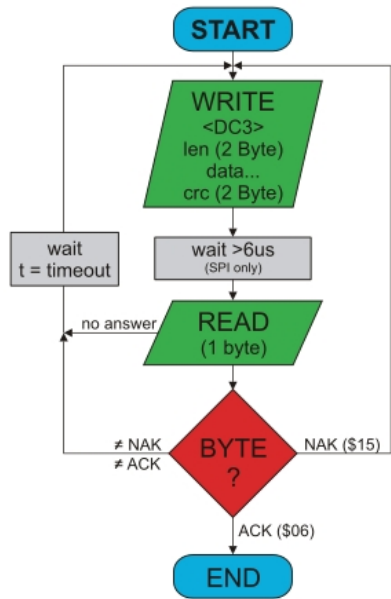
The <ACK> has to be read (SPI and I²C). If the master doesn't receive the acknowledgement, at least one byte is lost. In this case the time-out time needs to be observed before the packet is resent.

The protocol can be disabled on the serial port for testing purposes. To turn off the protocol, pin 14 has to be set low (see [pin assignment](#)).

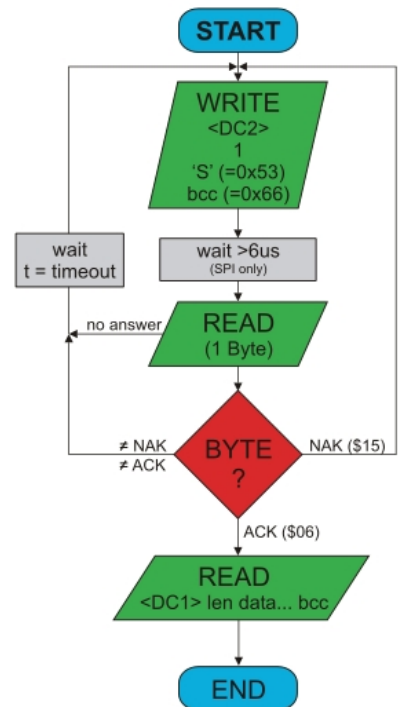
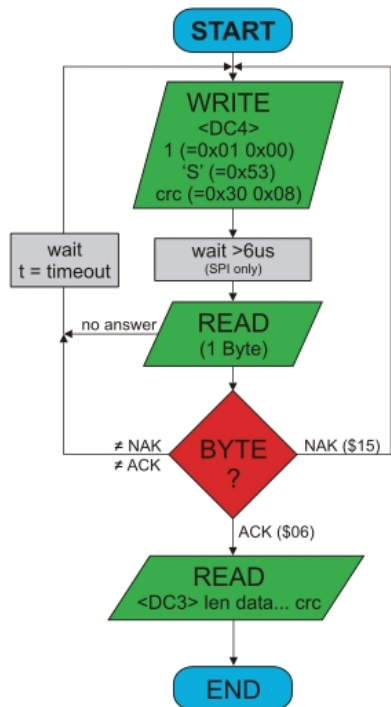
Short protocol

Small protocol

Send



Receive



Short Protocol commands

1. Commands / sending data to module

This protocol command transfers data to the display. Several graphics commands can be packaged into a single protocol package. If the amount of data is larger than the maximum packet size, the data can be split into several packets. The module reassembles the individual data packets.

The 16 Bit data are defined as little-endian (Intel format), means that the lower byte need to be sent first.

Module receives	DC3 0x13	length (16 Bit) 0xXX 0xXX	Data..... 0x....	crc (16 Bit) 0xXX 0xXX
Module sends	ACK 0x06			

Example: **#XCB20** changes the brightness to 20%. The command need to be terminated with [LF] which is **0x0A**. So the Short Protocol packet starts with DC3 followed by the **length** (count of data). At the end there's a **CRC16** (CCITT) necessary, calculated with all bytes. Here's a link to an [Online-CRC-Calculator](#).

Hex: **13 07 00 23 58 43 42 32 30 0A 3D CD** (here you get it [as a file](#); this may be put to terminal.exe via drag-n-drop)

Example: **#XCB80** changes the brightness to 80%.

Hex: **13 07 00 23 58 43 42 38 30 0A FC 0A** (here you get it [as a file](#))

2. Request data of send buffer

If data is generated in the module, it is stored in the module's send buffer. The data can be requested via the serial interfaces. Whether data is available can be monitored via the pin 20 SBUF, or the higher-level controller can cyclically poll the data.

Module receives	DC4 0x14	length (16 Bit) 0x01 0x00	'S' 0x53	crc (16 Bit) 0x30 0x08
Module sends	ACK 0x06			
Module sends	DC3 0x13	length (16 Bit) 0xXX 0xXX	Data..... 0x....	crc (16 Bit) 0xXX 0xXX

3. Repeat last data packet

If a packet received from the module is faulty (wrong length or checksum) it can be requested again:

Module receives	DC4 0x14	length (16 Bit) 0x01 0x00	'R' 0x52	crc (16 Bit) 0x11 0x18
Module sends	ACK 0x06			
Module sends	DC3 0x13	length (16 Bit) 0xXX 0xXX	Data..... 0x....	crc (16 Bit) 0xXX 0xXX

4. Request buffer information

This command queries whether user data is ready (= Pin13 SBUF) and also indicates how much free space is left in the device's receive buffer.

Module receives	DC4 0x14	length (16 Bit) 0x01 0x00	'I' 0x49			crc 0x4
Module sends	ACK 0x06					
Module sends	DC4 0x14	length (16 Bit) 0x04 0x00	send buffer bytes ready (16 Bit) 0xXX 0xXX	receive buffer bytes free (16 Bit) 0xXX 0xXX		crc 0xX

5. Protocol settings

This can be used to limit the maximum packet size that the display may send. The default maximum packet size is 2042 bytes. Furthermore, the time-out can be set in 1 / 1000s. The time-out is activated when individual bytes have been lost. After the timeout, the entire packet must be retransmitted.

Module receives	DC4 0x14	length (16 Bit) 0x05 0x00	'D' 0x44	packet size send buffer (16 Bit) 0xFA 0x07 (=2042 Byte)	Time-out (16 Bit) in ms 0xD0 0x07 (=2 seconds)	crc (16) 0x98 0
Module sends	ACK 0x06					

6. Protocol information

Request protocol settings (see 5.).

Module receives	DC4 0x14	length (16 Bit) 0x01 0x00	'P' 0x50			
Module sends	ACK 0x06					
Module sends	DC4 0x14	length (16 Bit) 0x06 0x00	maximum packet size send buffer (16 Bit) 0xFA 0x07 (=2042 Byte)	packet size Send buffer (16 Bit) 0xXX 0xXX		T 0

7. RS485 address select / deselect

With this command, a module can be selected or deselected on the RS485 bus. By default, the module with address 7 is always active.

Module receives	DC4 0x14	length (16 Bit) 0x03 0x00	'A' 0x41	'S' (=select) 'D' (=deselect) 0x53 or 0x44	RS485-address 0xXX	crc (16) 0xXX 0
Module sends	ACK 0x06 ----	→ select → deselect				

8. RS485 enable signal - delay

Some RS485 masters take some time to change the enable signal, e.g. to switch from write to read mode. In order to enable successful communication with these devices, this command can be used to delay switching to write mode.

Module receives	DC4 0x14	length (16 Bit) 0x03 0x00	'T' 0x54	Delay in 10 us 0x00 0x00	crc (16 Bit) 0xE9 0x7E
Module sends	ACK				

	0x06
--	------

9. Request interface exclusively

All 4 serial ports are handled in parallel and equivalently after reset. To ensure that a sequence of protocol packets is executed without interruption, the other serial interfaces can be disabled so the active interface can communicate with the module exclusively. This is useful, for example, for a project update via USB.

Module receives	DC4 0x14	length (16 Bit) 0x02 0x00	'G' 0x47	0x00 = Release 0x01 = Request	crc (16 Bit) 0xXX 0xXX
Module sends	ACK 0x06				
Module sends	DC4 0x14	length (16 Bit) 0x01 0x00	active (16 Bit) 0x00 = all 0x01 = RS232 0x02 = SPI 0x03 = IIC 0x04 = USB		crc (16 Bit) 0xXX 0xXX

10. Break-Command, Break / Stop execution

If a continuous loop has been programmed in a macro or if a normal process flow is blocked, this command can be used to interrupt and quit. This command is also suitable for update processes.

Module receives Default values	DC4 0x14	length (16 Bit) 0x02 0x00	'C' 0x43	break 0x01 = Wait command 0x02 = actual macro file 0x04 = Clear send buffer 0x08 = Clear receive buffer 0x10 = Delete macro definitions (e.g. port macros) 0xFF = Stop everything	crc (16 Bit) 0xXX 0xXX
Module sends	ACK 0x06				

11. Hardware Reset

The module is restarted with this protocol command. Depending on the parameter, various start options can be selected to automatically run after the reset.

Module receives Default values	DC4 0x14	Länge (16 Bit) 0x02 0x00	'B' 0x42	Option 0x00 = normal restart 0x01 = Restart with test mode 0x02 = Restart without running 'start.emc' 0x03 = Restart without loading default styles 0x04 = Show boot-menu (project selection) 0x05 = Reserved 0x06 = Mass Storage Mode (from V1.2)	crc (16 Bit) 0xXX 0xXX
Module sends	ACK 0x06				

CRC-Calculation

A cyclic redundancy check (CRC) is used to calculate the checksum. A common and well known CRC exam is the

CRC-CCITT. The starting value is 0xFFFF. The following is a typical C implementation. The functions must be called externally. The checksum must be preallocated with the starting value.

```
//-----
//function: buffer2crc16()
//input:   ptr data, ptr CRC, block length
//output:  ---
//descr:   CRC-CCITT of a buffer
//-----
void buffer2crc16(UBYTE *dat, UINT16 *pCRC, UINT32 len)
{
    while(len--)
        crc16(*dat++, pCRC);
}

//-----
//function: sp_crc16()
//input:   data, ptr CRC
//output:  ---
//descr:   CRC_CCITT (x^16+x^12+x^5+1 = 1 0001 0000 0010 0001 = 0x1021)
//-----
void crc16 (UBYTE dat, volatile UINT16 * crc)
{
    register UINT16 lcrc = *crc;
    lcrc = (lcrc >> 8) | (lcrc << 8);
    lcrc ^= dat;
    lcrc ^= (lcrc & 0xFF) >> 4;
    lcrc ^= lcrc << 12;
    lcrc ^= (lcrc & 0xFF) << 5;
    *crc = lcrc;
}
```

Small Protocol commands

1. Commands / data send to module

This protocol command transfers data to the display. Several graphics commands can be packaged in a protocol package. If the data is larger than the maximum packet size, the data can be split into several packets. The module reassembles the individual data packets.

Module receives	DC1 0x11	length (8 Bit) 0xXX	Data..... 0x....	bcc (8 Bit) 0xXX
Module sends	ACK 0x06			

Example: **#XCB25** changes the brightness into 25%. The command need to be terminated with [LF] which is **0x0A**.

So the Small Protocol packet starts with DC1 followed by the **length** (count of data). At the end there's a **bcc** (8 bit summary, modulo 256) necessary, calculated with all bytes. Here's a link to an [Online-CRC-Calculator](#).

Hex: **11 07 23 58 43 42 32 35 0A 89** (here you get it [as a file](#); this may be put to terminal.exe via drag-n-drop)

Example: **#XCB75** changes the brightness into 75%.

Hex: **11 07 23 58 43 42 37 35 0A 8E** (here you get it [as a file](#))

2. Request data of send buffer

If data is generated in the module, it is stored in the module's send buffer. The data can be requested via the serial interfaces. Whether data is available can be monitored via the pin 20 SBUF, or the higher-level controller can cyclically poll the data.

Module receives	DC2 0x12	length (8 Bit) 0x01	'S' 0x53	bcc (8 Bit) 0x66
Module sends	ACK 0x06			
Module sends	DC1 0x11	length (8 Bit) 0xXX	Data..... 0x....	bcc (8 Bit) 0xXX

3. Repeat last data packet

If a received packet of the module is faulty (wrong length or checksum) it can be requested again:

Module receives	DC2 0x12	length (8 Bit) 0x01	'R' 0x52	bcc (8 Bit) 0x65
Module sends	ACK 0x06			
Module sends	DC1 0x11	length (8 Bit) 0xXX	Data..... 0x....	bcc (8 Bit) 0xXX

4. Request buffer information

This command queries whether user data is ready (= Pin13 SBUF) and also indicates how much free space is left in the device's receive buffer.

Module receives	DC2 0x12	length (8 Bit) 0x01	'I' 0x49			bcc 0x5
Module sends	ACK 0x06					
Module sends	DC2 0x12	length (8 Bit) 0x02	send buffer bytes ready (8 Bit) 0xFF	receive buffer bytes free (8 Bit) 0xFF		bcc 0x

5. Protocol settings

This can be used to limit the maximum packet size that the display may send. As default a packet size with up to 2042 bytes of user data is set. Furthermore, the time-out can be set in 1 / 1000s. The time-out is activated when individual bytes have been lost. After the timeout, the entire packet must be retransmitted.

Module receives Default values	DC2 0x12	length (8 Bit) 0x03	'D' 0x44	packet size send buffer (8 Bit) 0xFF	Time-out (8 Bit) in 1/100s 0xC8 (=2 seconds)	bcc (8) 0x20
Module sends	ACK 0x06					

6. Protocol information

Request protocol settings (see 5.).

Module receives	DC2 0x12	length (8 Bit) 0x01	'P' 0x50			
Module sends	ACK 0x06					
Module sends	DC2 0x12	length (8 Bit) 0x03	maximum packet size send buffer (8 Bit) 0xFF	packet size send buffer (8 Bit) 0xFF	Time-out (8 Bit) in 1/100s 0xC8 (=2 seconds)	bcc (8) 0x20

7. RS485 address select / deselect

With this command, a module can be selected or deselected on the RS485 bus. By default, the module with address 7 is always active.

Module receives Default values	DC2 0x12	length (8 Bit) 0x03	'A' 0x41	'S' (=select) 'D' (=deselect) 0x53 or 0x44	RS485-address 0xFF	bcc (8) 0xFF
Module sends	ACK 0x06	→ select ---- → deselect				

8. RS485 enable signal - delay

Some RS485 masters take some time to change the enable signal, e.g. to switch from write to read mode. In order to enable successful communication with these devices, this command can be used to delay switching to write mode.

Module receives Default values	DC2 0x12	length (8 Bit) 0x03	'T' 0x54	Delay in 10 us 0x00 0x00	bcc (8 Bit) 0x69
Module sends	ACK 0x06				

9. Request interface exclusively

All 4 serial ports are handled in parallel and equivalently after reset. To ensure that a sequence of protocol packets is executed without interruption, the other serial interfaces can be disabled so the active interface can communicate with the module exclusively. This is useful, for example, for a project update via USB.

Module receives	DC2 0x12	length (8 Bit) 0x02	'G' 0x47	0x00 = Release 0x01 = Request	bcc (8 Bit) 0xXX
Module sends	ACK 0x06				
Module sends	DC2 0x12	length (8 Bit) 0x01	active (8 Bit) 0x00 = all 0x01 = RS232 0x02 = SPI 0x03 = IIC 0x04 = USB		bcc (8 Bit) 0xXX

10. Break-Command, Break / Stop execution

If a continuous loop has been programmed in a macro or if a normal process flow is blocked, this command can be used to interrupt and quit. This command is also suitable for update processes.

Module receives Default values	DC2 0x12	length (8 Bit) 0x02	'C' 0x43	break 0x01 = Wait command 0x02 = actual macro file 0x04 = Clear send buffer 0x08 = Clear receive buffer 0x10 = Delete macro definitions (e.g. port macros) 0xFF = Stop everything	bcc (8 Bit) 0xXX
Module sends	ACK 0x06				

11. Hardware Reset

The module is restarted with this protocol command. Depending on the parameter, various start options can be selected to automatically run after the reset.

Module receives Default values	DC2 0x12	length (8 Bit) 0x02	'B' 0x42	Option 0x00 = normal restart 0x01 = Restart with test mode 0x02 = Restart without running 'start.emc' 0x03 = Restart without loading default styles 0x04 = Show boot-menu (project selection) 0x05 = Reserved 0x06 = Mass Storage Mode (from V1.2)	bcc (8 Bit) 0xXX
Module sends	ACK 0x06				

BCC-Calculation

The calculation of the checksum requires a simple 8-bit sum test (modulo 256). The following is a typical C implementation.

```
//-----
```

```
//function: buffer2bcc()
//input:   ptr data, block length
//output:  Byte bcc
//descr:   calculate bcc for a buffer
//-----
UBYTE buffer2bcc(UBYTE *dat, UBYTE len)
{
    UBYTE bcc = 0;
    while(len--)
        bcc += *dat++;
    return bcc;
}
```

COMMAND SUMMARY

The EA uniTFTs-Series has an integrated command set, including graphical commands, calculations, hardware commands and many more.

The commands can be transmitted at runtime via the serial interfaces or stored in so-called macro files on the module's FLASH memory.

The following tables describe all commands. The default values are given in brackets after the respective parameters. BLACK written parameters must be set, GRAY ones are optional.

All command groups at a glance

Terminal window #Y	In the terminal window, all received data is displayed directly. This window is useful for quickly creating simple outputs or receiving error messages during development time.
Text output / strings #S	The group includes commands to display simple, formatted and self-changing strings. In addition, there is the possibility to place texts using edit boxes (one line inputs) and string boxes (multiple lines output).
Picture #P	Command group to display pictures. The design software uniTFT Designer automatically converts the data into the correct internal format. The design software uniTFT Designer allows to use following filetypes/graphic formats: png, bmp, jpg, jpeg, tga, gif, g16, svg, svgz. If uniTFT Designer is not used, those files can be converted by EAconvert.exe (directory \Simulator_and_Tools) (-> evg, epg, epa)
Touch functions #T	Command group to enable touch functions. Simple buttons and switches can be used, as well as radio buttons, sliders, bar-graphs and rotary or meter instruments.
Draw / graphic primitives #G	Command group to show graphical simple objects:
Bargraph / Instruments #I	Command group to show bar graphs, sliders and rotary / pointer instruments
Input elements per Touch #E	Commands to create touch input elements like menus, SpinBoxes or ComboBoxes.
Keyboard #K	Command group to represent a keyboard for value inputs. Normally, the keyboard is connected to an EditBox.
Action / Animation #A	Command group to animate objects, e.g. Show, fly away, rotate or fade out.
Object management #O	Command group to manage, modify and group objects.
Styles #C	Command group to create styles. The look of each object is based on a style appropriate to the object type. The maximum number of styles available for each style is 100.

Macros #M	Single or multiple command sequences can be collected as so-called macros and which reside in individual files stored in internal FLASH. The following commands describe how to work with macros.
Variables / Registers #V	Command group to execute calculations and logical operations. With the help of the string files, internationalization (multiple languages) can be realized. There are registers for numbers and strings (can record characters up to 200), integer registers use signed 32-bit, floating-point registers use 23-bit mantissa, 8-bit exponent, 1-bit signed.
I/O Port #H	I/O port lines, which can be expanded to up to 136. If the port input pins are changed, macros can be started, see #MHP , and #MHB .
Analog Input #H	Command group to parameterize and read out the analog input of the module. The module has four 12-bit analog inputs. If the analog input changes, a macro can be started, see #MHA .
PWM output #H	Command group for the PWM output
Serielle Master-Interface #H	Command group to use the 3 serial interfaces of the module as master interface. For example to connect additional peripherals like temperature sensor
Sound #H	Command group to play tones
Time #W	Command group to work with the built-in RTC.
Files / access to FLASH #F	Commands to handle file access on the built-in FLASH
System commands #X	Settings of the EA uniTFTs-Series.
Answer / Feedback	The module stores information in its send buffer after requests or touch events. Description of the module's answers.

Command syntax

All commands have same structure:

Start	Command code	Parameter	Ending
#	XXX	123, \$52, %01101010, "Hello"; R0	[CR]LF
ASCII : 35 (0x23) UniCode: 23 (0x23 0x00)	3-letter command	Parameters	ASCII : [13] 10 ([0x0D] 0x0A) UniCode: [13] 19 ([0x0D 0x00] 0x) CR is optional

All parameters are transferred as 16-bit values (unless otherwise stated).

Parameter

Numbers

- 123 decimal passed as ASCII characters
- \$5A hexadecimal passed as ASCII character
- % binary passed as ASCII character
- 1010001
- 5-8 Passing of a range. Commands influencing multiple objects can be passed using an object range. The given example relates to object-IDs 5,6,7,8
- ?x Code of a single character (Unicode/ASCII)
- R0 ... R499 Passing a register value
- Q0 ... Q499 Indexed passing of a register value (pointer) → R (R0 ... R499)
- (...) [Calculation-string](#) used for return value
- G len32 data... Passing binary data: len32 defines the data length (already passed as binary 32-bit value)
- lindex! Use values of a string file's index

Strings

- "string" or 'string' standard string passing
- "str" 32 "str" simple string with any code in between, which is incorporated into the final string
- "str1"; "str2" **Semicolon represents the string ending.** Important if passing two strings or following parameters after a string.
- S0 ... S499 Passing string registers
- T0 ... T499 Passing string registers using a register as index S(R0 ... R499).
- U"Hello" Interpret characters after U as 16 bit uni code (until next # or V, also CR + LF)
- V"Hello" Interpret characters after U as 8 bit ASCII (until next # or U, also CR + LF)
- lindex! Use string of a string file's index

Each parameter is separated by blank (' '), comma (','), semicolon(';') or point ('.'). **For separating Strings you have to use a semicolon.**

Ending

The ending always is a LF (0x0A). A leading CR (0x0D) is optional and skipped over.

Comments

Comments can be included into macro files. A comments starts with #- and is active till line end (LF).

Calculation

Every numerical parameter can be replaced by a calculation string. The calculation needs to be inside parentheses () to be passed as one parameter. The documentation on [calculation commands](#) lists all operations and functions, including mathematical, logical as well as module-specific operations, e.g. time or object properties.

Terminal window #Y

The terminal window is useful for quick test of serial interface connection; for that put the pin 14 (DPROT) to GND (switch off the [protocol](#)) and then power-up the display. All data received from serial interface are displayed directly (ASCII codes and CR/LF). FF clears the terminal window and set the cursor position to home position.

The terminal windows provides also an easy way for simple outputs and error messages during development.

Terminal window settings

Size and position settings (Terminal Define Window)	#YDW	x(0), y(0), Anchor(7), Columns(x-DisplayResolution/8), Rows(y-DisplayResolution/16)
Color settings (Terminal Define Color)	#YDC	Text-Color, Text-Opacity(100), Background-Color(\$000000), Background-Opacity(0)
Drawing order (Layer) (Terminal Define Layer)	#YDL	Layer
Terminal window on/off (Terminal Define Output)	#YDO	Output, Visibility(=Output)
Cursor on/off (Terminal Cursor Blink)	#YCB	Cursor
Set cursor position (Terminal Cursor Position)	#YCP	Column, Row(no change)
Save cursor position (Terminal Cursor Save)	#YCS	
Restore cursor position (Terminal Cursor Restore)	#YCR	

Terminal window output

Print string (Terminal Print Ascii)	#YPA	String
Print formatted string (Terminal Print Formated)	#YPF	"Formatted string"; Value1, Value2, ..., ValueN
Print date/time (Terminal Print Date)	#YPD	"Dateformat"; date (act. time)
Print module information (Terminal Print Info)	#YPI	
Print firmware version string (Terminal Print Version)	#YPV	

Terminal window settings

All important settings of the terminal window are summarized in this command group.

Size and position settings

#YDW	x(0), y(0), Anchor(7), Columns(x-DisplayResolution/8), Rows(y-DisplayResolution/16)
-------------	---

The command defines the dimensions of the terminal window. The width results from the specification of the columns and rows and the font size (8x16): Width in pixels = 8 * columns; Height in pixels = 16 * lines

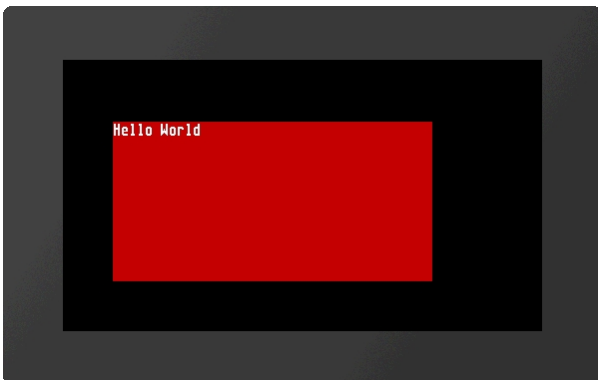


```
...
#YDW 50,50,7,40,10
...
```

Color settings

#YDC	Text-Color, Text-Opacity(100), Background-Color(\$000000), Background-Opacity(0)
------	--

The command sets the color and opacity of the font and the background. The color is transferred as a 24-bit RGB value (e.g. \$c80000, %110010000000000000000000, (RGB(200,0,0))).



```
...
#YDC $ffffff,100,$c80000
...
```

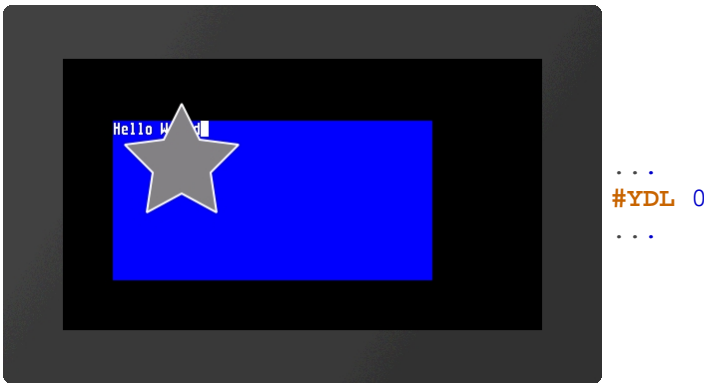
Drawing order (Layer)

#YDL	Layer
------	-------

The command sets the drawing order (**Layer**) of the terminal window:

Layer	
0	Terminal is shown behind all objects
1	Terminal is shown on top

By default, the terminal is always on top.



Terminal window on/off

#YDO Output, Visibility(=Output)

With this command the terminal output can be activated or deactivated and the visibility can be set. If only one parameter is passed, it applies to both values.

Definition of the **Output**:

Output	
0	Terminal output is deactivated
1	Terminal output is activated

Definition of the **Visibility**:

Visibility	
0	Terminal is invisible
1	Terminal is visible

#YDO 0	Outputs are disabled and the terminal is invisible
#YDO 1	Outputs are activated and the terminal is visible
#YDO 0,1	Outputs are deactivated and the terminal is visible
#YDO 1,0	Outputs are activated and the terminal is invisible.

Cursor on/off

#YCB Cursor

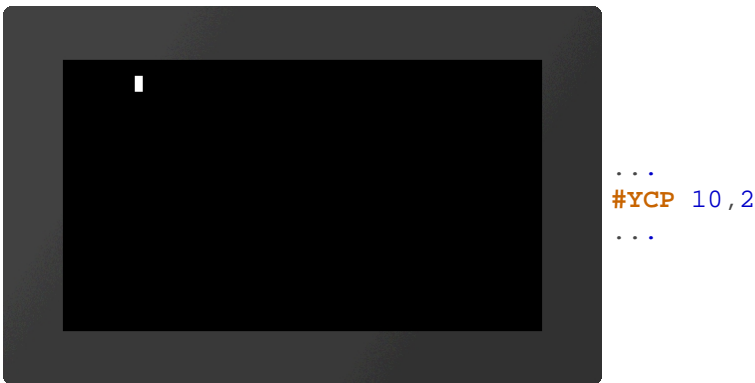
The command sets the visibility of the **Cursor**:

Cursor	
0	Cursor is invisible
1	Cursor is visible

Set cursor position

#YCP	Column, Row(no change)
------	------------------------

The command sets the cursor position within the terminal window. If no line is specified, it is not changed. The position starts at (1,1).



Save cursor position

#YCS	
------	--

The current position of the cursor is saved.

Restore cursor position

#YCR	
------	--

The cursor is placed on the last saved position.

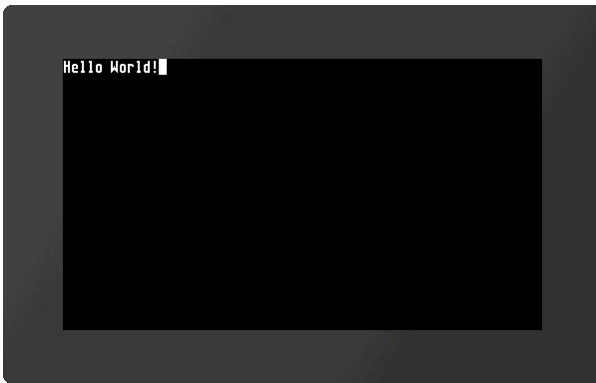
Terminal window output

This group includes commands to display strings and predefined outputs on the terminal.

Print string

#YPA	String
------	--------

The characters (strings) are displayed in the terminal window. Entire character strings (e.g. "Test", 'Test') or individual ASCII characters (\$21, 33, ?!) can be transferred. The semicolon forms the end of the string.

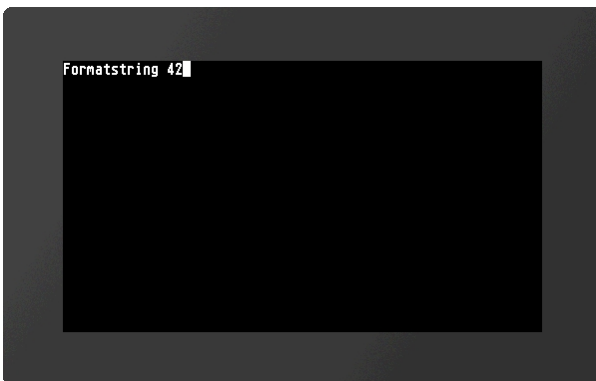


```
...
#YPA "Hello World" $21;
...
```

Print formatted string

#YPF	"Formatted string"; Value1, Value2, ..., ValueN
------	---

The **formatted string** is displayed on the terminal. If the variable set repeats, the format string is used again. The structure is explained in more detail in the section [Formatted string](#).

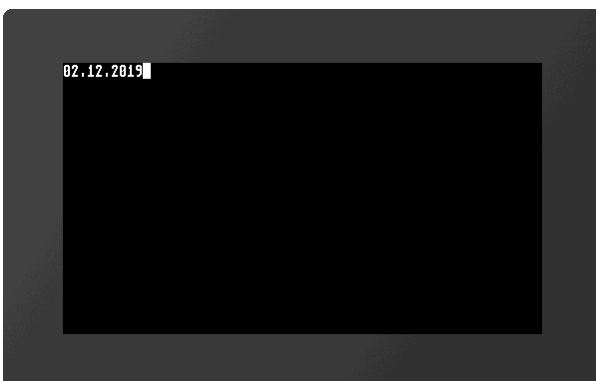


```
...
#YPF "Formatstring %d"; 42
...
```

Print date/time

#YPD	"Dateformat"; date (act. time)
------	--------------------------------

The date and time are displayed on the terminal. The way of presentation is based on the date format. The structure is explained in more detail in the section [Date formats](#).



```
...
#YPD "%D.%M.%Y";
...
```

Print module information

#YPI	
------	--

Module parameters (e.g. firmware version, resolution, or interface parameters) are displayed in the terminal

Print firmware version string

#YPV	
------	--

The firmware version of the module is displayed in the terminal.

Text output / string #S

The group includes commands to display simple, formatted and self-changing strings. In addition, there is the possibility to place texts using edit boxes (one line inputs) and string boxes (multiple lines output).

Simple strings

Place string (String Static Place)	#SSP	Obj-ID, TextStyle-No., x, y, Anchor, "Text"
Change string (String Static Change)	#SSC	Obj-ID, "Text"

Formatted strings

Place formatted string (String Formated Place)	#SFP	Obj-ID, TextStyle-No., x, y, Anchor, "Formatted string"; Value1, Value2, ..., ValueN, Value1,...,ValueN,...
Change parameter of formatted string (String Formated Change)	#SFC	Obj-ID, Value1, Value2, ..., ValueN
Convert string to formatted string (String Formated Format)	#SFF	Obj-ID, "Formatted string"; Value1, Value2, ..., ValueN

Auto update formatted strings

Place formatted string with Auto Update (String Automatic Place)	#SAP	Obj-ID, TextStyle-No., x, y, Anchor, "Formatted string"; (Calculation), Value1....., ValueN
Change calculation from formatted string with Auto Update (String Automatic Change)	#SAC	Obj-ID, (Calculation)
Convert string to formatted string with Auto Update (String Automatic Format)	#SAF	Obj-ID, "Formatted string"; (Calculation), Value1,....., ValueN

Date / time strings

Place the string with the date / time (String Date Place)	#SDP	Obj-ID, TextStyle-No., x, y, Anchor, "Dateformat"; date (act. time)
Change date / time in string (String Date Change)	#SDC	Obj-ID, date (act. time)
Convert string to string with date / time (String Date Format)	#SDF	Obj-ID, "Dateformat"; date (act. time)

EditBox

Place EditBox (String Edit Place)	#SEP	Obj-ID, DrawStyle-No., x, y, Anchor, Width, Height, Radius, TextStyle-No., BorderX(0), BorderY(0)
Define default string for EditBox (String Edit Default)	#SED	Obj-ID, "Default text"; "Default text (Obj-ID+1)"; "Default text (Obj-ID+2)";....
Send strings / codes to EditBox (String Edit Codes)	#SEC	Obj-ID, "String"; "String (Obj-ID+1)"; "String (Obj-ID+2)";....
Connect EditBox with keyboard (String Edit Keyboard)	#SEK	Keyboard-ID, Obj-ID, Obj-ID+1, ...
Activate/deactivate EditBox (String Edit Activate)	#SEA	Obj-ID(0), Keyboard-ID(0)
Define valid character codes	#SER	Obj-ID, Codes

(String Edit Range)		
Define input mask (String Edit Mask)	#SEM	Obj-ID, "Input mask"; Placeholder
Define password mode (String Edit Wildcard)	#SEW	Obj-ID, Wildcardcode

StringBox

Place StringBox (String Box Place)	#SBP	Obj-ID, x, y, Anchor, Width, Height, Radius, ScrollbarWidth(text height)
Define styles for StringBox (String Box Styles)	#SBS	Obj-ID, DrawStyle-No. Background, DrawStyle-No. Scrollbar, TextStyle-No., BorderX(0), BorderY(0), AutoWrap(1)
Add paragraph (String BoxAdd)	#SBA	Obj-ID, Paragraph, "Text"; "Text (Line+1)"; "Text (Line+2)";....
Remove paragraph (String Box Delete)	#SBD	Obj-ID, Paragraph, Line1. ...
Add text file (String Box File)	#SBF	Obj-ID, Paragraph, <Textfile>
Jump to line (String Box Offset)	#SBO	Obj-ID, Line, Time (0), ActionCurve-No (0)

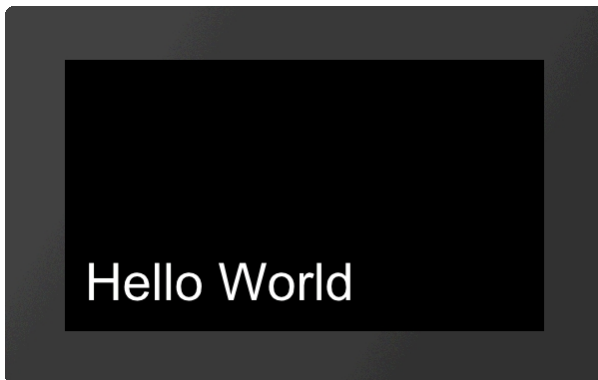
Simple strings

This group includes commands for placing and changing simple strings.

Place string

#SSP	Obj-ID, TextStyle-No., x, y, Anchor, "Text"
-------------	---

The command places a string with the given **Anchor** at the position **x, y**. The appearance of the character string is determined with the TextStyle (**TextStyle-No.**). This is explained in more detail in the section [TextStyle](#).

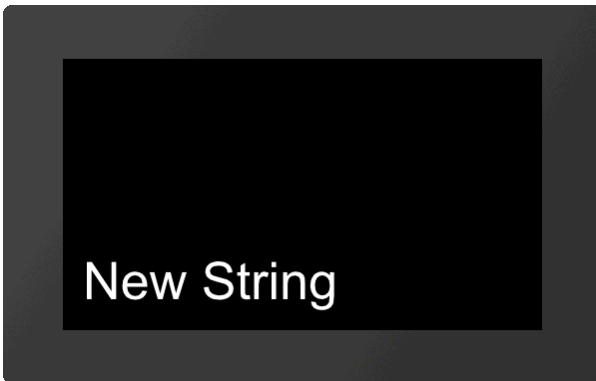


```
...
#SSP 1,1,20,20,7,"Hello World";
...
```

Change string

#SSC	Obj-ID, "Text"
-------------	----------------

The command changes an existing string. Other object properties (position, style, etc.) remain unchanged.



```
...
#SSC 1, "New String";
...
```

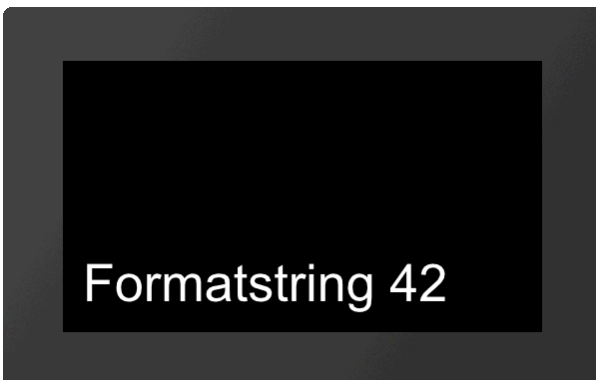
Formatted string

This group contains commands for placing and changing formatted strings.

Place formatted string

#SFP	Obj-ID, TextStyle-No., x, y, Anchor, "Formatted string"; Value1, Value2,, ValueN, Value1,...,ValueN,...
-------------	---

The command places a formatted string with the given **Anchor** at the position **x**, **y**. The appearance of the character string is determined with the TextStyle (**TextStyle-No.**). This is explained in more detail in the subsection [TextStyle](#). If the variable set repeats, the format string is used again. The structure is described in more detail in the subsection [Formatted string](#).

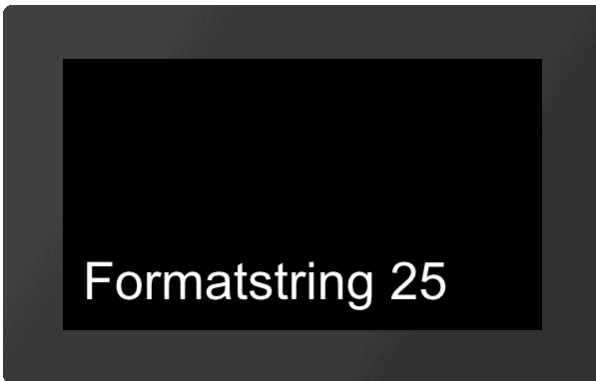


```
...
#SFP 1,1,20,20,7,"Formatstring %d"; 42
...
```

Change parameter of formatted string

#SFC	Obj-ID, Value1, Value2,, ValueN
-------------	---------------------------------------

This command changes the parameters of a formatted string. The object properties (position, style, etc.) remain unchanged.

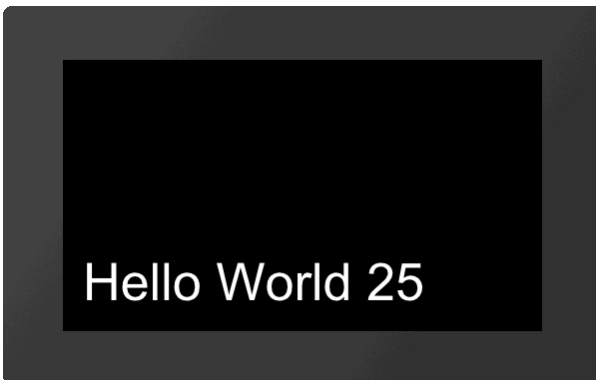


```
...
#SFC 1, 25
...
```

Convert string to formatted string

#SFF	Obj-ID, "Formatted string"; (Calculation), Value1, ..., ValueN
------	--

An existing string is changed to a formatted string. Other object properties (position, style, etc.) remain unchanged. The structure is explained in more detail in the section [Formatted string](#).



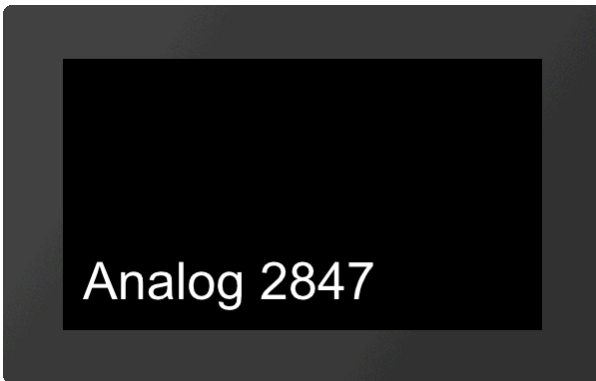
```
...
#SFF 1, "Hello World %d"; 25
...
```

Auto update formatted strings

Place formatted string with Auto Update

#SAP	Obj-ID, TextStyle-No., x, y, Anchor, "Formatted string"; (Calculation), Value1, ..., ValueN
------	---

The command places a formatted string with the given **Anchor** at the position **x**, **y**. The appearance of the character string is determined with the [TextStyle \(TextStyle-No.\)](#). The text-structure is described in more detail in the section [Formatted string](#). The output is renewed as soon as the **Calculation** changes. If the other parameters (value1, ... valueN) are also calculations, their value is recalculated, too (only if the value of the first calculation changes).

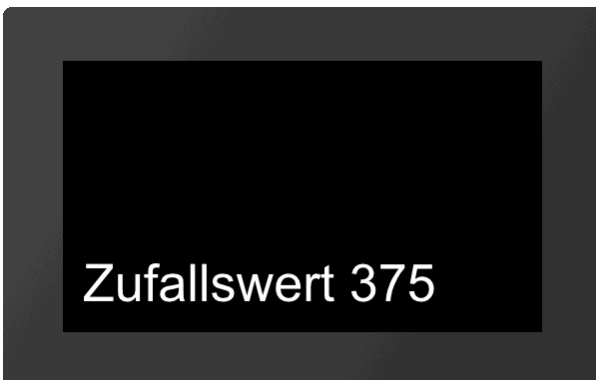


```
...
#SAP 1,1,20,20,7,"Analog %d";(analog(0))
...
```

Change calculation from formatted string with Auto Update

#SAC	Obj-ID, (Calculation)
------	-----------------------

The command changes the calculation of a formatted string with Auto Update. The new calculation only determines the time when the string is output again, without influencing the displayed values / calculation.

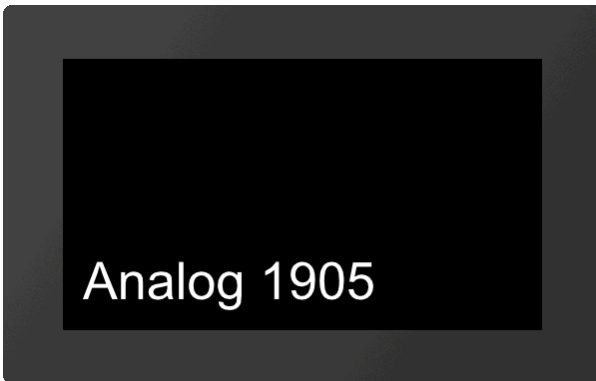


```
...
#SAP 1,1,20,20,7,"Zufallswert %d";
(rand()) /**Ausgabe eines Zufallswertes
#SAC 1,
(time()) /**Änderu
ng des Zufallswertes nur alle Sekunde
...
```

Convert string to formatted string with Auto Update

#SAF	Obj-ID, "Formatted string"; (Calculation), Value1, ..., ValueN
------	--

An existing character string is changed to a formatted character string with Auto Update function. Other object properties (position, style, etc.) remain unchanged. The structure is explained in more detail in the subsection [Formatted string](#). The string renews the output as soon as the **Calculation** changes. If the other parameters (value1, ... valueN) are also calculations, their value is recalculated too (only if the value of the first calculation changes).



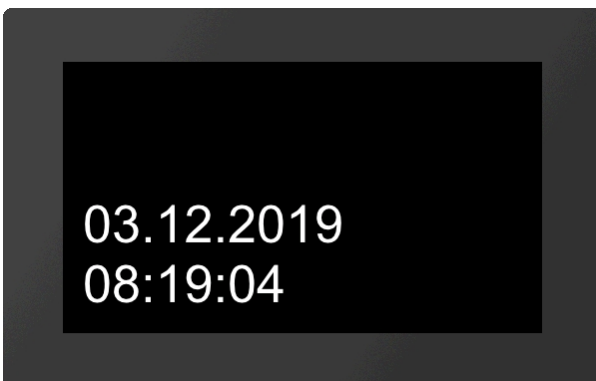
```
...
#SAF 1, "Analog %d"; (analog(0))
...
```

Date / time strings

Place string with date / time

#SDP	Obj-ID, TextStyle-No., x, y, Anchor, "Dateformat"; date (act. time)
------	---

The command places a character string with date / time and the given **Anchor** at the position **x**, **y**. The way of presentation is based on the **date format**. The structure is described in more detail in the sub-chapter [Date formats](#). If the current time is displayed, the output of the current time adapts automatically. The appearance of the character string is determined with the TextStyle (**TextStyle-No.**). This is explained in more detail in the subsection [TextStyle](#).

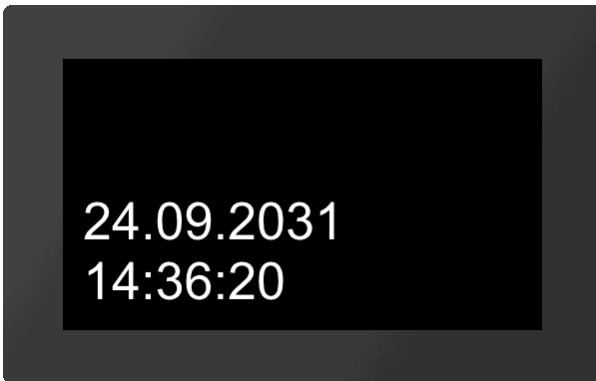


```
...
#SDP 1, 1, 20, 20, 7, "%D.%M.%Y | %h:%m:%s";
...
```

Change date / time in string

#SDC	Obj-ID, date (act. time)
------	--------------------------

The displayed time of the date format is changed. Other object properties (position, style, etc.) remain unchanged. The structure is described in more detail in the sub-chapter [Date formats](#).

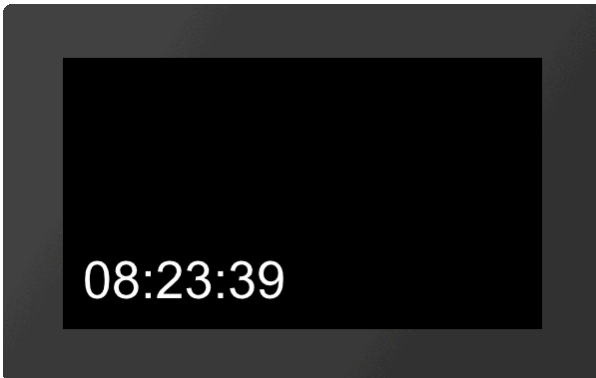


```
...
#SDC 1, (datetime(14,36,20,24,09,2031))
...
```

Convert string to string with date / time

#SDF Obj-ID, "Dateformat"; date (act. time)

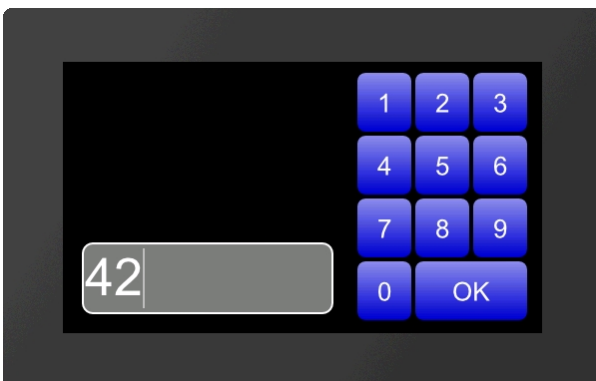
An existing string is changed to a string with date / time. Other object properties (position, style, etc.) remain unchanged. The structure is described in more detail in the sub-chapter [Date formats](#).



```
...
#SDF 1, "%h:%m:%s";
...
```

EditBox

EditBoxes are used for entering characters. The input is usually made using a keyboard. The definition of a keyboard is explained in more detail in the [Keyboard](#) sub-chapter. Entries can also be made by command (see [#SEC](#)). The box must be connected to a keyboard, that entries via keyboard end up in the EditBox (see [#SEK](#)). To receive values the EditBox must be active. This can be done either by command ([#SEA](#)) or by touch ([#TID](#)). In the following example an EditBox is placed, connected to a keyboard and activated by touch. The definition of the keyboard is not included.

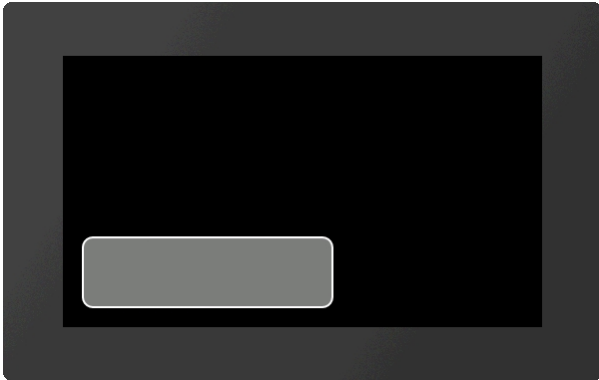


```
...
#SEP 1, 1, 20, 20, 7, 250, 70, 10, 1, 2, 2
#SEK 2, 1
#TID 1, 1
...
```

Place EditBox

#SEP	Obj-ID, DrawStyle-No., x, y, Anchor, Width, Height, Radius, TextStyle-No., BorderX(0), BorderY(0)
-------------	---

The command places an EditBox with the given **Anchor** at position **x, y** with a defined **Width** and **Height**. The DrawStyle defines the appearance of the background of the EditBox (**DrawStyle No.**). The structure is described in more detail in the [DrawStyle](#) subsection. The parameter **Radius** specifies the corner rounding. The appearance of the character string is determined with the TextStyle (**TextStyle-No.**). This is explained in more detail in the [TextStyle](#) subsection. With the two optional parameters (**BorderX** and **BorderY**) the distance of the text to the edge of the box can be specified.

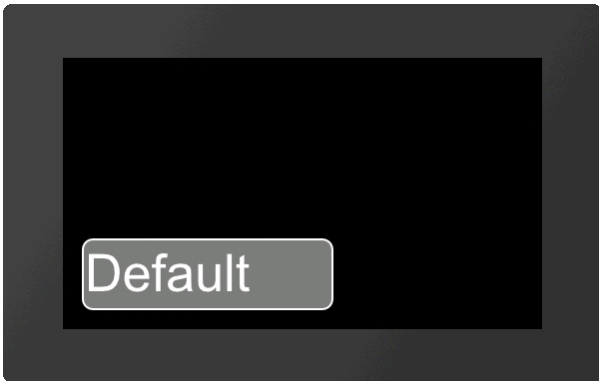


```
...
#SEP 1, 1, 20, 20, 7, 250, 70, 10, 1, 2, 2
...
```

Define default string for EditBox

#SED	Obj-ID, "Default text"; "Default text (Obj-ID+1)"; "Default text (Obj-ID+2)";....
-------------	---

A standard text is defined. Further strings indicate the default string for further EditBoxes with the object IDs Obj-ID+1, ..., Obj-ID+n.

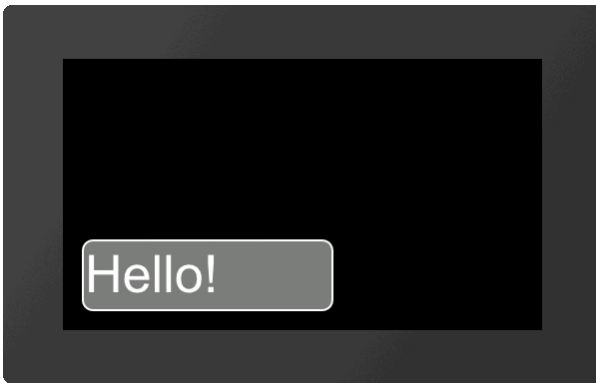


```
...
#SED 1, "Default";
...
```

Send strings / codes to EditBox

#SEC	Obj-ID, "String"; "String (Obj-ID+1)"; "String (Obj-ID+2)";....
-------------	---

The command can be used to send strings and codes to the EditBox. Additional strings are sent to the edit boxes with the object IDs Obj-ID + 1, ..., Obj-ID + n.



```
...
#SEC 1, "Hello"$21;
...
```

Connect EditText with keyboard

#SEK	Keyboard-ID, Obj-ID, Obj-ID+1, ...
------	------------------------------------

This command connects a keyboard (**Keyboard-ID**) with one or more EditTextes (**Obj-ID**)

Activate/deactivate EditText

#SEA	Obj-ID(0), Keyboard-ID(0)
------	---------------------------

The command activates or deactivates EditTextes.

Activate:

Obj-ID	Object ID of the edit box
Keyboard-ID	Not necessary

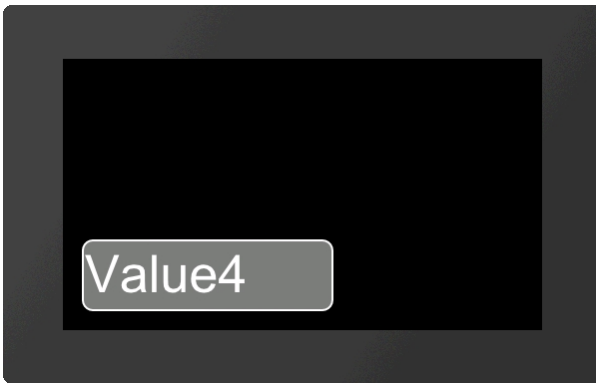
Deactivate:

Obj-ID	0	
Keyboard-ID	0	all edit boxes are deactivated
	Keyboard-ID	all edit boxes assigned to the keyboard are deactivated

Define valid character codes (from V1.2)

#SER	Obj-ID, Codes
------	---------------

The command specifies valid entries that are displayed in the EditText. Valid characters (codes) are separated by commas or specified as a range string (e.g. "0-9A-Za-z", which allows all digits and the Latin alphabet).



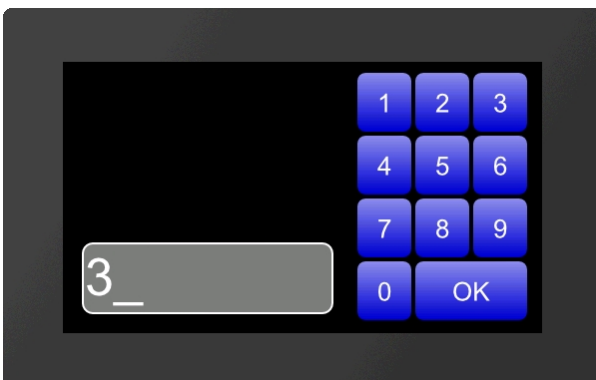
```
...
#SER 1, "A-Za-z,4"
#SEC 1, "Value 42";
...
```

Define input mask (from V1.2)

#SEM Obj-ID, "Input mask"; Placeholder

An input mask is defined for the EditBox. The **Placeholder** parameter defines the visible character code (e.g. '_').
Following masks are possible:

Type	Mask	Example
Integer	%Maximum valueI or %from;toI	"%42I"; or "%10;25I";
Float	%Maximum valueF or %from;toF	"%23.4I"; or "%0.5;7.9I";
ASCII	%character countA	"%4A" (max. 4 characters from the ASCII character set)
Unicode	%character countU	"%4U" (max. 4 characters from the Unicode area)
Range	%character countR	"%4R" (max. 4 characters from the range #SER)

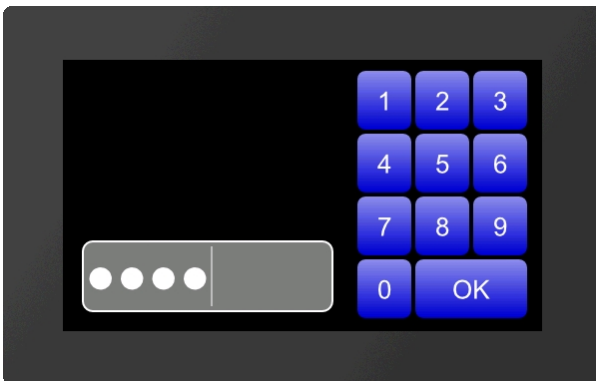


```
...
#SEM 1, "%42I";?_
...
```

Define password mode (from V1.2)

#SEW Obj-ID, Wildcardcode

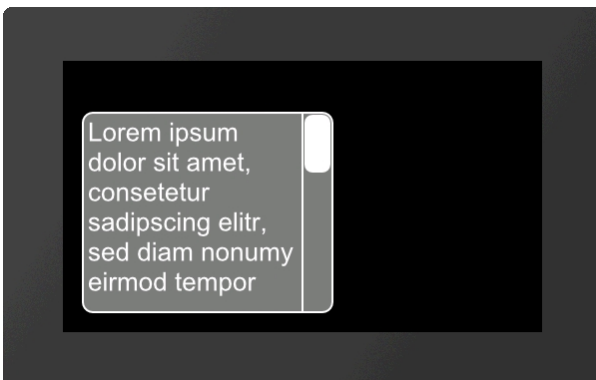
The replacement character (**Wildcardcode**) is displayed instead of the characters entered.



```
...
#SEW 1,$25cf
...
```

StringBox

StringBoxes can display large amounts of text. Additional text can be added or deleted at any time. Every newly added text ([#SBA](#), [#SBE](#)) is inserted as a new paragraph. If the [AutoWrap](#) (see [#SBS](#)) function is deactivated, the paragraph number is the same as the line number. Otherwise, the two can differ. However, there are calculations to convert them into each other. In the following example, a StringBox is created and one paragraph is added.



```
...
#SBP 1,20,20,7,250,200,10
#SBS 1,1,2,4,5,5
#SBA 1,1,"Lorem ipsum dolor ...";
...
```

Place StringBox (from V1.3)

#SBP	Obj-ID, x, y, Anchor, Width, Height, Radius, ScrollbarWidth(text height)
-------------	--

The command places a StringBox with the given **Anchor** at the position **x, y** with a defined **Width** and **Height**. Optionally, the width of the scrollbar can be specified (**ScrollbarWidth**). If no value is specified, the text height is used as the width. It's mandatory to assign a style to the StringBox, otherwise it's invisible (see [#SBS](#)).

Define styles for StringBox (from V1.3)

#SBS	Obj-ID, DrawStyle-No. Background, DrawStyle-No. Scrollbar, TextStyle-No., BorderX(0), BorderY(0), AutoWrap(1)
-------------	---

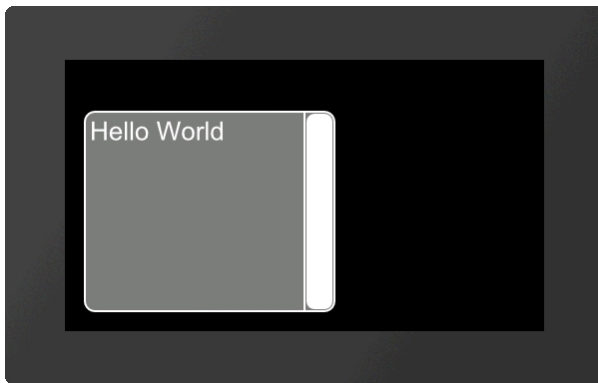
The command defines the appearance of the StringBox. Two DrawStyles are required. On the one hand the background of the EditText and on the other hand the bar of the slider (scrollbar) is defined. The structure is described in more detail in the subsection [DrawStyle](#). The appearance of the character string is determined with the TextStyle (**TextStyle-No.**). This is explained in more detail in the [TextStyle](#) subsection. With the two optional parameters (**BorderX** and **BorderY**) the distance of the text to the edge of the box can be specified. **AutoWrap** determines the line break:

AutoWrap	
0	Text is cut off at the end of line
1	Automatic line break active

Add paragraph (ab V1.3)

#SBA	Obj-ID, Paragraph, "Text"; "Text (Line+1)"; "Text (Line+2)";...
------	---

With the command additional lines can be added to the StringBox. The parameter **Paragraph** specifies the position in the box. The first line has the number 1. If 0 is selected as the paragraph, the text is added at the end.



```
...
#SBA 1,1,"Hello World";
...
```

Remove paragraph (from V1.3)

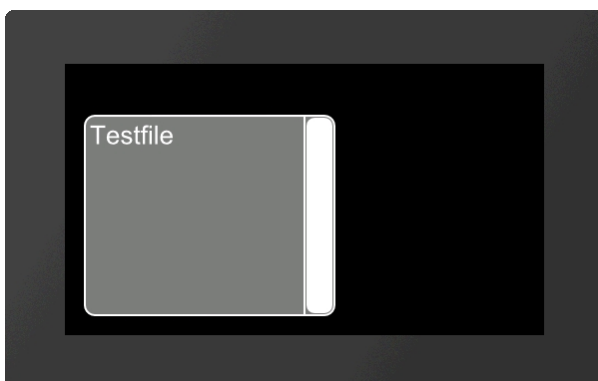
#SBD	Obj-ID, Paragraph, Line1. ...
------	-------------------------------

One or more paragraphs are removed from the StringBox. If 0 is passed as a paragraph, all strings are removed from the StringBox and the box is empty. Areas can also be specified, e.g. 1-5.

Add text file (from V1.3)

#SBF	Obj-ID, Paragraph, <Textfile>
------	-------------------------------

A StringBox can also display complete text files. The parameter <**Textfile**> specifies the path to the file. The parameter **Paragraph** specifies the position in the box. The first line has the number 1. If 0 is selected as the paragraph, the text is added at the end.



```
...
#SBF 1,1,<P:Testfile.txt>
...
```

Jump to line (from V1.3)

#SBO	Obj-ID, Line, Time (0), ActionCurve-No (0)
------	--

The content of the StringBox jumps to the specified line. The jump can be animated with optional parameters. The **Time** parameter is specified in 1/100s. If the value is positive, the duration is used for the entire scroll area. The speed is therefore constant. A negative value determines the time until the new line is reached. So the speed depends on the number of lines to be scrolled. The **ActionCurve-No.** determines the chronological sequence. This is explained in more detail in the sub-chapter [Action Curves and Action Paths](#).

Formatted string

Formatted strings are used in string outputs. The format is based on the C-function "printf". The function has a format specifier and the concrete arguments to be issued. The following specifiers are used in the format string for the various data types:

Type	Placeholder	Example
Fixed-point value decimal	%d	42
Octal value	%o	645
Hex value	%x, %X	7a, 7A
Float	%f,	299.57
Scientific notation	%e, %E	2.9957e+2, 2.9957E+2
Shortest Notation: Float or scientific	%g, %G	299.57
Character	%c	a

Each specifier can be additionally formatted with flags, field width and accuracy, in this order:

Flag	Description
-	Align the value left justified, Right justified is default.
+	Show '+' and '-' depending on the value
(space)	Show ' ' (space) if value is positive, '-' if value is negative.
#	Showing hex and octal values, the 0x, 0X or 0 is shown if the value is $\neq 0$. Float and scientific notation will always have a '.' output - even if only 0 follows. Default the point is only output if values follow
0	Within the field width, left-unnecessary space is filled up with 0.

Field width	Description
(number)	Minimum field width for outputting the value
*	The field width is given by the arguments. The width is directly defined in front of the value in the arguments.

Precision	Description
.number	Integer: Minimum count of digits (default =1) Float: Minimum count of digits after the point (default =6)
.*	Number of digits is taken from the argument list. The number of digits is directly before the actual argument in the list.

Pictures #P

Command group to display pictures. The design software uniTFTDesigner automatically converts the data into the correct internal format. The design software uniTFTDesigner allows to use following filetypes/graphic formats: png, bmp, jpg, jpeg, tga, gif, g16, svg, svgz.

If uniTFTDesigner is not used, those files can be converted by EAconvert.exe (directory \Simulator_and_Tools) (-> evg, epg, epa)

Place picture (Picture Place)	#PPP	Obj-ID, <Name>, y, y, Anchor(1), Width(0), Height (0), Angle (0)
Change animation parameters (Picture change)	#PPA	Obj-ID, AnimationType(0), Time, Image-No.
Load and place image via serial interface (Picture Interface Place)	#PIP	Obj-ID, Binary data; x(0), y(y-Resolution -1),Anchor(1), Width (0), Height(0), Angle (0)

Place picture

#PPP	Obj-ID, <Name>, y, y, Anchor(1), Width(0), Height (0), Angle (0)
-------------	--

With the command, a image (<Name>) with the given **Anchor** is placed at position **x, y**. If **Width = 0** and **Height = 0**, the original size of the image is adopted. If only one of the two parameters is 0, the image is scaled proportionally to the other. The further optional parameter **Angle** specifies the rotation of the image. If an animation is placed, it will be executed cyclically.



```
...
#PPP 1, <P:picture/Logo.epg>, 20, 20, 7, 300
...

...
#PPP 1, "Logo"; 20, 20, 7, 300
...
```

Change animation parameters

#PPA	Obj-ID, AnimationType(0), Time, Image-No.
-------------	---

The command changes an existing image animation. The two parameters **Time** and **Image-No.** are only considered if the **AnimationType** is 7. The animation then runs in the specified time (time in 1 / 100s) up to the picture number. The time between the pictures is recalculated. The following animation types can be selected:

AnimationType	
1	Cyclic
2	Cyclic backward
3	Ping Pong

4	Ping Pong backward
5	Single shot
6	Single shot backwards
7	Goto

Load and place image via serial interface (from V1.3)

#PIP	Obj-ID, Binary data; x(0), y(y-Resolution -1),Anchor(1), Width (0), Height(0), Angle (0)
-------------	--

The command displays an image. For this purpose, the data is transmitted in binary format via the serial interface in *.epg or *.evg format and is placed analogously to the [#PPP](#) command.

Touch functions #T

Command group to enable touch functions. Simple buttons and switches can be used, as well as radio buttons, sliders, bar-graphs and rotary or meter instruments.

Buttons and switches

Place rectangular button (Touch Button Rectangle)	#TBR	Obj-ID, ButtonStyle-No., "Text normal"; "Text down"; x, y, Anchor (5), Width(ButtonStyle Width), Height(ButtonStyle Height)
Place rectangular switch (Touch Switch Rectangle)	#TSR	Obj-ID, ButtonStyle-No., "Text normal"; "Text down"; x, y, Anchor (5), Width(ButtonStyle Width), Height(ButtonStyle Height)
Place elliptical button (Touch Button Ellipse)	#TBE	Obj-ID, ButtonStyle-No., "Text normal"; "Text down"; x, y, Anchor (5), Width(ButtonStyle Width), Height(ButtonStyle Height)
Place elliptical switch (Touch Switch Ellipse)	#TSE	Obj-ID, ButtonStyle-No., "Text normal"; "Text down"; x, y, Anchor (5), Width(ButtonStyle Width), Height(ButtonStyle Height)
Place picture button (Touch Button Picture)	#TBP	Obj-ID, ButtonStyle-No., "Text normal"; "Text down"; x, y, Anchor (5), Width(ButtonStyle Width), Height(ButtonStyle Height)
Place picture switch (Touch Switch Picture)	#TSP	Obj-ID, ButtonStyle-No., "Text normal"; "Text down"; x, y, Anchor (5), Breite(ButtonStyle Width), Height(ButtonStyle Height)
Place label-free icon button (Touch Button Icon)	#TBI	Obj-ID, x, y, Anchor, <Button-name normal>, Width normal (0), 'Button-name down' (no change); Height down (0), "Sound string"
Place label-free icon switch (Touch Switch Icon)	#TSI	Obj-ID, x, y, Anchor, <Button-name normal>, Width normal (0), 'Button-name down' (no change); Height down (0), "Sound string"
Convert object to button (Touch Button Object)	#TBO	Obj-ID, ButtonStyle-No., "Text normal"; "Text down";
Convert object to switch (Touch Switch Object)	#TSO	Obj-ID, ButtonStyle-No., "Text normal"; "Text down";

Settings of buttons and switches

Change labeling of button/switch (Touch Change Label)	#TCL	Obj-ID, "Text normal"; "Text down";
Change state of button/switch (Touch Change State)	#TCS	State, Obj-ID1, ..., Obj-IDn
Query state of button/switch (Touch Query State)	#TQS	Obj-ID1, ..., Obj-IDn
Activate/ deactivate button/switch (Touch Change Enable)	#TCE	Active, Obj-ID1, ..., Obj-IDn
Define feedback from touch events (Touch Change Response)	#TCR	Event, Filter, Obj-ID1, ..., Obj-IDn

Radiogroup

Add button/switch to radio group (Touch Radiogroup Add)	#TRA	Group-ID, Obj-ID1, ..., Obj-IDn
Query state of radio group (Touch Query Radiogroup)	#TQR	Group-ID1, ..., Group-IDn

Special touch functions

Internal touch processing (Touch Id Define)	#TID	Mask, Obj-ID1, ..., Obj-IDn
Place free touch area (Touch Area Free)	#TAF	Obj-ID, x, y, Anchor, Width, Height
Setting gesture times (Touch Configure Gesture)	#TCG	DoubleClick Time (30), LongClick Time (100)

Buttons and switches

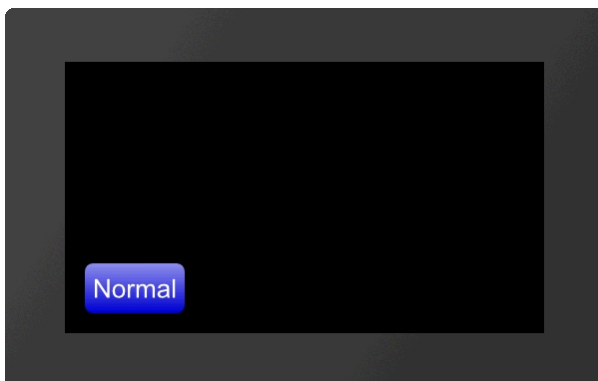
Buttons and switches can react to different events (Down, Up, Drag, DoubleClick, LongClick). There are two ways to evaluate the state changes of buttons and switches:

- **Changes are placed in the send buffer:**
The command [#TCR](#) can be used to specify which responses are placed in the send buffer (no DoubleClick and LongClick)
- **A macro is executed when changes are made:**
The commands [#MDT](#) and [#MDG](#) can be used to connect macros to the button / switch. When the status changes, the associated macro is called.

Place rectangular button/switch

#TBR	Obj-ID, ButtonStyle-No., "Text normal"; "Text down"; x, y, Anchor (5), Width(ButtonStyle Width),
#TSR	Height(ButtonStyle Height)

The command places a rectangular button / switch with the given **Anchor** at the position **x, y**. The parameter **"Text normal"** specifies the output in the unpressed state. **"Text down"** in the pressed state. With the ButtonStyle the appearance of the button / switch is determined (**ButtonStyle No.**). This is explained in more detail in the [ButtonStyle](#) subsection. The **Width** and **Height** of the button / switch is taken from the ButtonStyle, but can optionally be overwritten.



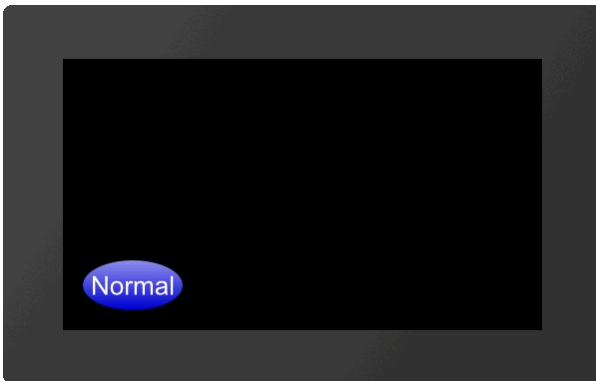
```
...
#TBR 1, 1, "Normal" ; "Pressed" ; 20, 20, 7
...
```

Place elliptical button/switch

#TBE	Obj-ID, ButtonStyle-No., "Text normal"; "Text down"; x, y, Anchor (5), Width(ButtonStyle Width),
#TSE	Height(ButtonStyle Height)

The command places an elliptical button / switch with the given **Anchor** at the position **x, y**. The parameter **"Text normal"** specifies the output in the unpressed state. **"Text down"** in the pressed state. With the ButtonStyle the appearance of the button / switch is determined (**ButtonStyle-No.**). This is explained in more detail in the [ButtonStyle](#)

subsection. The **Width** ($\emptyset X$) and **Height** ($\emptyset Y$) of the button / switch are taken from the ButtonStyle, but can optionally be overwritten.

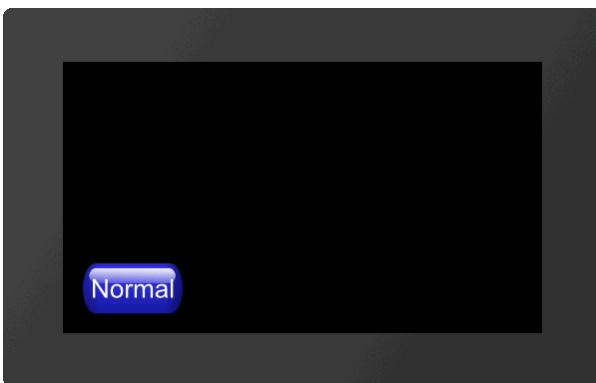


```
...
#TBE 1, 1, "Normal"; "Pressed"; 20, 20, 7
...
```

Place picture button/switch

#TBP	Obj-ID, ButtonStyle-No., "Text normal"; "Text down"; x, y, Anchor (5), Width(ButtonStyle Width),
#TSP	Height(ButtonStyle Height)

The command places a button / switch as an image with the given **Anchor** at position **x, y**. The parameter "**Text normal**" specifies the output in the unpressed state. "**Text down**" in the pressed state. With the ButtonStyle the appearance of the button / switch is determined (**ButtonStyle-No.**). This is explained in more detail in the [ButtonStyle](#) subsection. The **Width** and **Height** of the button / switch is taken from the ButtonStyle, but can optionally be overwritten.

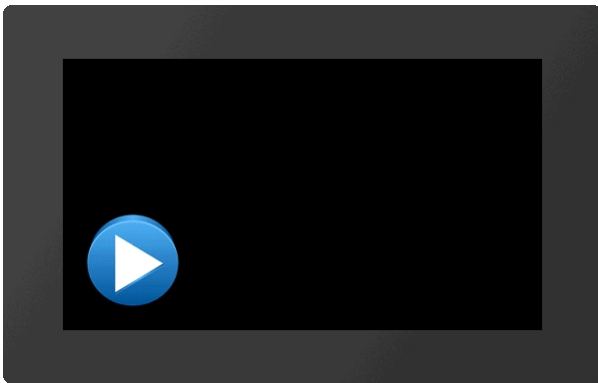


```
...
#TBP 1, 2, "Normal"; "Pressed"; 20, 20, 7
...
```

Place label-free icon button/switch

#TBI	Obj-ID, x, y, Anchor, <Button-name normal>, Width normal (0), 'Button-name down' (keine Änderung);
#TSI	Height down (0), "Sound string"

The command places a button / switch as an icon with the given **Anchor** at position **x, y**. A ButtonStyle is not necessary for this. The two parameters **<Buttonname normal>** and **<Buttonname down>** specify the images to be displayed. If no **Width** (in pixels) or zero is specified, the original size of the image is used. The height is calculated internally (proportional). The last parameter "**Sound string**" specifies the [notes](#) that are played when touched.



```

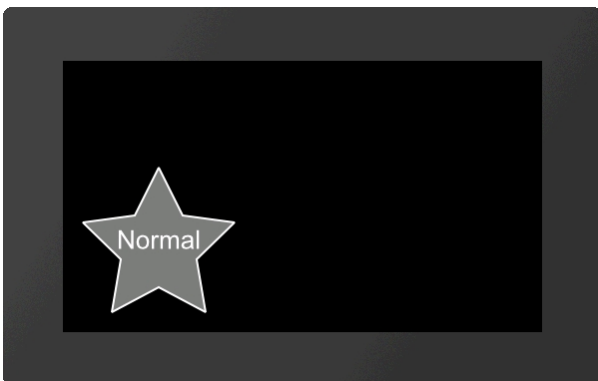
...
#TBI
1
, 20
, 20
,
,
7
, <P:button/Play.epg>
, 100, <P:button/Pause.epg>, 100
...
...
#TBI 1, 20, 20, 7, "Play"; 100, "Pause"; 100
...

```

Convert object to button/switch

#TBO	Obj-ID, ButtonStyle-No., "Text normal"; "Text down";
#TSO	

Any existing object is converted into a button / switch. The [ButtonStyle](#) provides additional information (e.g. Sound name). If the object is a graphic primitive (e.g. polygon) with the same DrawStyle as in the ButtonStyle, the DrawStyle of the ButtonStyle is automatically adopted for the pressed state.



```

...
#TBO 1, 1, "Normal"; "Pressed";
...

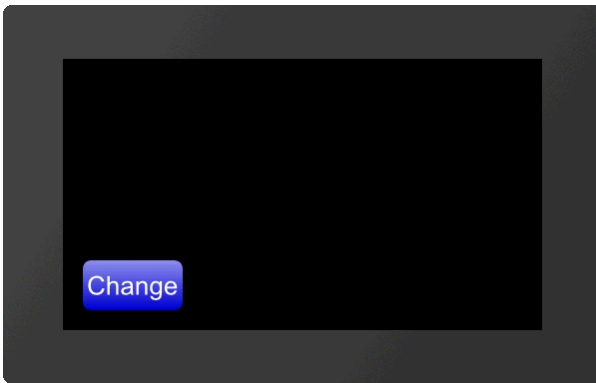
```

Settings of buttons and switches

Change labeling of button/switch

#TCL	Obj-ID, "Text normal"; "Text down";
------	-------------------------------------

The command changes the labeling of touch objects. If no text is specified for the pressed state ("Text down"), "Text normal" is used for both.



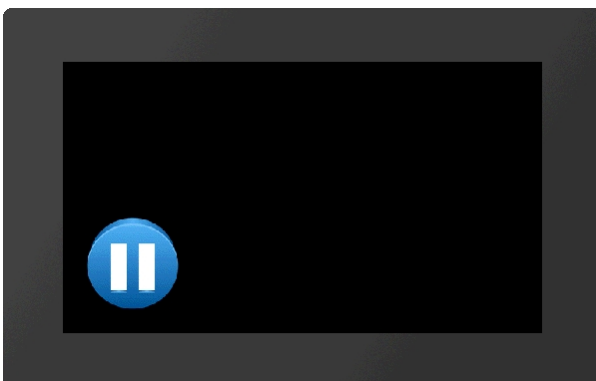
```
...
#TBR 1, ...
#TCL 1, "Change";
...
```

Change state of button/switch

#TCS State, Obj-ID1, ..., Obj-IDn

The command changes the state of the touch objects (**Obj-ID1**, ..., **Obj-IDn**):

State	
1	unpressed
2	pressed



```
...
#TBI
1
, 20
, 20
,
7
, <P:button/Play.epg>
, 100, <P:button/Pause.epg>, 100
#TCS 2, 1
...
```

Query state of button/switch

TQS Obj-ID1, ..., Obj-IDn

The state of the touch objects (**Obj-ID1**, ..., **Obj-IDn**) is placed in the [send buffer](#). The feedback has the following structure:

#	T	Q	S	Obj-ID	State	...
\$1B	\$54	\$51	\$53	16-Bit value	16-Bit value	

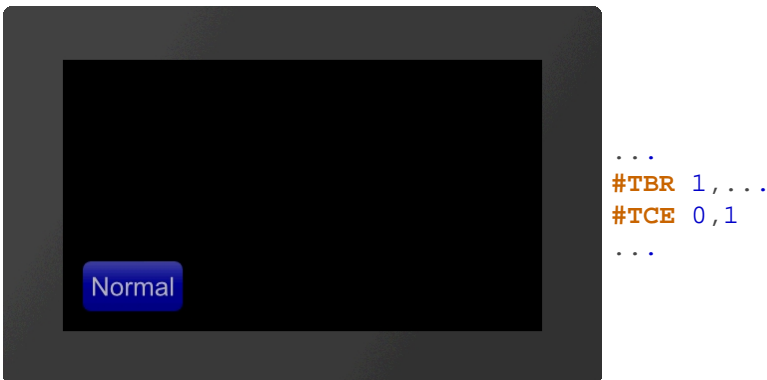
```
1Bh 54h 51h 53 h 01h 00h 01h 00h ...
#TBR 1, ...
#TQS 1
...
```

Activate/ deactivate button/switch

#TCE	Active, Obj-ID1, ..., Obj-IDn
------	-------------------------------

The command activates or deactivates touch objects (**Obj-ID1, ..., Obj-IDn**) (**Active**):

Active	
0	not active
1	active



Define feedback from touch events

#TCR	Event, Filter, Obj-ID1, ..., Obj-IDn
------	--------------------------------------

Each touch object can be assigned whether and which feedback is sent to the [send buffer](#). A distinction is made between three events: up, down and drag. For each of these events it can be set individually whether it triggers a feedback or not. This can be set with the **Event** parameter. The event is bit coded according to the following table:

	Event							
	0	1	2	3	4	5	6	7
Up								
Down								
Drag								

If you only want to receive feedback for up and down events, set the Event parameter to 3. In addition, the feedback can also be made dependent on a defined macro for the event (**Filter**):

Filter	
0	only send if no macro is defined
1	always send

Default settings	
Button	#TCR 2,0,Obj-ID
Switch	#TCR 3,0,Obj-ID
Bargraph/Instrument	#TCR 1,0,Obj-ID
Editbox	#TCR 1,0,Obj-ID

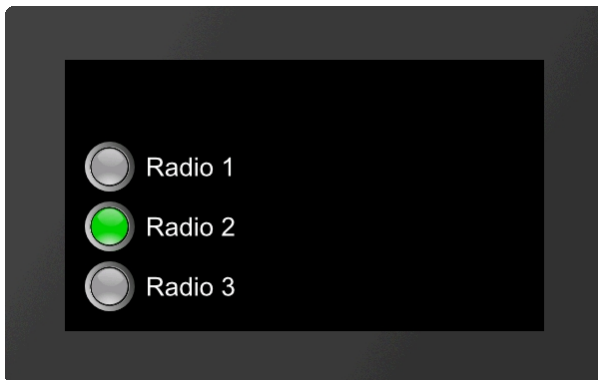
If no events should be sent, set `#TCR 0,0,Obj-ID`.

Radiogroup

Add button/switch to radio group

#TRA	Group-ID, Obj-ID1, ..., Obj-IDn
------	---------------------------------

One or more switches (**Obj-ID**, ..., **Obj-IDn**) are added to an [existing](#) or new radio group (**Group-ID**).



```
...
#TSP 1,2,"Radio 1";"Radio 1";20,140,7
#TSP 2,2,"Radio 2";"Radio 2";20,80,7
#TSP 3,2,"Radio 3";"Radio 3";20,20,7
#TRA 4,1,2,3
...
```

Query state of radio group

#TQR	Group-ID1, ..., Group-IDn
------	---------------------------

The active switch of the radio group (**Group-ID**) is placed in the [send buffer](#). The feedback is structured as follows:

ESC	T	Q	R	Obj-ID	Group-ID	...
-----	---	---	---	--------	----------	-----

\$1B	\$54	\$51	\$52	16-Bit value	16-Bit value	
------	------	------	------	--------------	--------------	--

```
1Bh 54h 51h 52h 03h 00h 04h 00h ...
#TRA 4, ...
#TQR 4
...
```

Special touch functions

Internal touch processing

#TID	Mask, Obj-ID1, ..., Obj-IDn
------	-----------------------------

A special touch action can be assigned to each object (**Obj-ID**) or touch input can be enabled. The individual bits of the **Mask** can be combined with bit decoding, that multiple touch actions are possible at the same time:

Mask	
1	Internal processing (e.g. Bargraphs / Instruments / EditBoxes)
2	Move (object can be moved by touching and dragging)
4	Object can be enlarged/reduced proportionally
8	Object can be rotated around the active anchor
16	Size change with two fingers
32	Rotate with two fingers
128	Object remains unchanged, touch macros are executed

Place free touch area (from V1.4)

#TAF	Obj-ID, x, y, Anchor, Width, Height
------	-------------------------------------

The command places a free touch area with the given **Anchor**, **Width** and **Height** at the position **x, y**.

Setting gesture times (from V1.4)

#TCG	DoubleClick Time, LongClick Time
------	----------------------------------

The command sets the time threshold of gestures. The **DoubleClick Time** specifies in 1 / 100s the maximum time that can pass between two down events, so that a valid double click is still recognized. The parameter **LongClick Time** determines which time span (in 1 / 100s) must elapse at least for a LongClick to be detected. The valid range for DoubleClick is 20 (=200 ms) to 100 (=1 sec.), for LongClick it is valid for 30 (=300 ms) to 1000 (=10 sec.).

Draw / graphic primitives #G

Command group to show graphical simple objects:

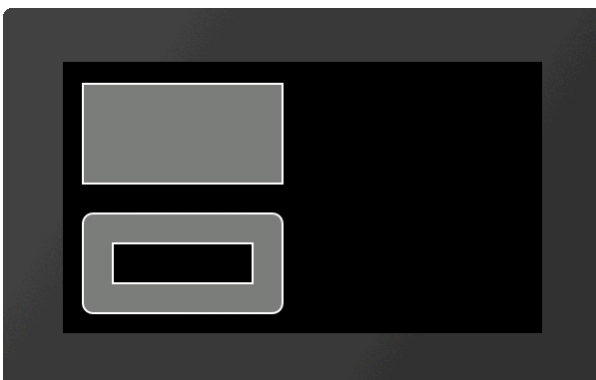
Place rectangle (Graphic Rounded Rectangle)	#GRR	Obj-ID, DrawStyle-No, x, y, Anchor, Width, Height(=Width), Radius (0), FrameThickness(0), Angle(0)
Place regular polygon (Graphic Geometric Polygon)	#GGP	Obj-ID, DrawStyle-No, x, y, Anchor, Radius, Corners, FrameThickness(0), Angle(0)
Place star (Graphic Geometric Star)	#GGS	Obj-ID, DrawStyle-No, x, y, Anchor, Radius1, Radius2, Tip, FrameThickness(0), Angle(0)
Place circle/ellipse (Graphic Ellipse Total)	#GET	Obj-ID, DrawStyle-No, x, y, Anchor, RadiusX, RadiusY(=RadiusX), FrameThickness(0), Angle(0)
Place circular sector/piece of cake (Graphic Ellipse Pie)	#GEP	Obj-ID, DrawStyle-No, x, y, Anchor, RadiusX, RadiusY, StartAngle, EndAngle, Angle(0)
Place circular segment (Graphic Ellipse Segment)	#GES	Obj-ID, DrawStyle-No, x, y, Anchor, RadiusX, RadiusY, StartAngle, EndAngle, WiAnglekel(0)
Place arc (Graphic Ellipse Arc)	#GEA	Obj-ID, DrawStyle-No, x, y, Anchor, RadiusX, RadiusY, StartAngle, EndAngle, Border(0), Angle(0)
Place polyline (Graphic Polygon Line)	#GPL	Obj-ID, DrawStyle-No, x1, y1, x2, y2, ... xn, yn
Place irregular polygon (Graphic Polygon Fill)	#GPF	Obj-ID, DrawStyle-No, x1, y1, x2, y2, ... xn4 y4
Add points to polyline (Graphic Polyline Add)	#GPA	Obj-ID, x1, y1, x2, y2, ... xn, yn

Geometrical figures

Place rectangle

#GRR	Obj-ID, DrawStyle-No, x, y, Anchor, Width, Height(=Width), Radius (0), FrameThickness(0), Angle(0)
-------------	--

The command places a rectangle with the **Anchor** and the **Width** at the position **x, y**. The appearance of the rectangle is determined with the DrawStyle (DrawStyle-No.). This is explained in more detail in the [DrawStyle](#) subsection. If no **Height** is specified, the height is set to the width (square). A **Radius** can optionally be specified. This rounds off the corners. It is also possible to determine a **FrameThickness**. Rotation around the anchor (**Angle**) can also be set.

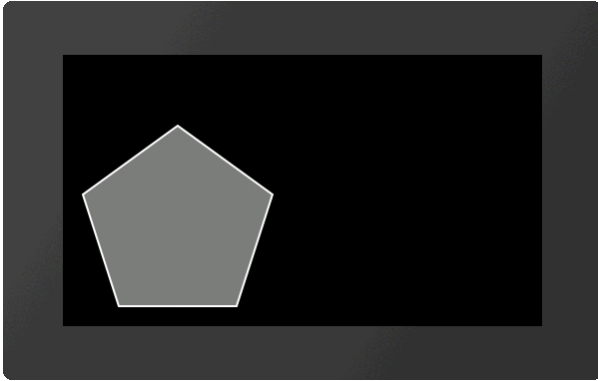


```
...
#GRR 1,1,20,150,7,200,100
#GRR 2,1,20,20,7,200,100,10,30
...
```

Place regular polygon

#GGP Obj-ID, DrawStyle-No, x, y, Anchor, Radius, Corners, FrameThickness(0), Angle(0)

The command places a regular polygon with the **Anchor** and the given number of **Corners** at the position **x, y**. With the DrawStyle the appearance of the n-corner is determined (**DrawStyle-No.**). This is explained in more detail in the [DrawStyle](#) subsection. The **Radius** determines the size of the figure. It is also possible to determine a **FrameThickness**. Rotation around the anchor (**Angle**) can also be set. If anchor = 0, the construction point is used.



...
#GGP 1, 1, 20, 20, 7, 100, 5
...

Place star

#GGS Obj-ID, DrawStyle-No, x, y, Anchor, Radius1, Radius2, Tip, FrameThickness(0), Angle(0)

The command places a star with the **Anchor** at the position **x, y**. The appearance of the star is determined with the DrawStyle (**DrawStyle-No.**). This is explained in more detail in the [DrawStyle](#) subsection. The first tip is set to **Radius1** above the center point. Then the connection to Radius2 is made then back to Radius1 etc. until the number of **Tips** is reached. It is also possible to determine a **FrameThickness**. Rotation around the anchor (**Angle**) can also be set. If anchor = 0, the construction point is used.

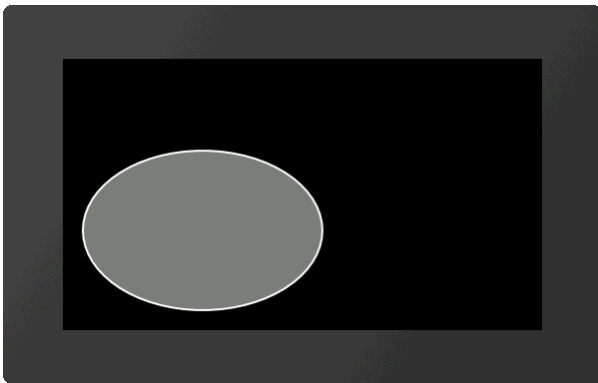


...
#GGS 1, 1, 20, 20, 7, 100, 50, 7
...

Place circle/ellipse

#GET Obj-ID, DrawStyle-No, x, y, Anchor, RadiusX, RadiusY(=RadiusX), FrameThickness(0), Winkel(0)

The command places an ellipse with the **Anchor** and the **RadiusX** at the position **x, y**. The appearance of the circle is determined with the DrawStyle (**DrawStyle-No.**). This is explained in more detail in the [DrawStyle](#) subsection. If no **RadiusY** is specified, it is set to RadiusX (circle). It is also possible to determine a **FrameThickness**. Rotation around the anchor (**Angle**) can also be set.



```
...
#GET 1,1,20,20,7,120,80
...
```

Place circular sector/piece of cake

#GEP	Obj-ID, DrawStyle-No, x, y, Anchor, RadiusX, RadiusY, StartAngle, EndAngle, Winkel(0)
------	---

The command places a circle sector / piece of cake with the **Anchor**, the **RadiusX** and the **RadiusY** at the position **x**, **y**. **Start-/EndAngles** indicate the size of the piece. With the DrawStyle the appearance of the circle sector is determined (**DrawStyle-No.**). This is explained in more detail in the [DrawStyle](#) subsection. If no RadiusY is specified, it is set to RadiusX (circle). Rotation around the anchor (**Angle**) can also be set.



```
...
#GEP 1,1,20,20,7,100,100,20,250
...
```

Place circular segment

#GES	Obj-ID, DrawStyle-No, x, y, Anchor, RadiusX, RadiusY, StartAngle, EndAngle, Winkel(0)
------	---

The command places a segment of a circle with the **Anchor**, the **RadiusX** and the **RadiusY** at the position **x**, **y**. **Start-/ EndAngles** indicate the size of the piece. The appearance of the circle segment is determined with the DrawStyle (**DrawStyle-No.**). This is explained in more detail in the [DrawStyle](#) subsection. If no **RadiusY** is specified, it is set to RadiusX (circle). Rotation around the anchor (**Angle**) can also be set.



```
...
#GES 1,1,20,20,7,100,100,20,250
...
```

Place arc

#GEA	Obj-ID, DrawStyle-No, x, y, Anchor, RadiusX, RadiusY, StartAngle, EndAngle, Border(0), Angle(0)
------	---

The command places an arc with the **Anchor**, the **RadiusX** and the **RadiusY** at the position **x, y**. **Start-/ EndAngles** indicate the size of the piece. The appearance of the circular arc is determined with the DrawStyle (**DrawStyle-No.**). This is explained in more detail in the [DrawStyle](#) subsection. If no **RadiusY** is specified, it is set to RadiusX (circle). It is also possible to determine a frame thickness. Rotation around the anchor (**Angle**) can also be set.

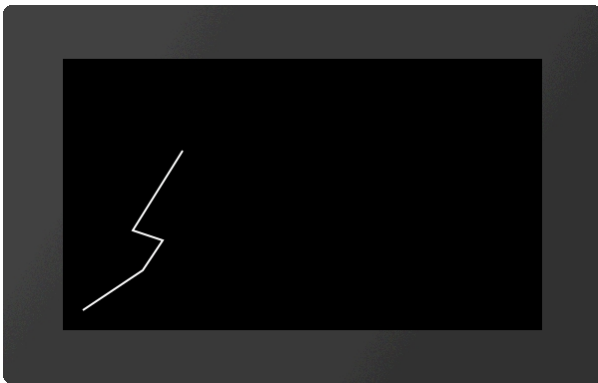


```
...
#GEA 1,1,20,20,7,100,100,20,250
...
```

Place polyline

#GPL	Obj-ID, DrawStyle-No, x1, y1, x2, y2, ... xn, yn
------	--

The command draws a polyline with the coordinates **[x1, y1], [x2, y2], ..., [xn, yn]**. The appearance of the polyline is determined with the DrawStyle (**DrawStyle-No.**). This is explained in more detail in the [DrawStyle](#) subsection.



```
...
#GPL 1,1,20,20,80,60,100,90,70,100,120,180
...
```

Place irregular polygon

#GPF Obj-ID, DrawStyle-No, x1, y1, x2, y2, ... x4, y4

The command draws a filled polygon with the coordinates **[x1, y1], [x2, y2], ..., [x4, y4]**. The appearance of the polygon is determined with the DrawStyle (**DrawStyle-No.**). This is explained in more detail in the [DrawStyle](#) subsection. From the last given point the shape is automatically closed. Only 2-4 points are allowed, meaning you can draw lines, triangle and quadrangle.

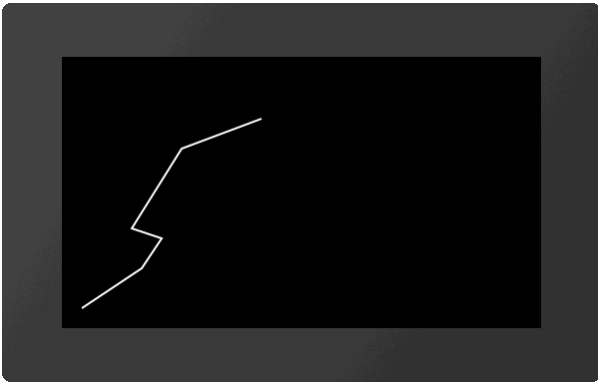


```
...
#GPF 1,1,20,20,100,100,120,180, 40,150,40,100
...
```

Add points to polyline

#GPA Obj-ID, x1, y1, x2, y2, ... xn, yn

The command adds coordinates **[x1, y1], [x2, y2], ..., [xn, yn]** at the end of a polyline. In the case of a polygon, the figure is closed automatically.



...
#GPL 1, 1, 20, 20, 80, 60, 100, 90, 70, 100, 120, 180
#GPA 1, 200, 210
...

Bargraph / instruments #I

Command group to show bar graphs, sliders and rotary / pointer instruments

Bargraph

Place rectangular Bargraph (Instrument Bar Rectangle)	#IBR	Obj-ID, DrawStyle-Filling, DrawStyle-Background, x, y, Anchor, Width, Height, Radius(0), StartValue(0), EndValue(100), Direction (1), Angle(0)
Place triangular Bargraph (Instrument Bar Triangle)	#IBT	Obj-ID, DrawStyle-Filling, DrawStyle-Background, x, y, Anchor, Width, Height, Tip(0), StartValue(0), EndValue(100), Direction(1), Angle(0)
Place curved Bargraph (Instrument Bar Arc)	#IBA	Obj-ID, DrawStyle-Filling, DrawStyle-Background, x, y, Anchor, Radius, Thickness, StartAngle, EndAngle, StartValue(0), EndValue(100), Direction (1)

Slider

Define Slider along path (Instrument Group Slider)	#IGS	Group-ID, Path-ID, Controller-ID, Tangential(0), StartValue(0), EndValue(100)
--	-------------	---

Rotary / pointer instruments

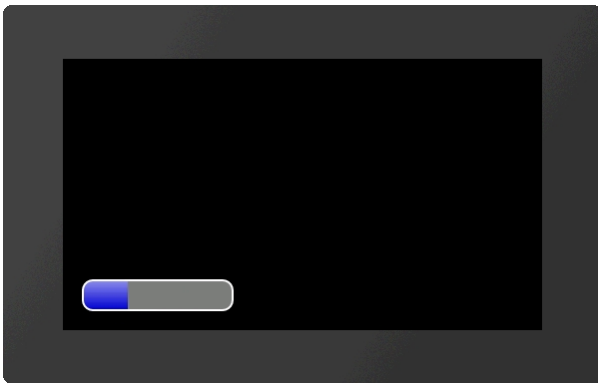
Define pointer instrument from objects (Instrument Group Meter)	#IGM	Group-ID, Indicator-ID, StartAngle, DeltaAngle, StartValue(0), EndValue(100)
Place a instrument from the eDIP series (Instrument Picture Place)	#IPP	Object-ID, 'InstrumentName'; x, y, Anchor, Width(0), Height(0), StartValue(0), EndValue(100), Angle(0)

Setting Bargraph / instrument

Set value of Bargraph/Instrument (Instrument Value Set)	#IVS	Obj-ID, Value, Time(0), ActionCurve-No.(0)
Change Start-/End-value of a Bargraph/Instrument (Instrument Value New)	#IVN	Obj-ID, StartValue, EndValue(no change)
Set Bargraph/Instrument value to calculation value with AutoUpdate (Instrument Value Autochange)	#IVA	Obj-ID, (Calculation), Time(0), ActionCurve-No.(0)
Change Bargraph/Instrument calculation for AutoUpdate (Instrument Value Calculation)	#IVC	Obj-ID, (Calculation)

Bargraph

Bargraphs can be used for displaying or for entering values. After the definition ([#IBR](#), [#IBT](#), [#IBA](#)), the Bargraph can neither be operated by touch nor does it display a predefined value. To activate it for touch input, you need the command [#TID](#). Values can be set with the [#IVS](#) command. The [#IVA](#) function is required if the Bargraph should change automatically when changing a calculation value (e.g. analog input, register value, ...). In the following example, a rectangular Bargraph is placed, pre-assigned to the value 30 and activated by touch.



```
...
#IBR 1, 2, 1, 20, 20, 7, 150, 30, 10
#IVS 1, 30
#TID 1, 1
...
```

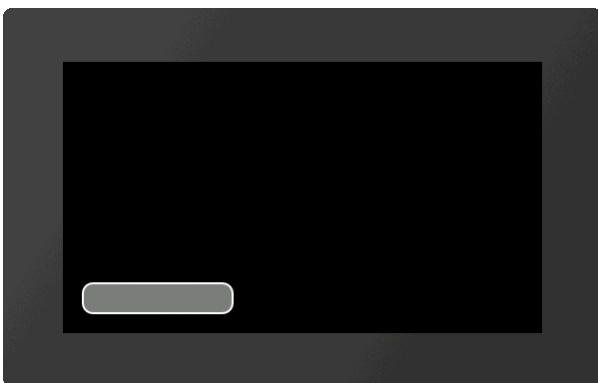
Place rectangular Bargraph

#IBR	Obj-ID, DrawStyle-Filling, DrawStyle-Background, x, y, Anchor, Width, Height, Radius(0), StartValue(0), EndValue(100), Direction (1), Angle(0)
------	--

The command places a rectangular Bargraph with the **Anchor**, **Width** and **Height** at the position **x**, **y**. The filling color is taken from the **DrawStyle-Filling**. The **DrawStyle-Background** specifies the background and frame color. The structure of the [DrawStyle](#) sub-chapter is explained in more detail. A **Radius** can optionally be specified. This rounds off the corners. The **StartValue** and **EndValue** determine the two limits of the Bargraph. The running direction is determined by **Direction**:

Direction	
0	Right to left
1	Left to right

Rotation around the anchor (**Angle**) can also be set.



```
...
#IBR 1, 2, 1, 20, 20, 7, 150, 30, 10
...
```

Place triangular Bargraph

#IBT	Obj-ID, DrawStyle-Filling, DrawStyle-Background, x, y, Anchor, Width, Height, Tip(0), StartValue(0), EndValue(100), Direction(1), Angle(0)
------	--

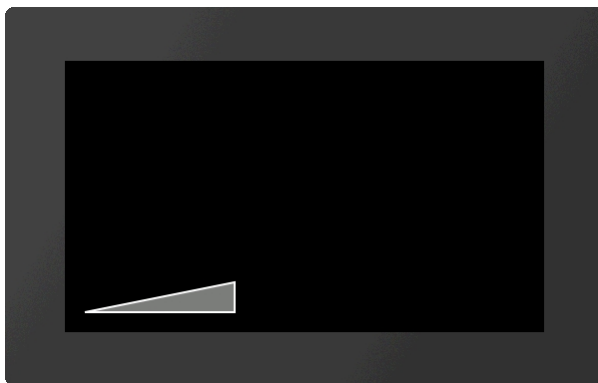
The command places a triangular Bargraph with the **Anchor**, **Width** and **Height** at the position **x**, **y**. The filling color is taken from the **DrawStyle-Filling**. The **DrawStyle-Background** specifies the background and frame color. The structure of the [DrawStyle](#) sub-chapter is explained in more detail. The tip is on the left side. The optional parameter **Tip** specifies the position of the tip:

Tip	
0	Bottom
1	Top
2	Middle

StartValue and **EndValue** determine the two limits of the Bargraph. The running direction is determined by **Direction**:

Direction	
0	Towards tip
1	Away from tip

Rotation around the anchor (**Angle**) can also be set.



```
...
#IBT 1,2,1,20,20,7,150,30
...
```

Place curved Bargraph

#IBA	Obj-ID, DrawStyle-Filling, DrawStyle-Background, x, y, Anchor, Radius, Thickness, StartAngle, EndAngle, StartValue(0), EndValue(100), Direction (1)
-------------	---

The command places a curved Bargraph with the **Anchor** and given **Thickness** at the position **x, y**. The size is determined by the parameters **Radius**, **StartAngle** and **EndAngle**. The filling color is taken from the **DrawStyle-Filling**. The **DrawStyle-Background** specifies the background and frame color. The structure of the [DrawStyle](#) sub-chapter is explained in more detail. The **StartValue** and **EndValue** determine the two limits of the Bargraph. The running direction is determined by **Direction**:

Direction	
0	Counterclockwise
1	Clockwise

Rotation around the anchor (**Angle**) can also be set.

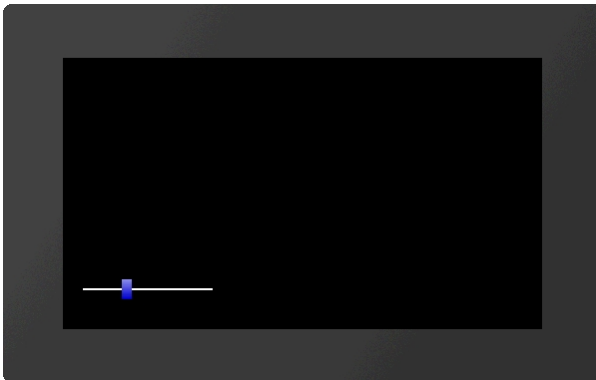


```
...
#IBA 1,2,1,20,20,7,100,40,0,180
...
```

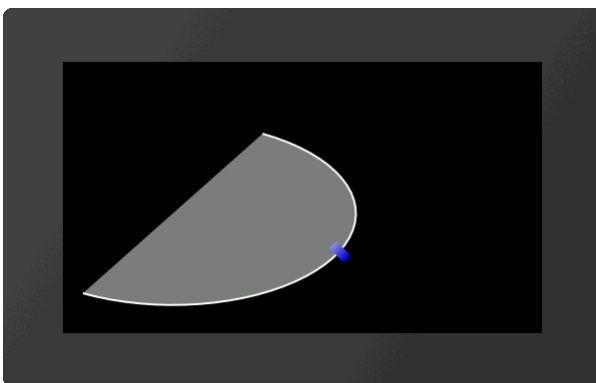
Slider

A slider consists of a path ([#GPL](#), [#GPP](#)) and a knob (e.g. [#GRR](#), [#PPP](#), ...). Both objects have to be defined in advance and grouped together ([#OGA](#)). The start position of the controller (value 0) coincides with the construction point of the path. Nevertheless, it makes sense to position the controller in the right place, since the group limitation (bounding box) does not adapt automatically.

To activate the slider for touch input, you need the command [#TID](#). Values can be set with the [#IVS](#) command.



```
...
#GPL 1,1,20,40,150,40
#GRR 2,2,20,40,4,10,20
#OGA 3,1,2
#IGS 3,1,2
#IVS 3,30
#TID 1,3
...
```



```
...
#GPP 1,1,20,40,?E0,100,50,0,200,200
#GRR 2,2,20,40,4,10,20
#OGA 3,1,2
#IGS 3,1,2,1
#IVS 3,60
#TID 1,3
...
```

Define slider along path

#IGS	Group-ID, Path-ID, Controller-ID, Tangential(0), StartValue(0), EndValue(100)
-------------	---

The command converts an existing group **Group-ID** into a slider. The group must contain at least two existing objects:

- A path ([#GPL](#), [#GPP](#)) **Path-ID**

- A controller (z.B. [#GRR](#), [#PPP](#), ...) **Regler-ID**

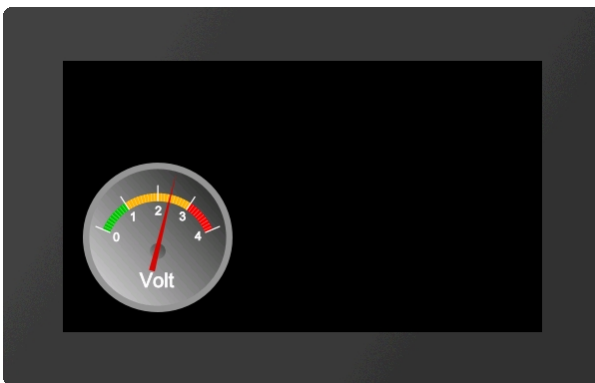
The controller moves along the path:

Tangential	
0	Controller does not turn
1	Controller rotates tangentially to the path

StartValue and **EndValue** determine the two limits of the slider.

Rotary / pointer instruments

A rotary / pointer instrument consists of a background (scale) and a pointer. Both objects have to be defined in advance and grouped together ([#OGA](#)). When designing the pointer, make sure that it is positioned in the zero position of the scale. To activate the instrument for touch inputs, the command [#IID](#) is required. Values can be set with the [#IVS](#) command.



```

...
#PPP
1,<P:picture/Voltmeter.epg>,20,20,7,150,150,0
#PPP
2
,<P:picture/Voltmeter_Needle.epg>
,66,91,5,6,100,70
#OAO 3,20,2
#OAA 0,2
#OGA 3,1,2
#IGM 3,2,160,-140,0,100
#IVS 3,60
#IID 1,3
...

```

Alternatively, instruments from the eDIP series can also be placed directly (see [#IPP](#))

Define pointer instrument from objects

#IGM	Group-ID, Indicator-ID, StartAngle, DeltaAngle, StartValue(0), EndValue(100)
-------------	--

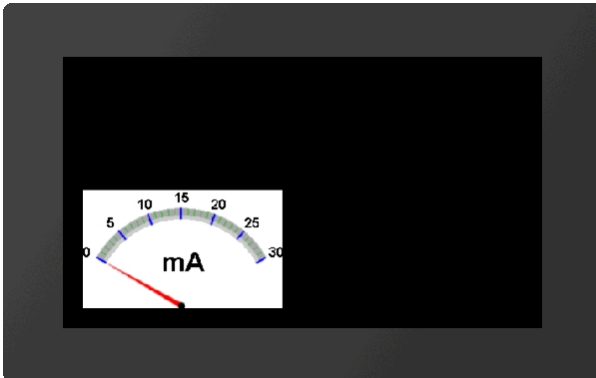
The command converts an existing group **Group-ID** into a pointer instrument. The parameter **Indicator-ID** determines the pointer. The pointer must be positioned in a 90° position (upwards). The **StartAngle** parameter specifies the start angle of the scale. The **DeltaAngle** determines the direction of rotation (positive: counterclockwise; negative: clockwise) and the total angle of rotation (from the start angle). Optionally, the start and end values (input and output values) can also be specified.

Place an instrument from the eDIP series

#IPP	Object-ID, 'InstrumentName'; x, y, Anchor, Width(0), Height(0), StartValue(0), EndValue(100), Angle(0)
-------------	--

This command is available for compatibility reasons. Finished instruments from the eDIP series, which were created with the LCDTools, can be displayed directly. The instrument must be converted to the *.epi format beforehand using

EAconvert.exe. The size is determined by the parameters **Width** and **Height**. If width = 0 and height = 0, the original size of the instrument is adopted. If only one of the two parameters is 0, the instrument is adjusted proportionally. Optionally, the start and end values (input and output values) and the rotation around the anchor (**Angle**) can be specified.



```
...
#IPP
1,<P:instrument/instrument.epi>,20,20,7,200
...

...
#IPP 1,"instrument";20,20,7,200
...
```

Setting Bargraph/Instrument

Set value of Bargraph/Instrument

#IVS	Obj-ID, Value, Time(0), ActionCurve-No.(0)
------	--

With the command, a bar graph or instrument is set to a new **Value**. The **Time** parameter is specified in 1/100s. If the value is positive, the time period is used for the total deflection, meaning a constant speed. A negative value determines the time until the new value is reached. The speed is depending on the distance. The **ActionCurve-No.** determines the chronological sequence. This is explained in more detail in the subchapter [Action Curves and Action Paths](#).

Change Start-/End-value of a Bargraph/Instrument

#IVN	Obj-ID, StartValue, EndValue(no change)
------	---

The command assigns a new start and / or end value to the Bargraph or instrument.

Set Bargraph/Instrument value to a calculation with AutoUpdate

#IVA	Obj-ID, (Calculation), Time(0), ActionCurve-No.(0)
------	--

The value of a Bargraph or instrument is automatically calculated using the **Calculation** (e.g. analog input, register, calculation) and always changed when its value changes. The **Time** parameter is specified in 1/100s. If the value is positive, the time period is used for the total deflection, meaning a constant speed. A negative value determines the time until the new value is reached. The speed is depending on the distance. The **ActionCurve-No.** determines the chronological sequence. This is explained in more detail in the subchapter [Action Curves and Action Paths](#).

Change Bargraph/Instrument calculation for AutoUpdate

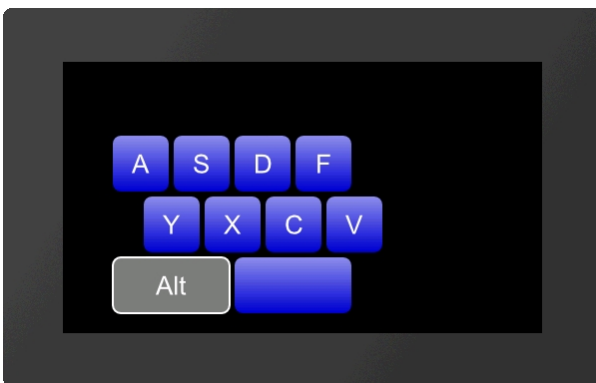
#IVC	Obj-ID, (Calculation)
------	-----------------------

The command changes the **Calculation** for the AutoUpdate function. Now the value of the Bargraph / instrument is updated whenever the new calculation value changes. However, the old calculation ([#IVA](#)) is used for display value.

Keyboard #K

Command group to represent a keyboard for user inputs. The module must be equipped with a touch screen (order numbers: EA uniTFTxxx-ATC or EA uniTFTxxx-ATP). Normally, the keyboard is connected to an EditBox.

Define layout of keyboard (Keyboard Define Buttons)	#KDB	Obj-ID, Layout-No., "ButtonStringLine1"; ...; "ButtonStringLineN"
Definition of alternative key labels / styles (Keyboard Define Styles)	#KDS	Obj-ID, Code, ButtonStyle, "Label"; Code1, ButtonStyle1, "Label1" ...; CodeN, ButtonStyleN, "LabelN";
Place and display keyboard (Keyboard Place)	#KKP	Obj-ID, ButtonStyleNormal, ButtonStyleSpecial, x, y, Anchor, KeySpacing, TotalWidth, TotalHeight(0), Appear(0)



```
...
#KDB 1,1,"ASDF";"\NYXCV";"\O\O ";
#KDS 1,24,3,"Alt";
#KKP 1,1,1,20,20,7,5,300
...
```

Define layout of keyboard

#KDB	Obj-ID, Layout-No., "ButtonStringLine1"; ...; "ButtonStringLineN"
-------------	---

A keyboard can have up to 4 different layouts (**Layout-No.**). Keys (codes) can be assigned to each layout. Multiple lines are separated by the end of the string ';' marked
Keys can be passed as a string (e.g. "ASDF") or as ASCII / Unicode (e.g. \$41 \$53 \$44 \$56). The following key codes are available for special keys:

Code	Code in string	Description
1	\1	Show keyboard No 1
2	\2	Show keyboard No 2
3	\3	Show keyboard No 3
4	\4	Show keyboard No 4
5	\5	SHIFT (Use keyboard No 2 for one key, then automatically keyboard no 1 again)
6	\6	CAPSLOCK (Switch between keyboard no 1 and no 2)
7	\7	DELETE

8	\8	BACKSPACE
9	\9	Reserved for future use
10	\A	Reserved for future use
11	\B	INSERT/ OVERWRITE
12	\C	CLEAR
13	\D	SEND
14	\E	Cursor left
15	\F	Cursor right
16	\G	Reserved for future use
17	\H	Reserved for future use
18	\I	POS1
19	\J	END
20	\K	Reserved for future use
21	\L	CANCEL
22	\M	Cursor On/Off
23	\N	Place holder (this code is not drawn, empty gap)
24 - 31	\O - \V	8 Function keys

Definition of alternative key labels / styles

#KDS	Obj-ID, Code, ButtonStyle, "Label"; Code1, ButtonStyle1, "Label1" ...; CodeN, ButtonStyleN, "LabelN";
-------------	---

A specific key (**Code**) is assigned a special key label ("**Label**") and ButtonStyle. This setting overrides the style definition of the [#KKP](#) command. The ButtonStyles are not completely adopted: The size information from the ButtonStyle is ignored, the radius is adopted once. If the radius changes afterwards in the ButtonStyle, these values are not adopted in the keyboard, but a color change or TextStyle change does.

Place and display keyboard

#KKP	Obj-ID, ButtonStyleNormal, ButtonStyleSpecial, x, y, Anchor, KeySpacing, TotalWidth, TotalHeight(0), Appear(0)
-------------	--

The keyboard defined with the commands [#KDB](#) and [#KDS](#) is placed at **x, y** with the given **Anchor**. The width of a key is automatically calculated from the **TotalWidth** or the **TotalHeight** and the distances between the keys (**KeySpacing**). If the total height or total width = 0, this length is automatically calculated from the resulting key size. The size values are the desired maximum values. The buttons are split evenly. The **ButtonStyleNormal** defines the style for letters and numbers, **ButtonStyleSpecial** applies to special keys. The last parameter (**Appear**) indicates whether the keyboard is displayed immediately or appears according to a defined animation ([#AOA](#) / [#AOR](#)):

Appear	
0	Keyboard is displayed immediately

>=1 (Time)	Keyboard appears according to animation in the defined time (in 1/100s)
--------------------------------	---

Input element per Touch #E

Commands to create touch input elements like menus, SpinBoxes or ComboBoxes. The functions are available from firmware V1.2.

Menu

Define styles for menu (Input Menu Styles)	#EMS	Obj-ID, MenuDirection, TextStyle-No., DrawStyle-No. Background, DrawStyle-No. Selection, DrawStyle-No. Icon, "Note string"
Define entries for menu (Input Menu Define)	#EMD	Obj-ID, ItemNumber, "Entry"
Assign icon to menu entry (Input Menu Icons)	#EMI	Obj-ID, ItemNumber, <Iconname>, ItemNumber2, <Iconname2>, ... ItemNumberN, <IconnameN>
Enable/Disable menu entry (Input Menu Enable)	#EME	Obj-ID, Enable, ItemNumber, ItemNumber2, ... ItemNumberN
Check/Uncheck menu entry (Input Menu Check)	#EMC	Obj-ID, Check, ItemNumber, ItemNumber2, ... ItemNumberN
Place and show menu (Input Menu Place)	#EMP	Obj-ID, x, y, Anchor, Radius(0), BorderX(0), BorderY(0), Time(0)
Select menu entry (Input Menu choOse)	#EMO	Obj-ID, ItemNumber

ComboBox

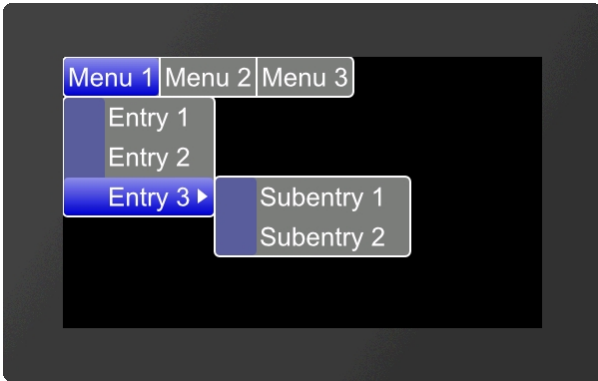
Define styles for ComboBox (Input Comobox Styles)	#ECS	Obj-ID, ComboBoxDirection, TextStyle-No., DrawStyle-No. Background, DrawStyle-No. Selection, DrawStyle-No. Scrollbar, "Note string"
Define entries for ComboBox (Input Comobox Define)	#ECD	Obj-ID, "Entries"
		Obj-ID, "Formatted string"; StartValue, EndValue
Assign icon to ComboBox entry (Input Comobox Icons)	#ECI	Obj-ID, ItemNumber, <Iconname>, ItemNumber2, <Iconname2>, ... ItemNumberN, <IconnameN>
Enable/ Disable ComboBox entry (Input Comobox Enable)	#ECE	Obj-ID, Enable, ItemNumber, ItemNumber2, ... ItemNumberN
Place and show ComboBox (Input Comobox Place)	#ECP	Obj-ID, x, y, Anchor, Radius(0), Width(0), VisiableEntries(0), BorderX(0),BorderY(0), Time(0)
Select ComboBox entry (Input Comobox choOse)	#ECO	Obj-ID, ItemNumber

SpinBox

Define styles for SpinBox (Input Spinbox Styles)	#ESS	Obj-ID, RollingBehaviour, TextStyle-No., DrawStyle-No. Background, DrawStyle-Nr. Selection, "Note string"
Define entries for SpinBox (Input Spinbox Define)	#ESD	Obj-ID, Box-Nr, "Entries"
		Obj-ID, Box-Nr, "Formatted string"; StartValue, EndValue
Assign icon to SpinBox entry (Input Spinbox Icons)	#ESI	Obj-ID, ItemNumber, <Iconname>, ItemNumber2, <Iconname2>, ... ItemNumberN, <IconnameN>
Enable/Disable SpinBox entry (Input Spinbox Enable)	#ESE	Obj-ID, Enable, ItemNumber, ItemNumber2, ... ItemNumberN
Place and show SpinBox (Input Spinbox Place)	#ESP	Obj-ID, x, y, Anchor, Radius, Width, VisiableEntries, BorderX(0), Distance(0)

Select SpinBox entry (Input Spinbox choOse)	#ESO Obj-ID, ItemNumber
---	--------------------------------

Menu



```

...
#EMS 1,0,4,1,5,17
#EMD 1,0,"Menu 1|Menu 2|Menu 3";
#EMD 1,1,"Entry 1|Entry 2|Entry 3";
#EMD 1,769,"Subentry 1|Subentry 2";
#EMP 1,0,271,1,5,5,5,10
...

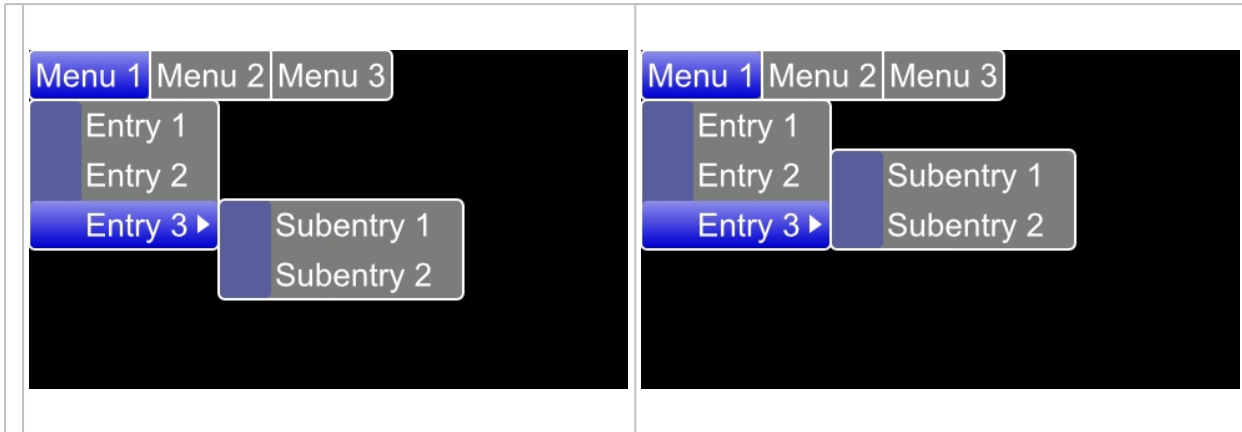
```

Define styles for menu

#EMS	Obj-ID, MenuDirection, TextStyle-No., DrawStyle-No. Background, DrawStyle-No. Selection, DrawStyle-No. Icon, "Note string"
-------------	--

The command determines the appearance of the menu. Three DrawStyles are required. The background of the menu (**DrawStyle-No. Background**), the appearance of the selected entry (**DrawStyle-No. Selection**) and the background of the icon (**DrawStyle-No. Icon**) are defined. The structure is described in more detail in the subsection [DrawStyle](#). The appearance of the character string is determined with the TextStyle (TextStyle No.). This is explained in more detail in the [TextStyle](#) subsection. The direction in which the menu is pulled-down is also set (**MenuDirection**).

MenuDirection								
	0	1	2	3	4	5	6	7
Direction	bottom/right	top/right	bottom/left	top/left	bottom/right	top/right	bottom/left	top/left
	normal				save space			



Finally, the parameter "**Note string**" can optionally be set. This specifies the [note string](#) to be played, while pressing.

Define entries for menu

#EMD Obj-ID, ItemNumber, "Entry"

Der Befehl fügt dem Elternobjekt Untermenüs hinzu. Das Hauptmenü hat die **ItemNumber** 0, die Hauptmenüeinträge \$01 - \$FF. Die Submenüeinträge werden mit dem nächsten höherwertigen Byte zugeordnet (z.B. \$0301 ordnet dem dritten Eintrag des ersten Menüeintrages weitere Subeinträge zu). Nachfolgend ist dies exemplarisch aufgeführt.

The command adds sub-menus to the parent object. The main menu has the **ItemNumber** 0, the main menu entries \$ 01 - \$ FF. The submenu entries are assigned with the next higher byte (e.g. \$ 0301 assigns further subentries to the third entry of the first menu entry).

ItemNumber			
Main menu 1 \$01	Main menu 2 \$02	Main menu 3 \$03	...
Menu 1 \$01 01			
Menu 2 \$02 01			
Menu 3 \$03 01	Sub-menu 1 \$01 03 01		
...	Sub-menu 2 \$02 03 01		
	...		

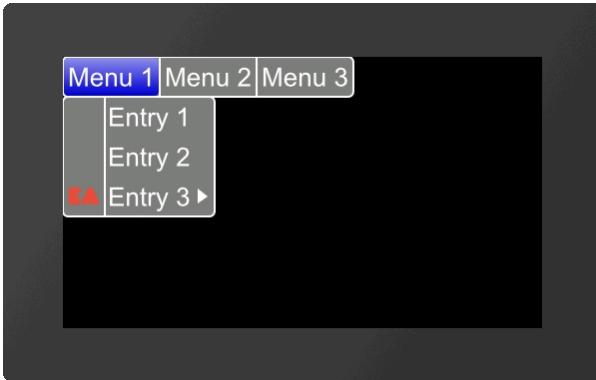
The individual entries are displayed as a string ("**Entry**") with a pipe '|' handed over separately. A double pipe '||' adds a hyphen / separator.

Assign icon to menu entry

#EMI Obj-ID, ItemNumber, <Iconname>, ItemNumber2, <Iconname2>, ... ItemNumberN, <IconnameN>

An icon can be assigned to each entry ([ItemNumber](#)) <Iconname>. In order for an icon to be assigned, the entry

must already exist.



```
...
#EMI 1,$0301,<P:picture/EA.epg>
...

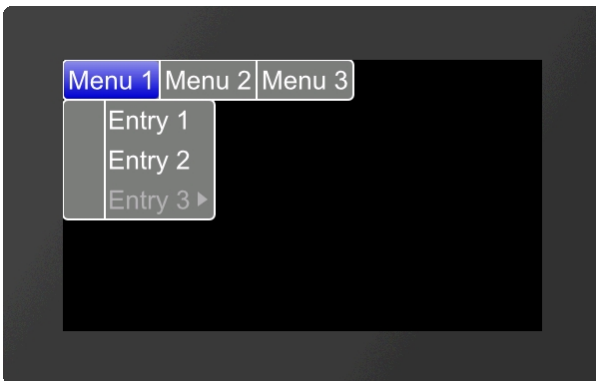
...
#EMI 1,$0301,"EA";
...
```

Enable/Disable menu entry

#EME Obj-ID, Enable, ItemNumber, ItemNumber2, ... ItemNumberN

The command activates/deactivates an entry ([ItemNumber](#)). If an entry is deactivated, it cannot be selected by touch. By default, all entries are active.

Enable	
0	Disable
1	Enable
2	Toggle



```
...
#EME 1,0,$0301
...
```

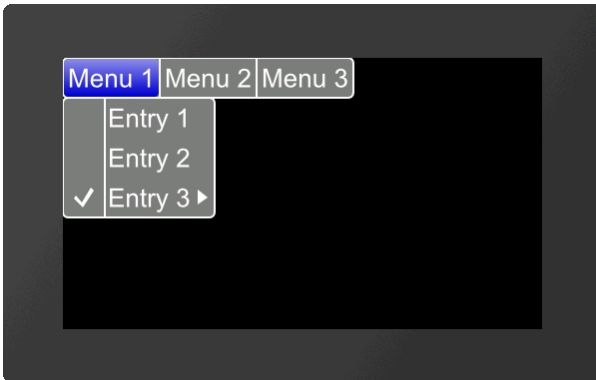
Check/Uncheck menu entry

#EMC Obj-ID, Check, ItemNumber, ItemNumber2, ... ItemNumberN

The command selects / deselects an entry ([ItemNumber](#)). A check mark is displayed visually. No entry is selected by default.

Check	
0	Deselect
1	Select

2	Toggle
---	--------



```
...
#EMC 1,1,$0301
...
```

Place and show menu

#EMP	Obj-ID, x, y, Anchor, Radius(0), BorderX(0), BorderY(0), Time(0)
------	--

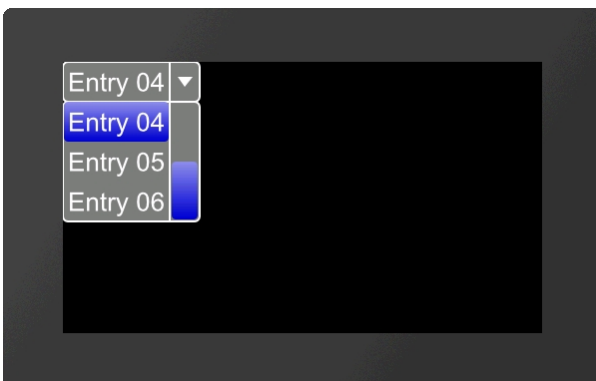
The menu defined with the commands [#EMS](#) and [#EMD](#) is placed with the given **Anchor** at **x, y**. The **Radius** parameter gives the corner rounding. With the two optional parameters (**BorderX** and **BorderY**) the distance of the text to the edge of the menu is specified. With the parameter **Time** the opening / closing of the menu can be animated in 1/100s.

Select menu entry

#EMO	Obj-ID, ItemNumber
------	--------------------

The command selects an entry ([ItemNumber](#)).

ComboBox



```
...
#ECS 1,0,6,1,5,5
#ECD 1,"Entry %02d";1,6
#ECP 1,0,271,1,5,0,3,5,5,10
#ECO 1,4
...
```

Define styles for ComboBox

#ECS	Obj-ID, ComboBoxDirection, TextStyle-No., DrawStyle-No. Background, DrawStyle-No. Selection, DrawStyle-No. Scrollbar, "Note string"
------	---

The command defines the appearance of the ComboBox. Three DrawStyles are required. The background of the ComboBox (**DrawStyle-No. Background**), the appearance of the selected entry (**DrawStyle-No. Selection**) and the

the scrollbar (**DrawStyle No. Scrollbar**) are defined. The structure is described in more detail in the subsection [DrawStyle](#). The appearance of the character string is determined with the TextStyle (TextStyle No.). This is explained in more detail in the [TextStyle](#) subsection. The direction in which the ComboBox is pulled-down is also set (**ComboBoxDirection**).

ComboBoxDirection	
0	Down
1	Up

Finally, the parameter "**Note string**" can optionally be set. This specifies the [note string](#) to be played, while pressing.

Define entries for ComboBox

#ECD	Obj-ID, "Entries"
	Obj-ID, "Formatted string"; StartValue, EndValue

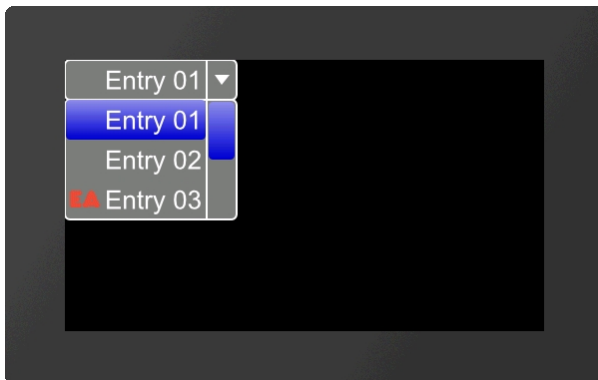
There are two ways to transfer the ComboBox entries:

1. The individual entries are displayed as a string ("**Entries**") with a pipe '|' passed separately (e.g. "**Entry1 | Entry2 | Entry3**" ;)
2. The individual entries are transferred as a format string with start and end values (e.g. "**Entry %d**" ; 1, 3)

Assign icon to ComboBox entry

#ECI	Obj-ID, ItemNumber, <Iconname>, ItemNumber2, <Iconname2>, ... ItemNumberN, <IconnameN>
------	--

An icon can be assigned to each entry (**ItemNumber**) **<Iconname>**. The entry must already exist.



```

...
#ECI 1,3,<P:picture/EA.epg>
...

...
#ECI 1,3,"EA";
...

```

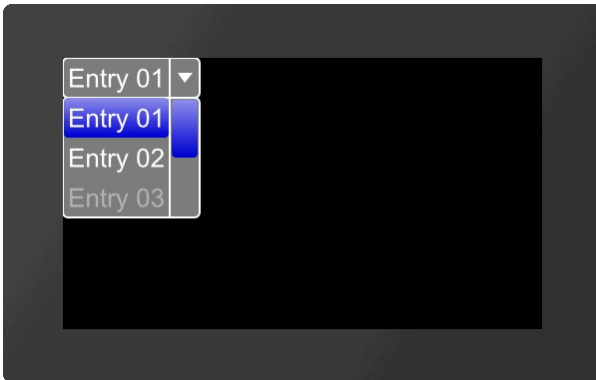
Enable/ Disable ComboBox entry

#ECE	Obj-ID, Enable, ItemNumber, ItemNumber2, ... ItemNumberN
------	--

The command activates / deactivates an entry (**ItemNumber**). If an entry is deactivated, it cannot be selected by touch. By default, all entries are active.

Enable	
0	Disable
1	Enable

2	Toggle
---	--------



```
...
#ECE 1, 0, 3
...
```

Place and show ComboBox

#ECP	Obj-ID, x, y, Anchor, Radius(0), Width(0), VisibleEntries(0), BorderX(0), BorderY(0), Time(0)
------	---

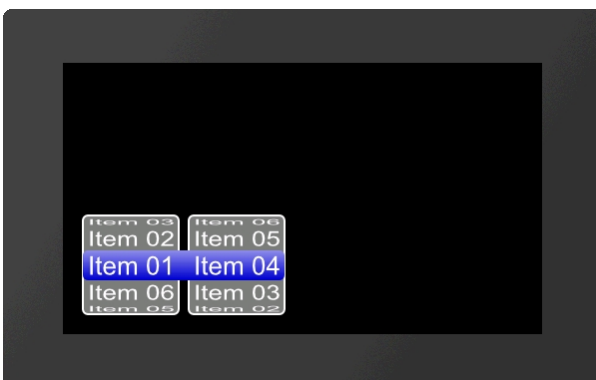
The ComboBox defined with the commands #ECS and #ECD is placed in with the given **Anchor** at **x, y**. The **Radius** parameter gives the corner rounding. **Width** indicates the width of the box in pixels. If width = 0, the width of the box is automatically determined based on the widest entry. The parameter **VisibleEntries** defines the number of visible entries (VisibleEntries = 0: all entries visible). With the two optional parameters (**BorderX** and **BorderY**) the distance of the text to the edge of the menu can be specified. With the parameter **Time** the opening / closing of the ComboBox can be animated in 1/100s.

Select ComboBox entry

#ECO	Obj-ID, ItemNumber
------	--------------------

The command selects an entry (**ItemNumber**).

SpinBox



```
...
#ESS 1, 0, 4, 1, 5
#ESD 1, 1, "Item %02d"; 1, 6
#ESD 1, 2, "Item %02d"; 1, 6
#ESP 1, 20, 20, 7, 5, 0, 5, 5, 10
#ESO 1, $0400
...
```

Define styles for SpinBox

#ESS	Obj-ID, RollingBehaviour, TextStyle-No., DrawStyle-No. Background, DrawStyle-Nr. Selection, "Note string"
------	---

The command defines the appearance of the SpinBox. Two DrawStyles are required. The background of the SpinBox

(**DrawStyle-No. Background**) and the appearance of the selected entry (**DrawStyle-No. Selection**) are defined. The structure is described in more detail in the [DrawStyle](#) subsection. The appearance of the character string is determined with the **TextStyle (TextStyle-No.)**. This is explained in more detail in the [TextStyle](#) subsection. The **RollingBehavior** position of the selection frame of the SpinBox is also defined here:

		RollingBehaviour			
		0	1	2	3
RollingBehaviour		Endless	With stop	Endless	With stop
Selection frame		Behind the text		Before the text	

Finally, the parameter "**Note string**" can optionally be set. This specifies the [note string](#) to be played, while pressing.

Define entries for SpinBox

#ESD	Obj-ID, Box-Nr, "Entries"
	Obj-ID, Box-Nr, "Formatted string"; StartValue, EndValue

Eine SpinBox kann bis zu 4 untergeordnete Boxen besitzen. Der Parameter **Box-Nr.** gibt die aktuelle Box an. Es gibt zwei Möglichkeiten die Einträge der SpinBox zu übergeben:

A SpinBox can have up to 4 subordinate boxes. The parameter **Box-No.** indicates the current box. There are two ways to transfer the entries of the SpinBox:

1. The individual entries are displayed as a string ("**Entries**") with a pipe '|' passed separately (e.g. "**Entry1 | Entry2 | Entry3**" ;)
2. The individual entries are transferred as a format string with start and end values (e.g. "**Entry %d**" ; 1, 3)

Assign icon to SpinBox entry

#ESI	Obj-ID, ItemNumber, <Iconname>, ItemNumber2, <Iconname2>, ... ItemNumberN, <IconnameN>
------	--

An icon can be assigned to each entry **<Iconname>**. To assign an icon, the entry must already exist. **ItemNumber** is composed as follows:

	ItemNumber			
	Box 1	Box 2	Box 3	Box 4
Item 1	\$01	\$01 00	\$01 00 00	\$01 00 00 00
Item 2	\$02	\$02 00	\$02 00 00	\$02 00 00 00
Item 3	\$03	\$03 00	\$03 00 00	\$03 00 00 00
...				



```

...
#ESI 1,$0200,<P:picture/EA.epg>
...
...
#ESI 1,$0200,"EA";
...

```

Enable/Disable SpinBox entry

#ESE	Obj-ID, Enable, ItemNumber, ItemNumber2, ... ItemNumberN
-------------	--

The command activates / deactivates an entry (**ItemNumber**). If an entry is deactivated, it cannot be selected by touch. By default, all entries are active.

Enable	
0	Disable
1	Enable
2	Toggle



```
...
#ESE 1,0,$0200
...
```

Place and show SpinBox

#ESP	Obj-ID, x, y, Anchor, Radius, Width, VisibleEntries, BorderX(0), Distance(0)
-------------	--

The SpinBox defined with the commands [#ESS](#) and [#ESD](#) is placed with the given **Anchor** at **x, y**. The **Radius** parameter gives the corner rounding. **Width** indicates the width of the box in pixels. If width = 0, the width of the box is automatically determined based on the widest entry. The parameter **VisibleEntries** defines the entries to be displayed above the selection box. **BorderX** specifies the distance between the entry and the box edge and the icon. **Distance** defines the distance of the boxes within the SpinBox group.

Select SpinBox entry

#ESO	Obj-ID, ItemNumber
-------------	--------------------

The command selects an entry ([ItemNumber](#)).

Action / Animation #A

Command group to animate objects, e.g. Show, fly away, rotate or fade out.

Define and set animation

Start/end animation definition (Action Define Condition)	#ADC	Start
Define animation absolut (Action Object Absolut)	#AOA	Obj-ID, Action1, ..., ActionN
Define animation relative (Action Object Relative)	#AOR	Obj-ID, Action1, ..., ActionN
Set animation type and time (Action Object Type)	#AOT	Obj-ID, Type, TotalTime(100), Start (0), End(0)
Stop animation (Action Object Stop)	#AOS	Obj-ID, Stop(0), Abort(0)
Delete animation (Action Object Delete)	#AOD	Obj-ID, ..., Obj-IDn

Define action paths and action curves

Define action curve (Action Curve Define)	#ACD	CurveNumber, x1, y1, x2, y2
---	-------------	-----------------------------

Define and set animation

Start/end animation definition

#ADC	Start
-------------	-------

If the animation definition is within a macro, this command is not necessary because a macro is always processed completely before the screen content is redrawn.

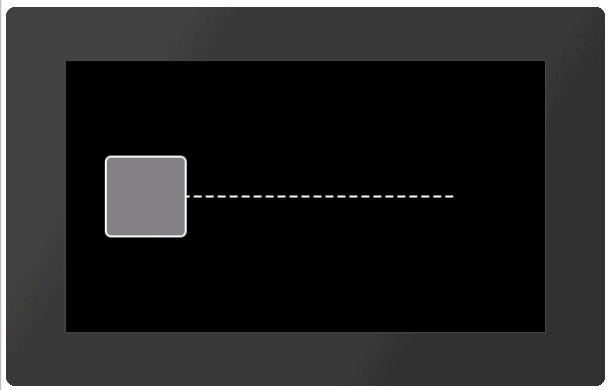
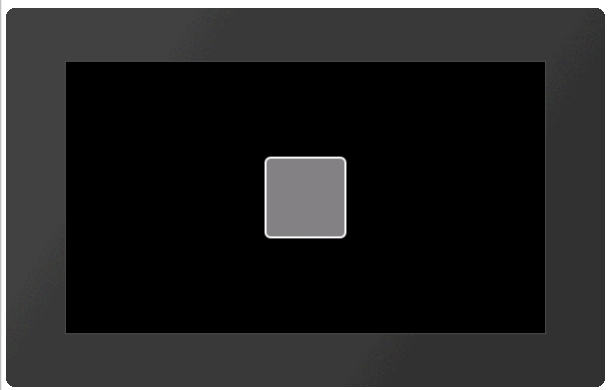
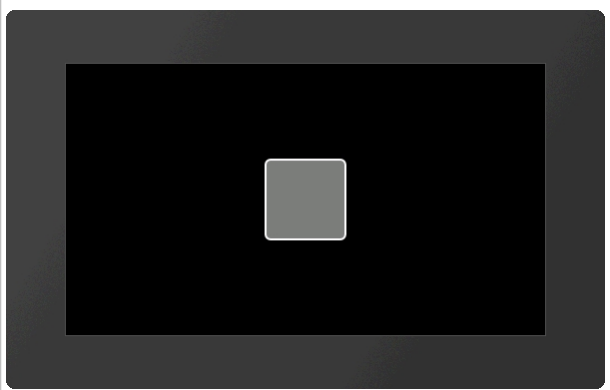
Start	
0	End animation definition start all new animations
1	Start animation definition no animation is executed

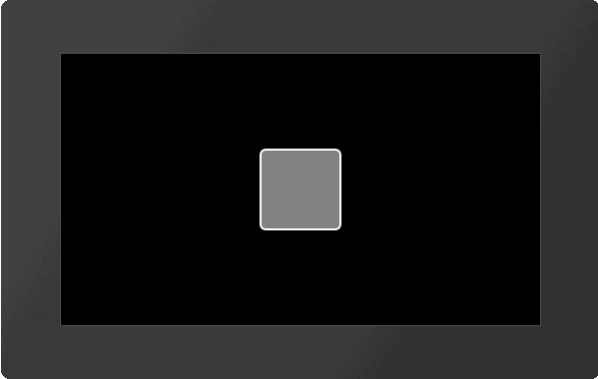
Define animation absolut/relative

#AOA	Obj-ID, Animation1, ..., AnimationN
#AOR	

The command defines an animation for an object. The **Animation** parameter specifies the animation. Depending on the command, **absolute #AOA** or **relative #AOR** values are transferred.

Animation

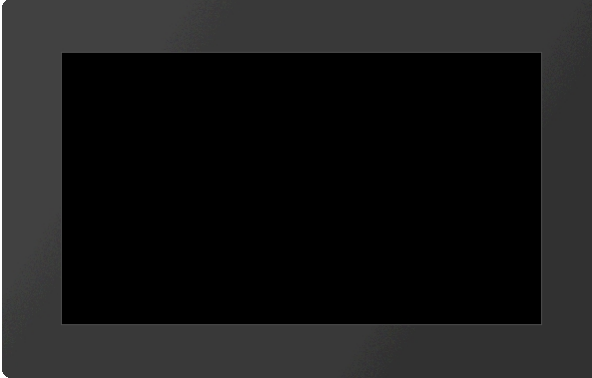
Change position	100+ 1..10	x, y		... #AOA 1, 104 , 400, 136 ...
	The object moves in a straight line to the specified point (x, y). The course of time is specified by the action curve (1..10).			
Change scale	200+ 1..10	ScaleX, ScaleY		... #AOA 1, 204 , 200, 200 ...
	The object size is changed linearly in percentage to the original size. The course of time is specified by the action curve (1..10).			
Change opacity	500+ 1..10	Opacity		... #AOA 2, 504, 0 ...
	The object changes the Opacity . The course of time is specified by the action curve (1..10).			


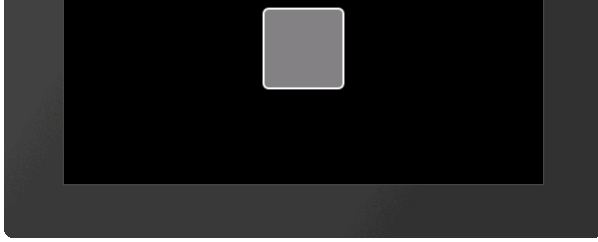
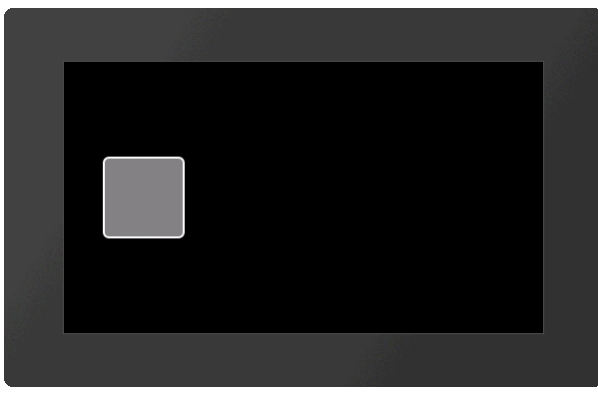
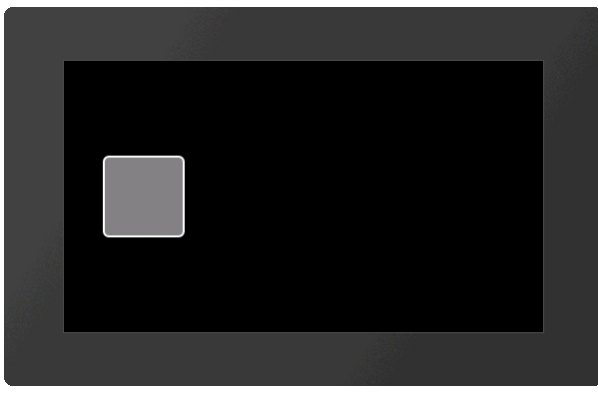
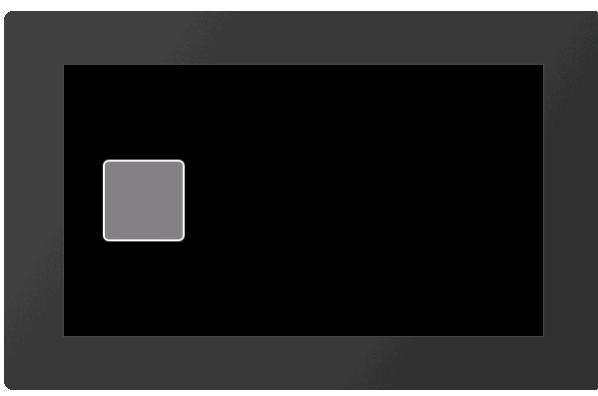
	600+ 1..10	R, G, B		<p>...</p> <p>#AOA 2,604,- 27,-1,18</p> <p>...</p>
<p>Change color channel</p>	<p>Change the color channels Red, Green and Blue. The color channel of the target color is determined relative to the parameters passed. The parameters (R, G, B) are transferred as percentage values in the range from -100 to 100.</p> <p><u>Example:</u> Assume that the output color should be changed from RGB (50.0.0) to RGB (200.0.0). The target color has only changed in the red component. The difference in the red component is 150. The value has to be converted into a percentage value:</p> $\frac{150}{255} \cdot 100 = 117,65$ <p>#AOA 1,604,118,0,0</p> <p>The changing progress is specified by the action curve (1..10).</p>			

Set animation type and time

#AOT	Obj-ID, Type, TotalTime(100), Start (0), End(0)
------	---

The **TotalTime** (in 1 / 100s) of the animation includes the delay at the **Start** (in 1 / 100s) and at the **End** (in 1 / 100s). The **Type** specifies the animation type:

		Typ
1	Appear	

2		Object is deleted	
3	Disappear	Object becomes invisible	
4	Change (once)		
5	Cyclical		
6	PingPong		

The animation starts automatically after the command. However, if the [#ADC](#) command was called beforehand **with parameter 1**, the animation is only executed after [#ADC 0](#).

Stop animation

#AOS Obj-ID, Stop(0), Abort(0)

The animation is stopped. The command can only be used for periodic animations (cyclic / ping pong). The **Stop** parameter specifies the point of time:

Stop	
0	Immediately
1	At the beginning of animation
2	At the end of animation

The **Abort** behavior indicates what to do with the object:

Abort	
0	Object keeps current state
1	Jump (only with Stop = 1 or = 2)
2	Delete object
3	Make object invisible

Delete animation

#AOD Obj-ID, ..., Obj-IDn

The command deletes one or more animations. If the object ID 0 is passed, all animations are deleted.

Define action curves

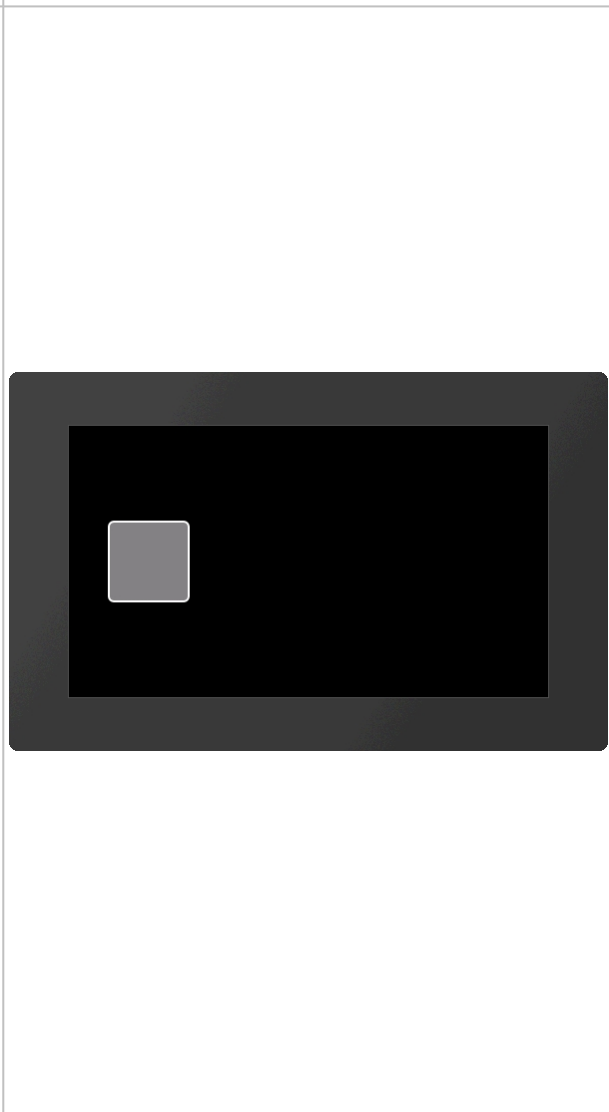
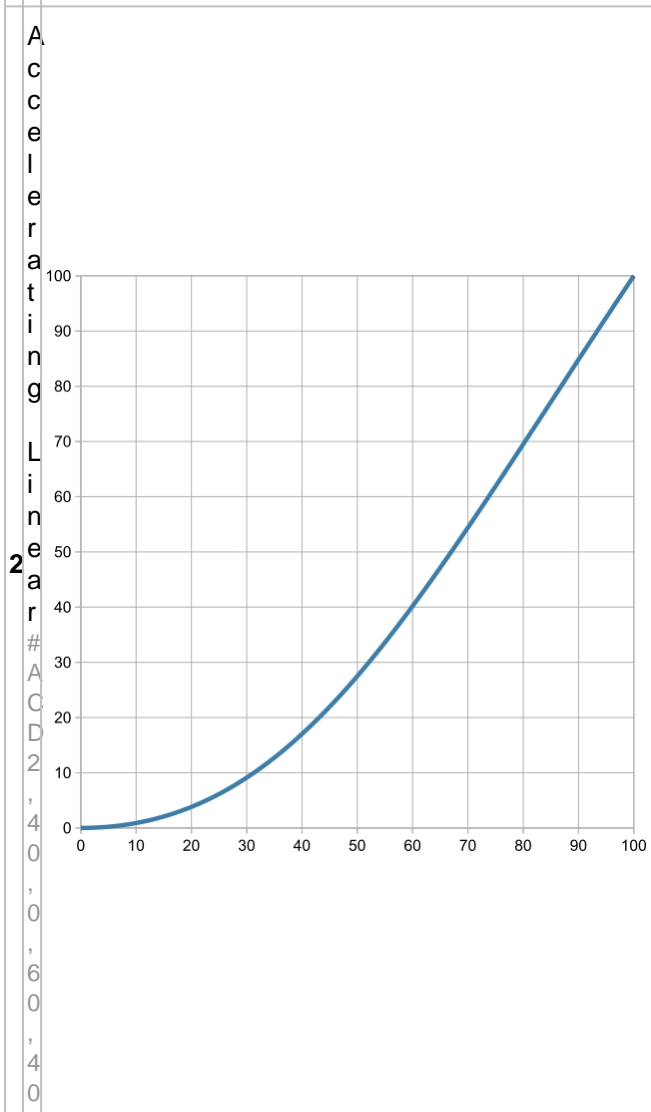
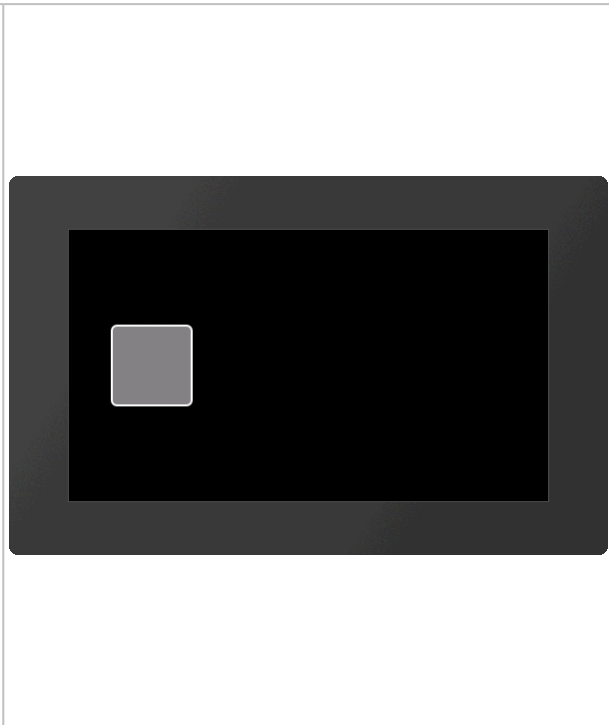
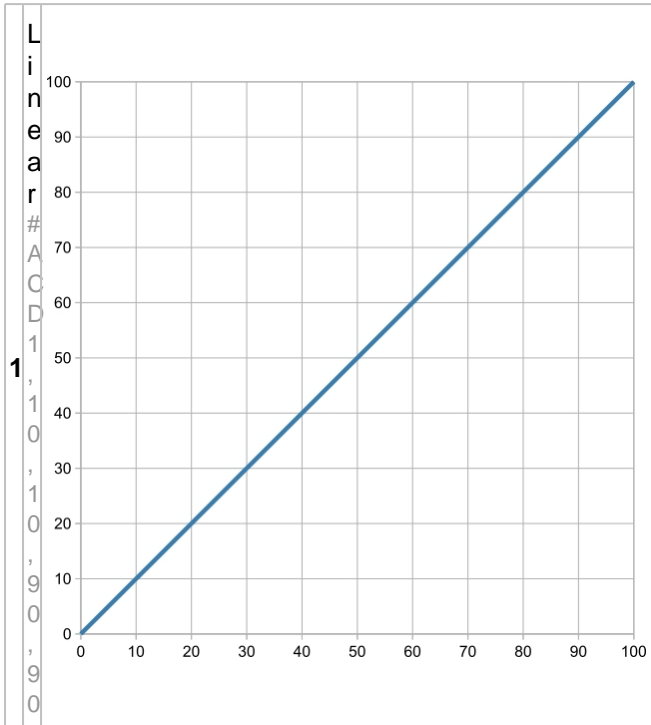
#ACD CurveNumber, x1, y1, x2, y2

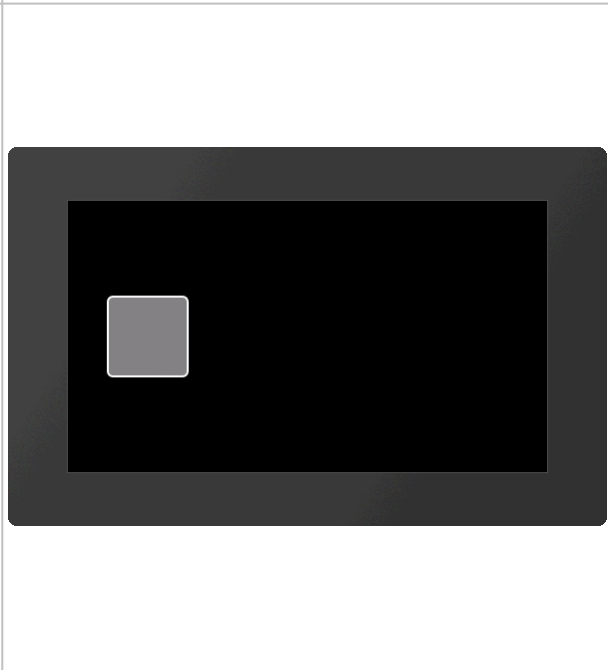
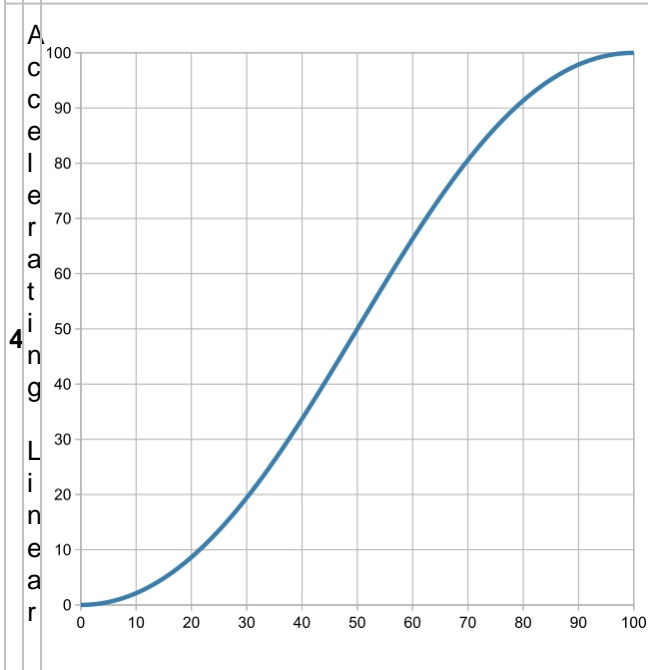
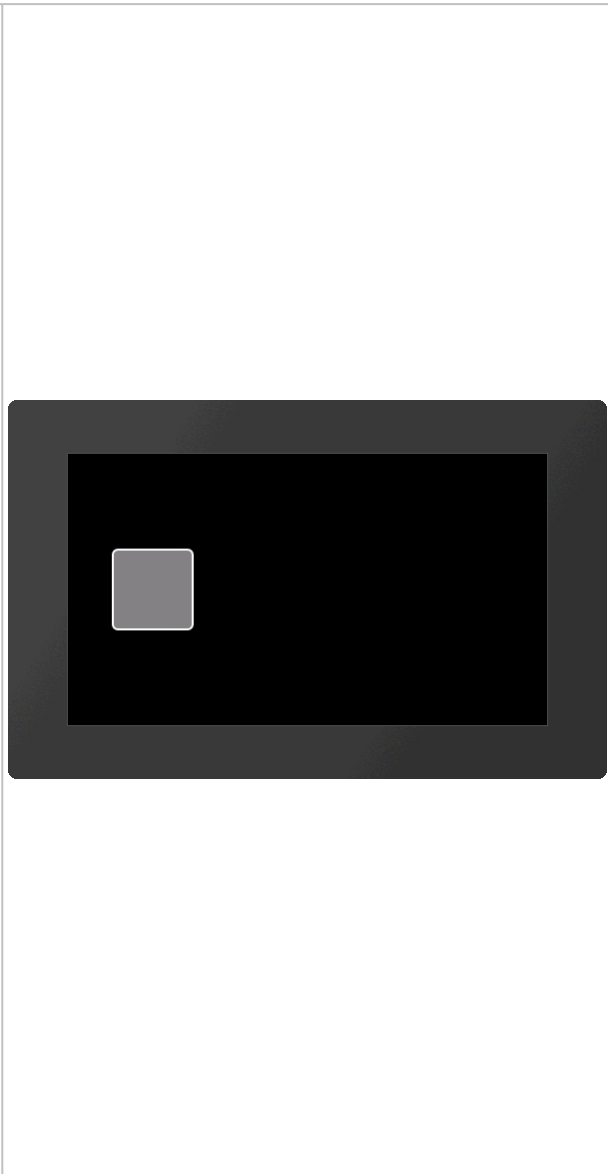
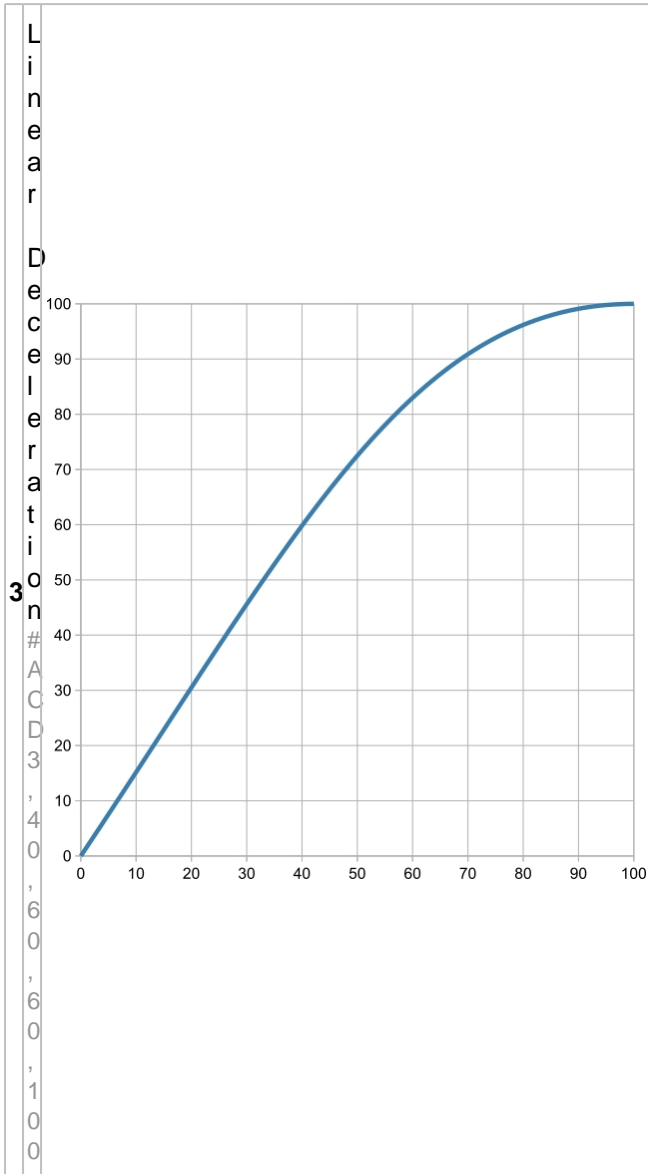
The command creates its own action curve, which specifies the time course of the animation. There are 10 predefined curves that can be overwritten. **CurveNumber** (1-10) indicates the action curve that is overwritten. The action curve is a cubic Bézier curve. The control points of the curve are specified with the parameters **x1**, **y1**, **x2** and **y2**. The value range of the parameters is for **X** 0 ... 100, for **Y** -200 ... 300.

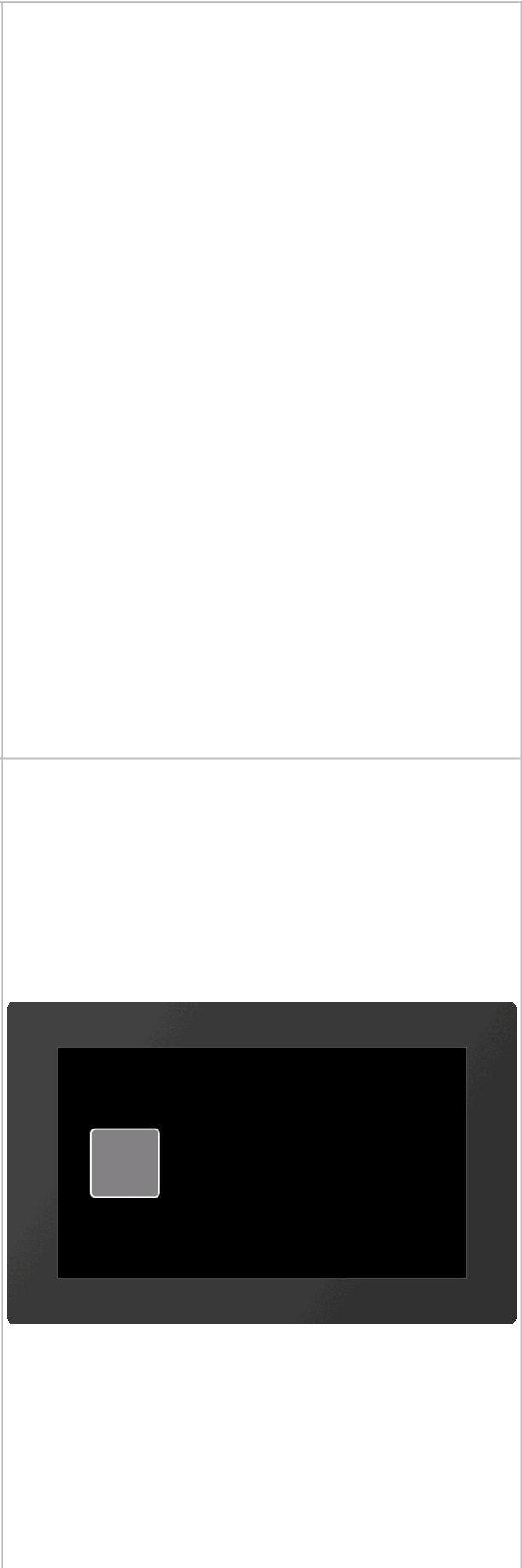
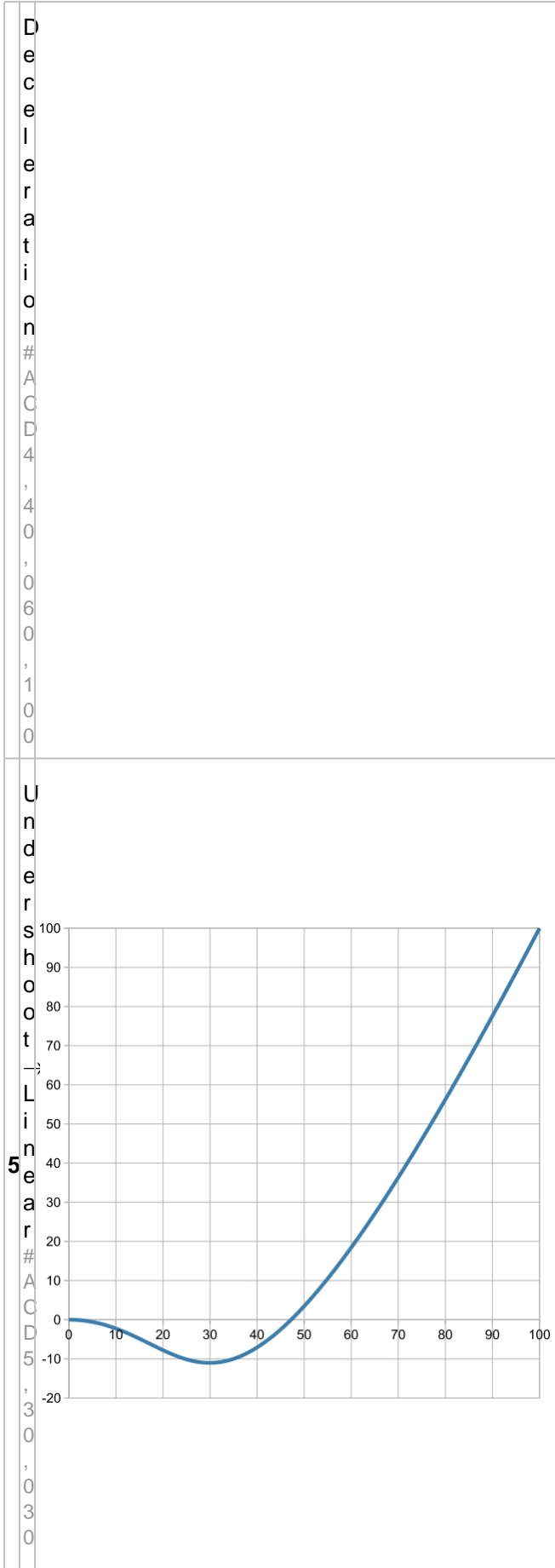
Predefined action curves

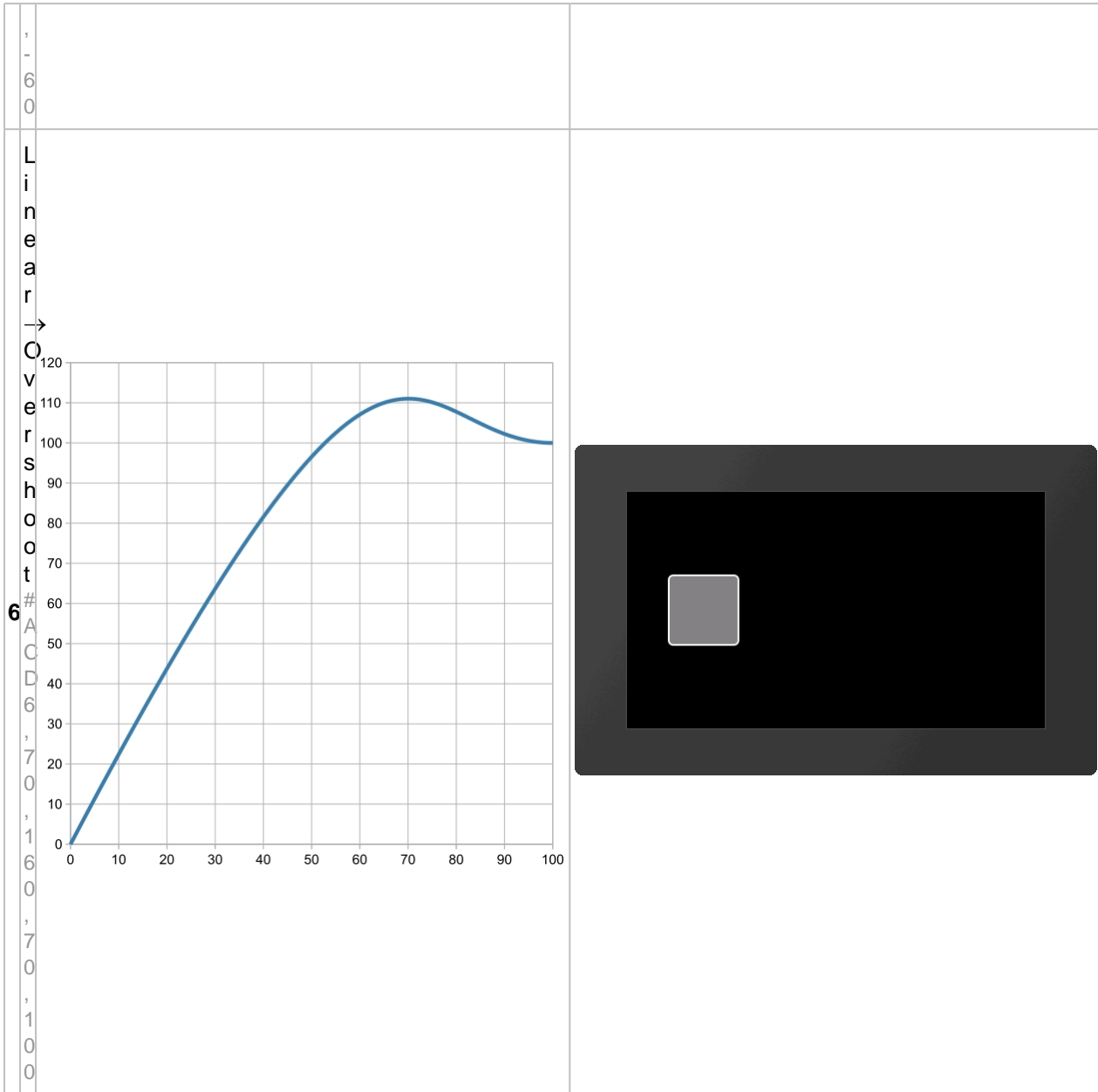
The action curves indicate the time course of the animation. There are 10 predefined curves that can be changed (#ACD). The curves are cubic Bezier curves with two control points:

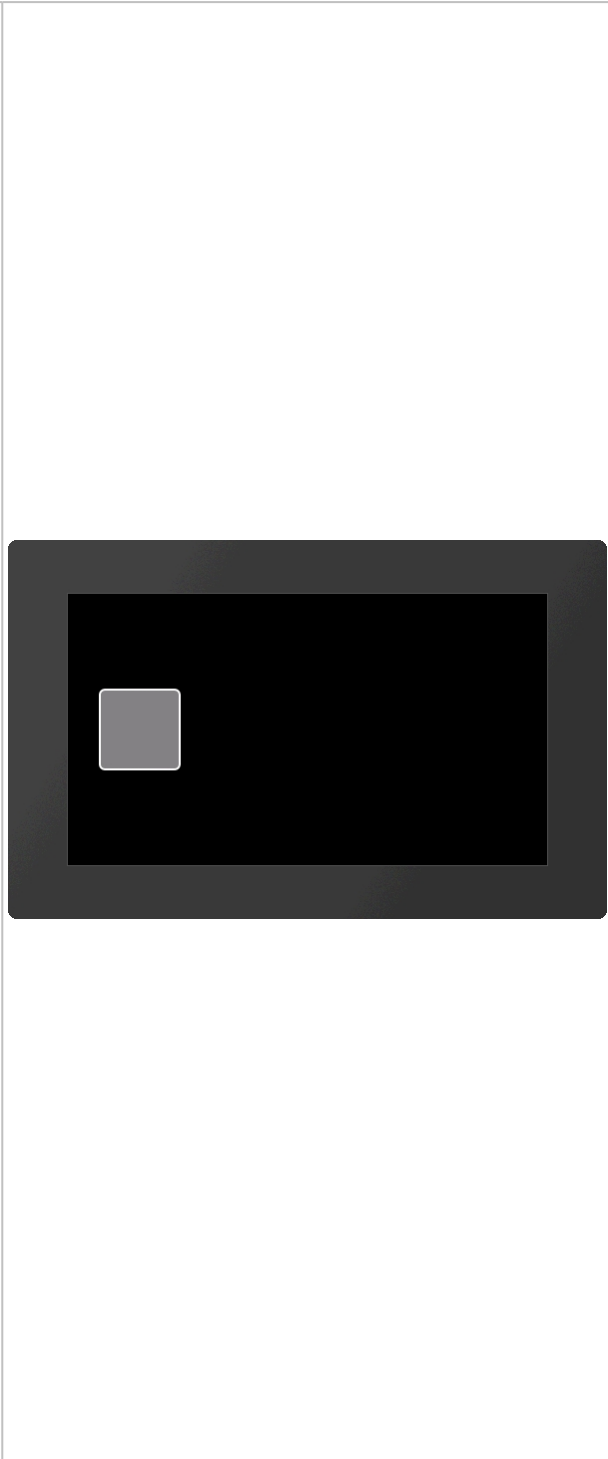
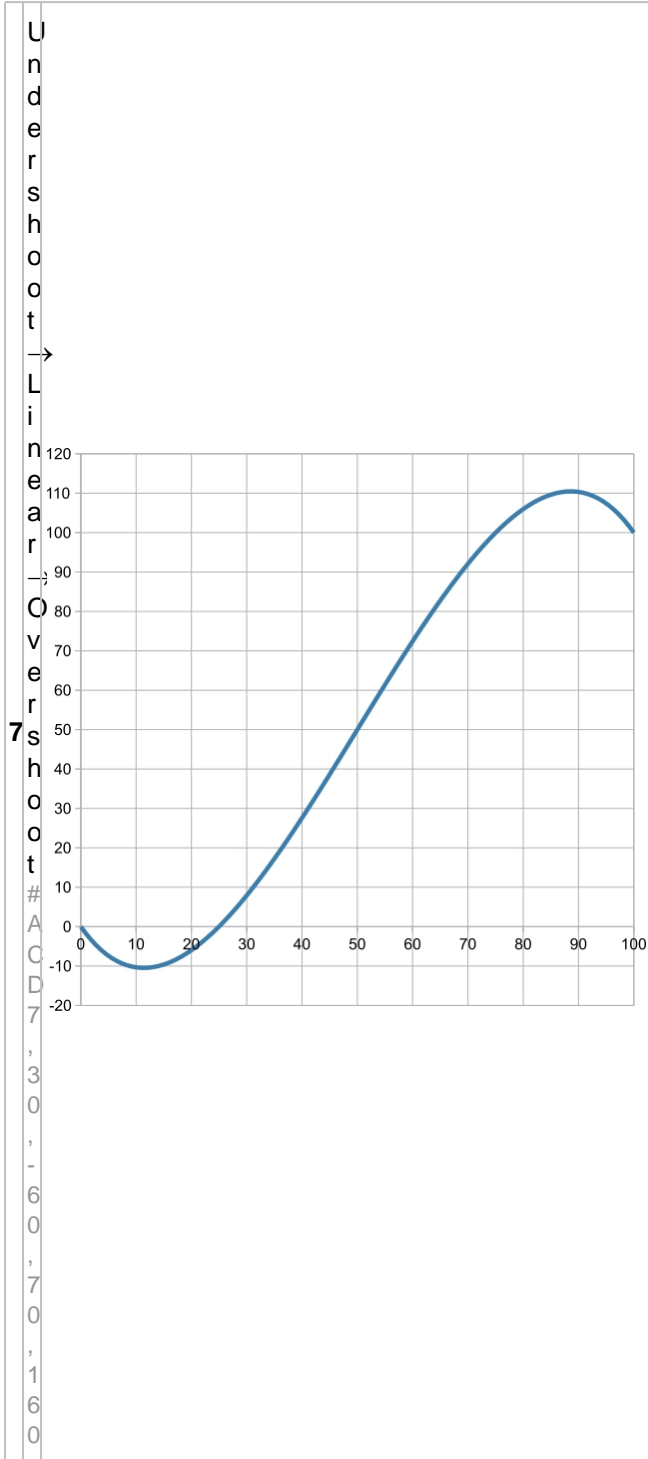
Action curve

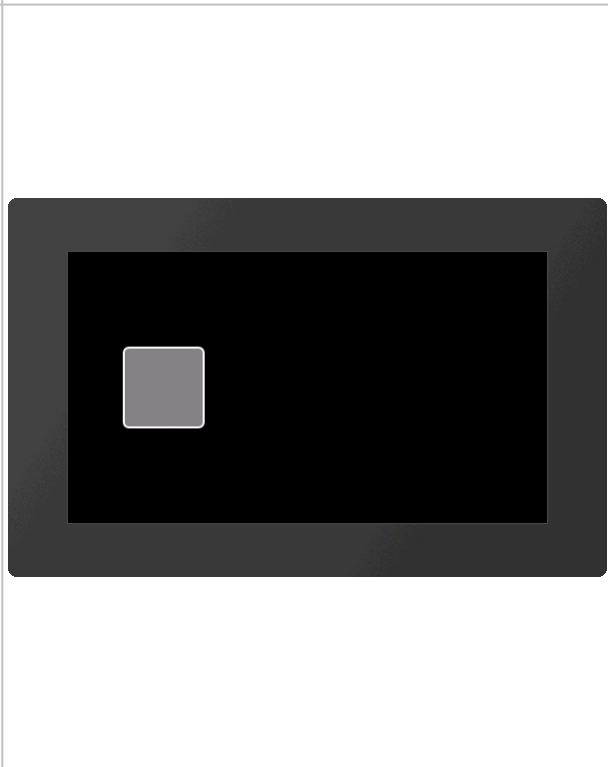
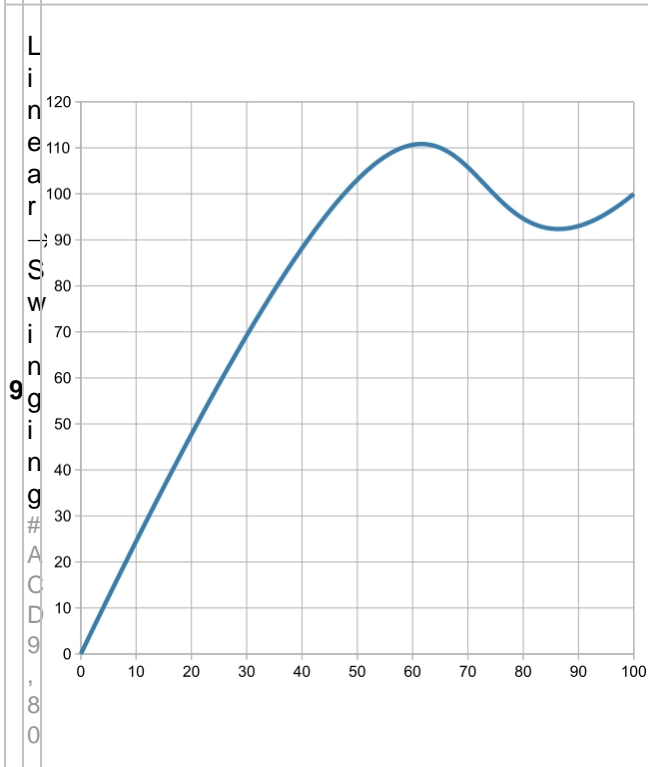
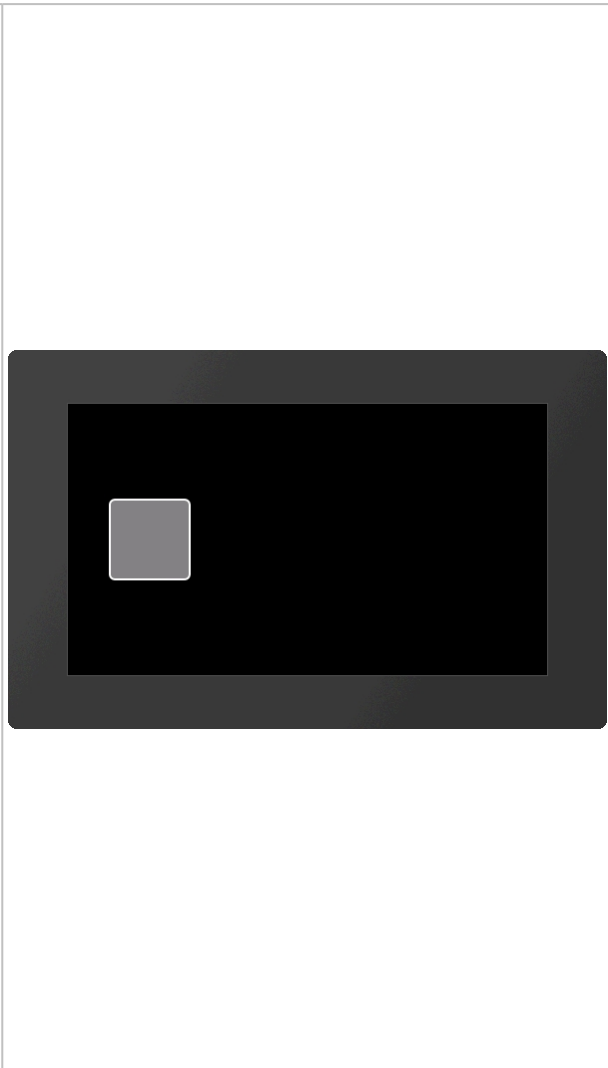
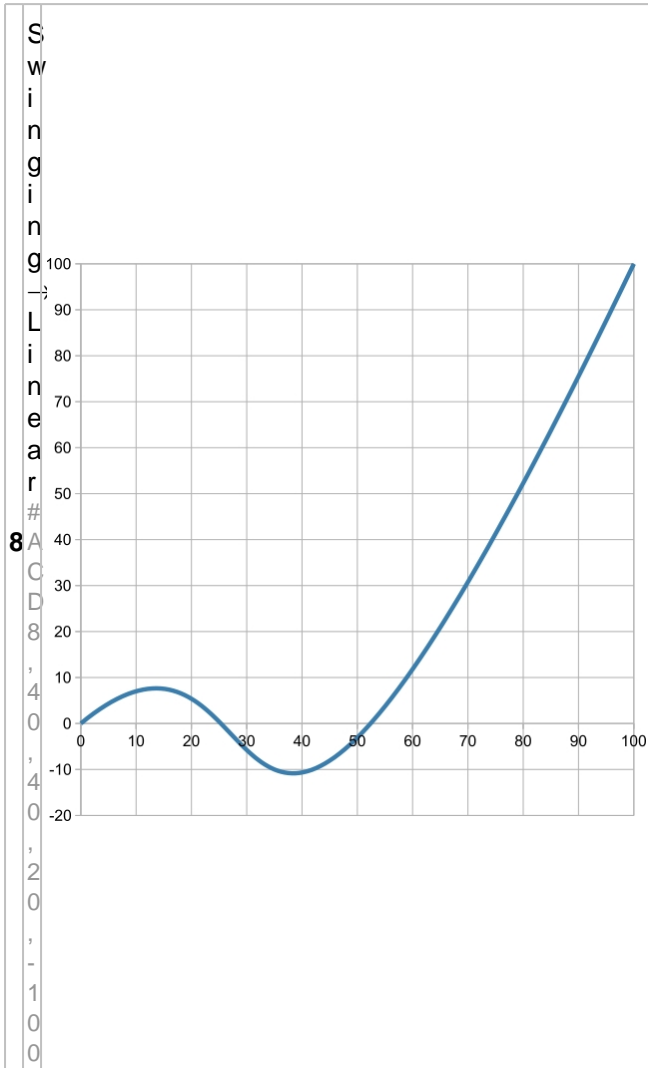


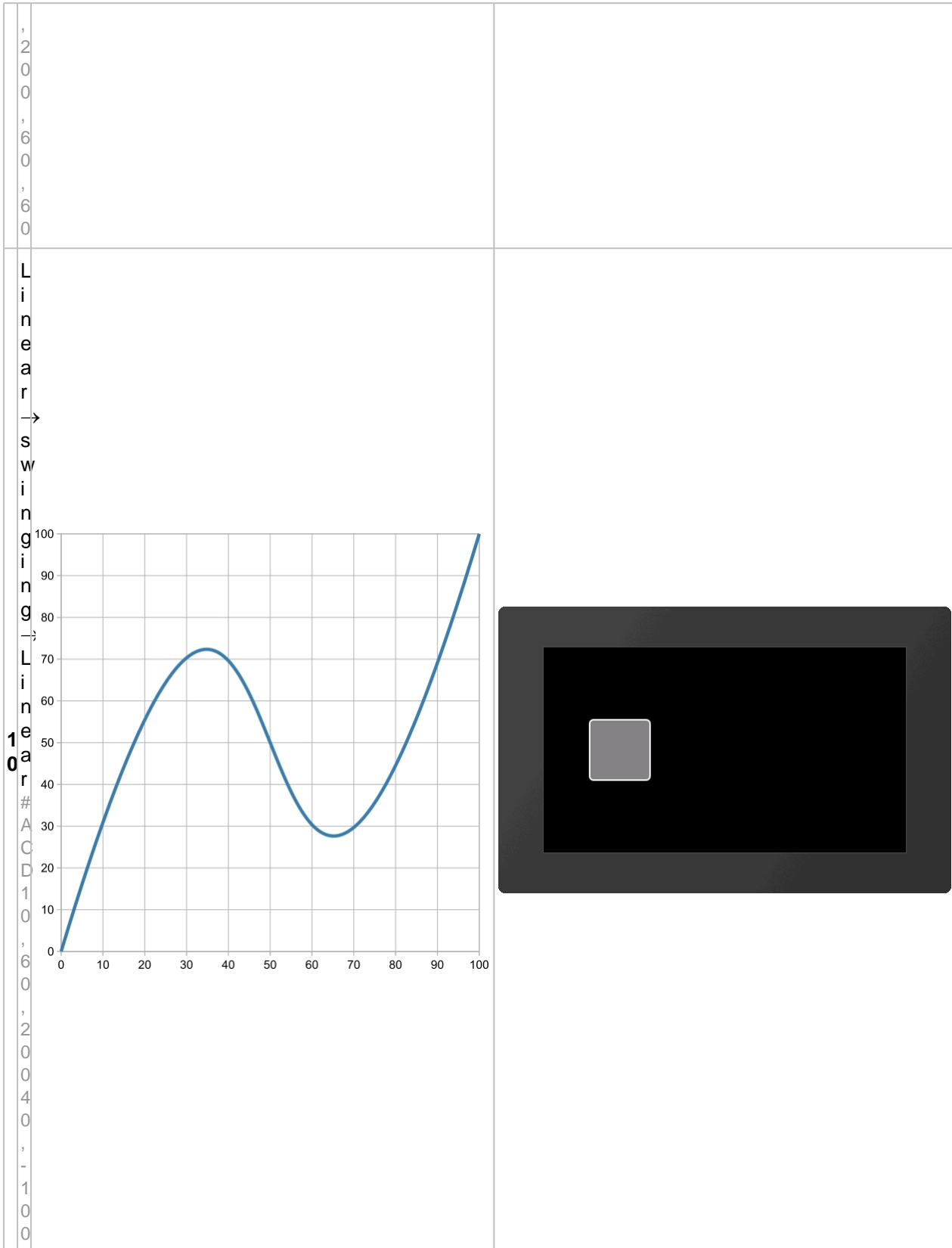












Object management #O

Command group to manage, modify and group objects.

Object manipulation

Delete object (Object Delete Id)	#ODI	Obj-ID, ..., Obj-IDn
Object delete protection (Object Delete Protection)	#ODP	DeleteProtection, Obj-ID, ..., Obj-IDn
Change visibility of object (Object Visible Id)	#OVI	Visibility, Obj-ID, ..., Obj-IDn
Change position absolut (Object Position Absolut)	#OPA	x, y, Obj-ID, ..., Obj-IDn
Change position relative (Object Position Relative)	#OPR	x, y, Obj-ID, ..., Obj-IDn
Change size absolut (Object Scale Absolut)	#OSA	Width, Height, Obj-ID, ..., Obj-IDn
Change size relative (Object Scale Relative)	#OSR	Width, Height, Obj-ID, ..., Obj-IDn
Rotate object absolut (Object Rotation Absolut)	#ORA	Angle, Obj-ID, ..., Obj-IDn
Rotate object relative (Object Rotation Relative)	#ORR	Angle, Obj-ID, ..., Obj-IDn
Change opacity absolut (Object Opacity Absolut)	#OOA	Transparency, Obj-ID, ..., Obj-IDn
Change opacity relative (Object Opacity Relative)	#OOR	Transparency, Obj-ID, ..., Obj-IDn
Change object color (Object Change Color)	#OCC	R,G,B, Obj-ID, ..., Obj-IDn
Change object style (Object Change Style)	#OCS	Style-No, Obj-ID, ..., Obj-IDn
Define frame/background (Object Frame Place)	#OFP	DrawStyleNo, addX, addY, Obj-ID, ..., Obj-IDn
Set anchor (Object Anchor Active)	#OAA	Anchor, Obj-ID, ..., Obj-IDn
Set free anchor absolut (Object Anchor Screen)	#OAS	x, y, Obj-ID, ..., Obj-IDn
Set free anchor relative (Object Anchor Object)	#OAO	x, y, Obj-ID, ..., Obj-IDn
Change draw order (layer) absolut (Object Layer Absolut)	#OLA	Draw order, Obj-ID, ..., Obj-IDn
Change draw order (layer) relative (Object Layer Relative)	#OLR	Draw order, Obj-ID, ..., Obj-IDn
Set user value (integer) (Object User Integer)	#OUI	Obj-ID, Value, ..., Value n (Obj-ID n)
Set user value (float) (Object User Float)	#OUF	Obj-ID, Value, ..., Value n (Obj-ID n)

Group

Add object to group (Object Group Add)	#OGA	Obj-ID Group, Obj-ID, ..., Obj-IDn
--	-------------	------------------------------------

Background

Move object to the background layer (Object to BackGround)	#OBG	RGB, Obj-ID, ..., Obj-IDn
Load image into background layer (Object Background Picture)	#OBP	<Name>, x(0), y(0), Anchor(7),<Gradient>, Time, Direction, 'Endmacro'

Object manipulation

Delete object

#ODI	Obj-ID, ..., Obj-IDn
-------------	----------------------

The command deletes single or multiple objects. If the **Obj-ID = 0** is transferred, all objects, with Obj-ID = -1 all objects and the background are deleted (from V1.2).

Object delete protection

#ODP	DeleteProtection, Obj-ID, ..., Obj-IDn
-------------	--

Objects with **DeleteProtection = 1** cannot be deleted by the [#ODI](#) command and remain. They are also not moved to the background level (from V1.2).

Change visibility of object

#OVI	Visibility, Obj-ID, ..., Obj-IDn
-------------	----------------------------------

The command sets the **Visibility** of objects. If the **Obj-ID = 0** is passed, the command is applied to all objects:

Visibility	
0	Invisible
1	Visible

See also [objV\(id\)](#) (from V1.4)

Change position absolut/relative

#OPA	x, y, Obj-ID, ..., Obj-IDn
#OPR	

The command moves objects (absolute or relative) to the new position. If the **Obj-ID = 0** is passed, all objects are moved.

See also [objX\(id\)](#), [objY\(id\)](#)

Change size absolut/relative

#OSA	Width, Height, Obj-ID, ..., Obj-IDn
#OSR	

Change the Width or Height of an object as a percentage of the object size. **Obj-ID** = 0 Size change for all objects.

See also [objW\(id\)](#), [objH\(id\)](#), [objSW\(id\)](#), [objSH\(id\)](#)

Rotate object absolut/relative

#ORA	Angle, Obj-ID, ..., Obj-IDn
#ORR	

The object (**Obj-ID**) is rotated by the **Angle**. Obj-ID = 0 rotation of all objects. Only 90° steps are allowed for the angle.

See also [objR\(id\)](#)

Change opacity absolut/relative

#OOA	Transparency, Obj-ID, ..., Obj-IDn
#OOR	

Set visibility (**Transparency**) from 0 (completely transparent) to 100 (completely visible). Apply **Obj-ID** = 0 to all objects.

See also [objO\(id\)](#)

Change object color

#OCC	R,G,B, Obj-ID, ..., Obj-IDn
------	-----------------------------

Change the color channels **Red**, **Green** and **Blue**. The color channel of the target color is determined relative to the parameters passed. The parameters (R, G, B) are transferred as percentage values in the range from -100 to 100.

Example:

Assume that the output color should be changed from RGB (50.0.0) to RGB (200.0.0).

The target color has only changed in the red component. The difference in the red component is 150. The value has to be converted into a percentage value:

$$\frac{150}{100} = 1,5$$

#OCC 118,0,0,...

Color changes are always related to the initial-color (even with multiple use). **Obj-ID** =0 to all objects.

Change object style

#OCS Style-No, Obj-ID, ..., Obj-IDn

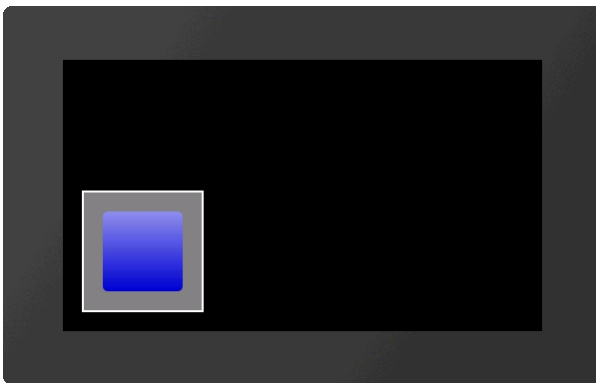
A new style is assigned to an object (**Obj-ID** = 0 all). The style depends on the object. A string e.g. is automatically assigned a TextStyle. The command can only be used on simple graphic objects (e.g. not on buttons, SpinBox, ...). Also monochrome pictures can be assigned a style once.

See also [objC\(id\)](#)

Define frame/background

#OFP DrawStyleNo, addX, addY, Obj-ID, ..., Obj-IDn

A background is assigned to an object (**Obj-ID** = 0 all). The colors are determined via the **DrawStyle**. The two parameters **addX** and **addY** change the size (in pixels) of the background on the left / right and upper / lower edge compared to the object.



...
#OFP 1, 20, 20, 1
...

Set anchor

#OAA Anchor, Obj-ID, ..., Obj-IDn

A new **Anchor** is assigned to an object (**Obj-ID** = 0 all). The active anchor is e.g. used to rotate the object.

See also [objA\(id\)](#)

Set free anchor absolut/relative

#OAS
#OAO x, y, Obj-ID, ..., Obj-IDn

Set the anchor 0 of an object (**Obj-ID**). The command also marks anchor 0 as active.

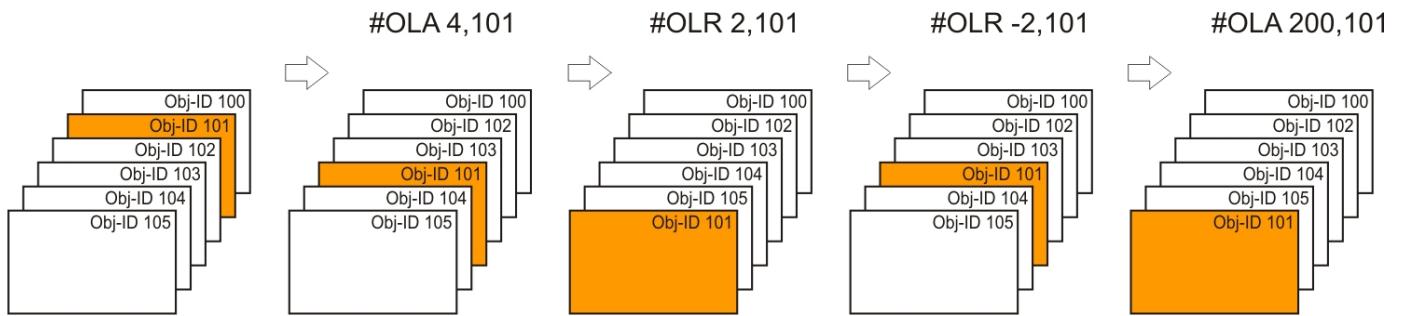
Change draw order (layer) absolut/relative

#OLA
#OLR Draw order, Obj-ID, ..., Obj-IDn

This command changes the drawing order of one or more objects. The object (**Obj-ID**) with the highest **Draw order** is drawn as a last resort.

The very first object will be put to "layer 1". Next objects will be drawn on a higher "layer" above. Those may cover up

prior drawn objects.



A group will be moved collectively. Its also possible to move objects inside of a group.

Set user value (integer)

#OUI Obj-ID, Value, ..., Value n (Obj-ID n)

An integer **Value** can be assigned to each object. The value can also be a calculation.

See also [objUI\(id\)](#)

Set user value (float)

#OUF Obj-ID, Value, ..., Value n (Obj-ID n)

A float **Value** can be assigned to each object. The value can also be a calculation.

See also [objUF\(id\)](#)

Group

Add object to group

#OGA Obj-ID Group, Obj-ID, ..., Obj-IDn

Create a group (**Obj-ID Group**) or add objects to an existing group.

Background

Move object to the background layer

#OBG RGB, Obj-ID, ..., Obj-IDn

Existing objects are moved to the background. The background color is specified by the parameter **RGB**. After the PowerOn reset, the background color is black (RGB = 0). If RGB = -1 is transferred, the previously set color remains unchanged.

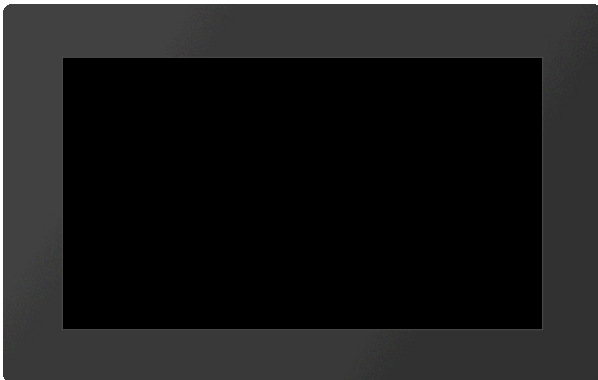
Load image into background layer

#OBP <Name>, x(0), y(0), Anchor(7),<Gradient>, Time, Direction, 'Endmacro'

The command places an image from the FLASH directly to the background. Transformations (like scaling) are

impossible. If transformations are necessary, an image object must be created (#PPP) and the transformations applied before the object is moved to the background with the #OBG command. The parameter <Gradient> specifies a grayscale image that is used for the transition. The cross-fading is determined by the gray values and the Time in 1/100 s. The transition effect can be shown forward or backward (Direction). After the crossfading the macro 'Endmacro' is called.

Direction	
0	Forward
1	Backward



```

...
#OBP
<P:picture/GrandCanyon.epg>
,0,0,7,<P:picture/Gradient.epg>,200,2
...

...
#OBP "GrandCanyon";0,0,7,"Gradient";200,2
...

```

Styles #C

Command group to create styles. The look of each object is based on a style appropriate to the object type. The maximum number of styles available for each style is 100.

DrawStyle

Delete filling (Style Fill Delete)	#CFD	DrawStyle-No.
Define filling width color (Style Fill Color)	#CFC	DrawStyle-No., RGB, Opacity(100)
Define filling with linear gradient (Style Fill Linear)	#CFL	DrawStyle-No, ColorRamp-No, Angle(0)
Define filling with radial gradient (Style Fill Radial)	#CFR	DrawStyle-No, ColorRamp-No, FocusX (5000), FocusY (0)
Define filling with conical gradient (Style Fill Conial)	#CFK	DrawStyle-No, ColorRamp-No, FocusX(5000), FocusY(0), Direction(1)
Define filling with pattern (Style Fill Pattern)	#CFP	DrawStyle-No, <PatternName>, 0, 0, 0, FocusX(5000), FocusY(0)
Change angle of linear gradient (Style Fill Angle)	#CFA	DrawStyle-No, Angle
Change gradient (Style Fill Gariant)	#CFG	DrawStyle-No, ColorRamp-No.
Change focus of gradient (Style Fill Focus)	#CFF	DrawStyle-No, FocusX, FocusY, PatternAnchor (no change)
Change pattern (Style Fill pattern Name)	#CFN	DrawStyle-No, <PatternName>
Delete line (Style Line Delete)	#CLD	DrawStyle-No.
Define line color and thickness (Style Line Style)	#CLS	DrawStyle-No, RGB, Opacity(100), Width(1), JoinStyle(0)
Change line color (Style Line Color)	#CLC	DrawStyle-No, RGB, Opacity (no change)
Change line with (Style Line Width)	#CLW	DrawStyle-No, Width
Change join style (Style Line End)	#CLE	DrawStyle-No, JoinStyle

TextStyle

Define TextStyle (Style Text Font)	#CTF	TextStyle-No, <FontName>, 0, Alignment(0), DrawStyle-No.(0), 0, LineSpace (0), CharacterSpace (0)
Change font (Style Text Name)	#CTN	TextStyle-No, <FontName>
Change alignment (Style Text Align)	#CTA	TextStyle-No, Alignment
Change DrawStyle (Style Text drawstyle)	#CTC	TextStyle, DrawStyle-No
Change spacing (Style Text Gap)	#CTG	TextStyle, LineSpace, CharacterSpace (no change)

Change space width (Style Text space Width)	#CTW	TextStyle-No, SpaceCode, SpaceWidth(100)
---	-------------	--

ButtonStyle

Define picture ButtonStyle (Style Button Picture)	#CBP	ButtonStyle-No, <ButtonNameNormal>, <ButtonNameDown> (=Normal), Width(0), Height(0), scale/pixels(0)
Define ButtonStyle (Style Button Drawstyle)	#CBD	ButtonStyle-No, DrawStyle-Normal, DrawStyle-Down (=Normal), Width (0), Height (0), Radius(0)
Define Text (Style Button Textstyle)	#CBT	ButtonStyle-No, TextStyleNormal, TextStyleDown (=Normal), OffsetX(0), OffsetY(0)
Define DownEvent (Style Button Offset)	#CBO	ButtonStyle-No, OffsetX(0), OffsetY (=OffsetX), Size(100), Angle(0)
Define sound for DownEvent (Style Button Sound)	#CBS	ButtonStyle-No, "Sound string"
Define disabled ButtonStyle (Style Button Greyout)	#CBG	R (-30), G (=R), B (=R), Opacity(0)

ColorRamp

Define ColorRamp (Style Color Ramp)	#CCR	ColorRamp-No, Offset1, RGB1, Transparency1, ... Offset10, RGB10, Transparency10
Animate ColorRamp (Style Animate Colorramp)	#CAC	ColorRamp-No, Type (1), Time (100)

DrawStyle

Delete filling

#CFD	DrawStyle-No.
-------------	---------------

This command deletes the filling of the DrawStyle (**DrawStyle No.**).

Define filling width color

#CFC	DrawStyle-No., RGB, Opacity(100)
-------------	----------------------------------

A full-color (**RGB**) fill is assigned to the DrawStyle (**DrawStyle No.**). The **Opacity** can be set as a percentage

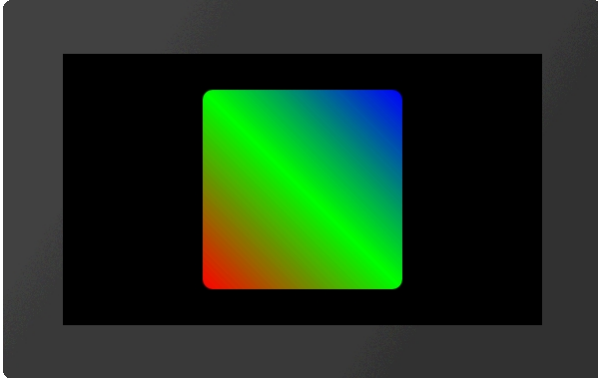


...
#CFC 15, \$3B7EAE
...

Define filling with linear gradient

#CFL	DrawStyle-No, ColorRamp-No, Angle(0)
------	--------------------------------------

The DrawStyle (**DrawStyle No.**) is assigned a linear gradient (**ColorRamp-No.**). The gradient must be defined in advance with the #CCR command. The orientation can optionally be specified (**Angle** in degrees). Exceptionally this command allows single degree steps.

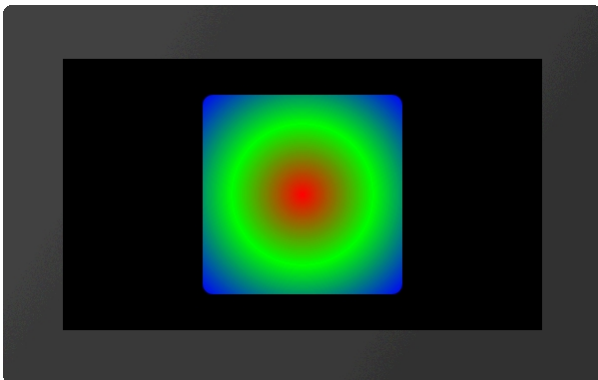


```
...
#CCR
5, 0, $FF0000, 100, 50, $00FF00, 100, 100, $0000FF, 100
#CFL 15, 5, 45
...
```

Define filling with radial gradient

#CFR	DrawStyle-No, ColorRamp-No, FocusX (5000), FocusY (0)
------	---

The DrawStyle (**DrawStyle No.**) is assigned a radial gradient (**ColorRamp-No.**). The gradient must be defined in advance with the #CCR command. The focus determines the starting point of the course as a percentage. With **FocusX** = 5000, the anchor to be used as the starting point of the gradient is specified with **FocusY**.



```
...
#CCR
5, 0, $FF0000, 100, 50, $00FF00, 100, 100, $0000FF, 100
#CFR 15, 5, 5000, 5
...
```

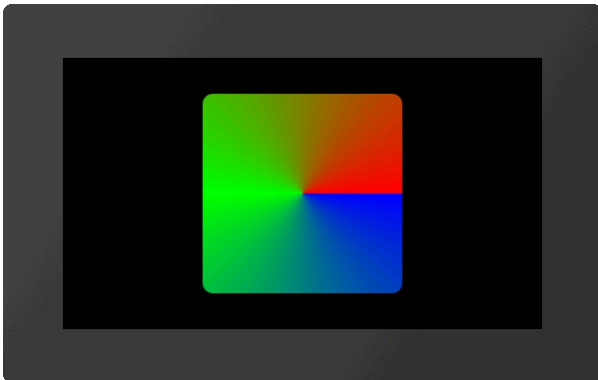
Define filling with conical gradient

#CFK	DrawStyle-No, ColorRamp-No, FocusX(5000), FocusY(0), Direction(1)
------	---

The DrawStyle (**DrawStyle No.**) is assigned a conical gradient (**ColorRamp-No.**). The gradient must be defined in advance with the #CCR command. The focus determines the starting point of the course as a percentage. With **FocusX** = 5000, the anchor to be used as the starting point of the gradient is specified with **FocusY**. The optional parameter **Direction** specifies the direction of rotation.

Direction	
0	Counterclockwise

1	Clockwise
---	-----------

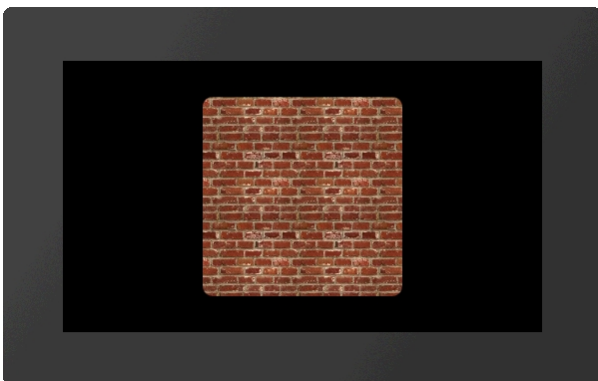


```
...
#CCR
5,0,$FF0000,100,50,$00FF00,100,100,$0000FF,100
#CFK 15,5,5000,5,0
...
```

Define filling with pattern

#CFP	DrawStyle-No, <PatternName>, 0, 0, 0, FocusX(5000), FocusY(0), PatternAnchor(1)
------	---

A pattern (<PatternName>) is used as a fill for the DrawStyle (**DrawStyle No.**). The focus determines the percentage of the starting point of the pattern. With **FocusX** = 5000, the anchor to be used as the starting point of the pattern is specified with **FocusY**. The pattern is set directly to the focus point .



```
...
#CFP 15,<P:pattern/Brick.epg>,40
...

...
#CFP 15,"Brick";40
...
```

Change angle of linear gradient

#CFA	DrawStyle-No, Angle
------	---------------------

The **Angle** of a linear gradient is changed. Applies only to a linear gradient and redrawing of the object.

Change gradient

#CFG	DrawStyle-No, ColorRamp-No.
------	-----------------------------

A new gradient is assigned to the DrawStyle (**ColorRamp-No**)

Change focus of gradient

#CFF DrawStyle-No, FocusX, FocusY, PatternAnchor (no change)

The focus determines the starting point of the course or the pattern as a percentage. With **FocusX** = 5000, the anchor to be used as the starting point of the course is specified with **FocusY**. The last parameter (**PatternAnchor**) is only necessary for patterns: The pattern is set directly to the focus point with the PatternAnchor.

Change pattern

#CFN DrawStyle-No, <PatternName>

A new pattern (<**PatternName**>) will be assigned to the filling.

Delete line

#CLD DrawStyle-No

This command deletes the line of the DrawStyle (**DrawStyle No.**).

Define line color and thickness

#CLS DrawStyle-No, RGB, Opacity(100), Width(1), JoinStyle(0)

Der Befehl definiert die Linienfarbe (**RGB**), die Deckkraft (**Transparenz** in Prozent), sowie die Linien-**Dicke** in Pixeln. Der Parameter **Verbindung** bestimmt die Art des Linienendes bzw, die Verbindung zweier Linien:
The command defines the line color (**RGB**), the **Opacity** (in percent) and the line **Width** in pixels. The **JoinStyle** parameter determines the type of line end or the connection of two lines:

JoinStyle	
0	Miter
1	Round

Change line color

#CLC DrawStyle-No, RGB, Opacity (no change)

Assign a new color (**RGB**) to the line.

Change line width

#CLW DrawStyle-No, Width

Change the thickness of the line.

Change join style

#CLE DrawStyle-No, JoinStyle

Change the join style of the line

JoinStyle

0	Mitered
1	Rounded

TextStyle

Define TextStyle

#CTF	TextStyle-No, <FontName>, 0, Alignment(0), DrawStyle-No.(0), 0, LineSpace (0), CharacterSpace (0)
------	---

Definition of a TextStyle with font (<FontName>), and **Alignment**.

Alignment	
0	Left
1	Center
2	Right

The **DrawStyle** specifies the color. For performance reasons, we recommend simple filling without an outline. The remaining two parameters specify the **LineSpacing** and additional **CharacterSpacing**.

Change font

#CTN	TextStyle-No, <FontName>
------	--------------------------

The command changes the font (<FontName>) of the TextStyle.

Change alignment

#CTA	TextStyle-No, Alignment
------	-------------------------

The command changes the **Alignment** of the text.

Alignment	
0	Left
1	Center
2	Right

Change DrawStyle

#CTC	TextStyle, DrawStyle-No
------	-------------------------

Change color using the DrawStyle (**DrawStyle No.**).

Change spacing

#CTG	TextStyle, LineSpace, CharacterSpace (no change)
------	--

An additional **LineSpacing** or **CharacterSpacing** is defined (in % of the character height). Negative values are also allowed.

Change space width

#CTW	TextStyle-No, SpaceCode, SpaceWidth(100)
------	--

The width of the space can be taken from any other code (**SpaceCode**). The width can also be defined in %: Standard: 100 (**SpaceWidth**).

ButtonStyle

Define picture ButtonStyle

#CBP	ButtonStyle-No, <ButtonNameNormal>, <ButtonNameDown> (=Normal), Width(0), Height(0), scale/pixels(0)
------	--

The command defines a ButtonStyle: Display two images for the unpressed (<**ButtonNameNormal**>) and pressed (<**ButtonNameDown**>) state. The size is determined by **Width** and **Height** (= 0 original size). The last parameter **scale/pixel** indicates whether the image should be scaled (=0) or whether the pixels are repeated in the middle of the image (=1 frame magnification)

Define ButtonStyle

#CBD	ButtonStyle-No, DrawStyle-Normal, DrawStyle-Down (=Normal), Width (0), Height (0), Radius(0)
------	--

The command defines a ButtonStyle: Display of two DrawStyles for the unpressed (**DrawStyleNormal**) and pressed (**DrawStyleDown**) state. The following are further parameters for the **Width** and **Height** of the button and the corner rounding (**Radius**).

Define text

#CBT	ButtonStyle-No, TextStyleNormal, TextStyleDown (=Normal), OffsetX(0), OffsetY(0)
------	--

Define the text of the button style. The **Offset** specifies an additional distance in pixels where the text is positioned on the button.

Define DownEvent

CBO	ButtonStyle-No, OffsetX(0), OffsetY (=OffsetX), Size(100), Angle(0)
-----	---

The behaviour of the button when pressed is defined. The button is drawn with the **Offset** (in pixels). The **Size** changes proportionally as a percentage. The **Angle** (in degrees) can also be changed.

Define sound for DownEvent

CBS	ButtonStyle-No, "Sound string"
-----	--------------------------------

A short tone sequence ("**Sound string**") is played in the DownEvent of the ButtonStyle. If the parameter "**Sound string**" is empty, the jingle is deleted.

Define disabled ButtonStyle

#CBG R (-30), G (=R), B (=R), Opacity(0)

The deactivated state of a button is the percentage change in color of the ButtonStyle normal. Each color channel can be addressed individually. The **Opacity** can also be changed.

ColorRamp

Define ColorRamp

#CCR ColorRamp-No, Offset1, RGB1, Transparency1, ... Offset10, RGB10, Transparency10

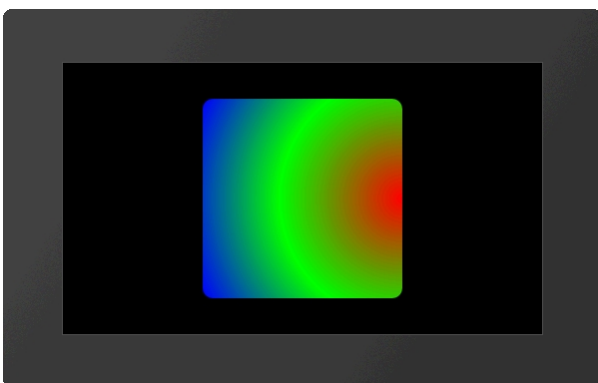
The command defines a gradient. The base point (**Offset**) defines the color point in the course in percent, the color is indicated by **RGB** and **Opacity**. A maximum of 10 control points can be specified.

Animate ColorRamp

#CAC ColorRamp-No, Type (1), Time (100)

The position of the vertices of the gradient are changed. The type specifies the animation type. **Time** in 1/100 s indicates the duration.

Type	
0	Stop animation
1	Cyclic
2	Cyclic backward
3	PingPong
4	PingPong backward



```
...
#CCR 5, 0, $FF0000, 100, 50, $00FF00, 100, 100, $0000FF, 100
#CFR 15, 5, 5000, 6
#CAC 5, 3
...
```

Macros #M

Single or multiple command sequences can be collected in a so-called macro (*.emc) and stored in internal FLASH memory. A macro could also contain lots of commands to build up a complete screen - including a command that deletes all old objects (#ODIO).

Run macros

Run normal macro / Start a screen (Macro Run Normal)	#MRN	<Macroname>
Run normal macro conditionally (Macro Run Conditionally)	#MRC	(Condition), <MacronameTrue>, <MacronameFalse>
Run normal macro delayed (Macro Run Delayed)	#MRD	Delay-No., Time, <Macroname>
Run I/O-Port macro (Macro Run Port)	#MRP	Port
Run I/O-Bit macro (Macro Run Bit)	#MRB	Portpin, Edge
Run analogue macro (Macro Run Analogue)	#MRA	Channel, Type
Run touch macro (Macro Run Touch)	#MRT	Obj-ID, Type, Point-No.(0)

Define macros

Define touch macro (Macro Define Touch)	#MDT	Obj-ID, <MacronameDown>, <MacronameUp>; <MacronameDrag>
Define macro process (Macro Process Define)	#MPD	Process-No, Time, <Macroname>, StartNumber(no), EndNumber (StartNumber), Type(1)
Define conditional macro process (Macro Process Conditionally)	#MPC	Process-No, Time, (Condition), <Macroname>, StartNumber(no), EndNumber (StartNumber), Type(1)
Define automatic macro process (Macro Process Autochange)	#MPA	Process-No, Time, (Calculation), <Macroname>, StartNumber(no), EndNumber (StartNumber), Type(1)
Change macro process time (Macro Process Time)	#MPT	Process-No, Time
Define action end macro (Macro Define Actionend)	#MDA	Obj-ID, <Macroname>
Define I/O/Port macro (Macro Hardware Port)	#MHP	Port
Define I/O-Bit macro (Macro Hardware Bit)	#MHB	Portpin, Edge, <Macroname>
Define analogue macro (Macro Hardware Analogue)	#MHA	Channel, Type, <Macroname>
Define RS232 receive macro (Macro Hardware RS232 master)	#MHR	BufferSize, <Macroname>
Define second macro (RTC) (Macro Define Second)	#MDS	<Macroname>
Define sound end macro (Macro Hardware Soundend)	#MHS	<Macroname>
Define backlight auto-dimming macro	#MDL	<Macroname>

(Macro Define Led)		
Define gesture macro (Macro Define touch Gesture)	#MDG	Obj-ID, <MacronameDoubleClick>, <MacronameLongClick>
Delete macro definition (Macro Clear Defines)	#MCD	Mask

Commands within macros

Skip commands (Macro File Skip)	#MFS	(Condition), Commands(1)
Jump (Macro File Jump)	#MFJ	(Condition), Marker-No(0), Delete(0)
Set jump destination (marker) (Macro File Marker)	#MFM	Marker-No(0)
Jump with call (Macro File Call)	#MFC	(Condition), Marker-No(0), Delete(0)
Jump to call (Macro File Return)	#MFR	(Condition) (true)
Exit macro (Macro File Exit)	#MFE	(Condition) (true), <Macroname>
Delete marker (Macro File Delete)	#MFD	Marker-No.

Run macros

Run normal macro / Show a screen

#MRN	<Macroname>
-------------	-------------

This command executes a macro ().

```

...
#MRN <P:macro/macro.emc>
...
...
#MRN
"macro" ;
...

```

Alternatively this command runs a macro that set-up a full screen.

```

...
#MRN <P:macro/screen/Screen1.emc>
...
...
#MRN
"screen/Screen1" ;
...

```

Run normal macro conditionally

#MRC	(Condition), <MacronameTrue>, <MacronameFalse>
-------------	--

If the **Condition** is true, <MacronameTrue> is executed, otherwise <MacronameFalse>.

```

...
#MRC (R0<10) , <P:macro/macroTRUE.emc> , <P:macro/macroFALSE.emc>
...
#MRC

```

```

...
(R0
<10), "mac
roTRUE"
; "macroFA
LSE" ;
...

```

Run normal macro delayed

#MRD	Delay-No., Time, <MacroName>
-------------	------------------------------

The command executes the macro (<MacroName>) with a delay. Up to 10 macros can be started at the same time with a delay (**Delay-No.** 1 ... 10). The **Time** is given in 1/100 s.

```

...
#MRD 1,100,<P:macro/macro.emc>
...
...
#MRD
1
,100
,"macro";
...

```

Run I/O-Port macro manually

#MRP	Port
-------------	------

The command executes a port macro (**Port** 0 ... 16).

Run I/O-Bit macro manually

#MRB	Portpin, Edge
-------------	---------------

The command executes a bit macro (**Portpin** 0 ... 136).

Edge	
0	Falling
1	Rising

Run analogue macro manually

#MRA	Channel, Type
-------------	---------------

The command executes an analog macro (**Channel** 0 ... 3). The parameterization of the analog input ([limits](#), [hysteresis](#)) is set with the command group '[Analog input](#)'.

Type	
0	Every change
1	Decrement
2	Increment

3	Lower limit 1
4	Upper limit 1
5	Lower limit 2
6	Upper limit 2
7	Leave window
8	Enter window

Run touch macro

#MRT	Obj-ID, Type, Point-No.(0)
-------------	----------------------------

The command executes a touch macro (object **Obj-ID**). **PointNo.** (0 ... 4) indicates the finger: 0 = first, 1 = second etc. contact point.

Type	
1	Normal
2	Pushed
3	Drag

Define macros

Define touch macro

#MDT	Obj-ID, <MacronameDown>, <MacronameUp>; <MacronameDrag>
-------------	---

The command defines a touch macro. The macro **<MacronameDown>** is called when the key is pressed, **<MacronameUp>** when released, **<MacronameDrag>** when dragging (especially useful for bar-graphs and instruments). With an empty string ("";) the corresponding macro is deleted.

Modules with capacitive touch panels also support multi-finger operation. Up to 5 points are recognized. The first three macro names then apply to the first point, the next three to the second, etc. If no special macro is defined, the macros for the first point are always called.

```
...
#MDT 1, <P:macro/macroDOWN.emc>, <P:macro/macroUP.emc>, <P:macro/macroDRAG.emc>
...
```

.
.
.

M
D
T
1
,
"
m
a
c
r

Define macro process

#MPD	Process-No, Time, <Macroname>, StartNumber(no), EndNumber (StartNumber), Type(1)
------	--

Macro processes define an automatic chronological sequence of macros. The process (**Process-No.** 1 ... 10) automatically calls the next macro (<**Macroname**>) in (**Time** in 1/100 s). Several macros can be called (**StartNumber** to **EndNumber** e.g. #MPD 1,100,"MacroProcess";1,50 MacroProcess1 .. MacroProcess50) are called. The **Type** specifies the call order:

Type	
1	Cyclic
2	Cyclic backward
3	PingPong
4	PingPong backward
5	Once
6	Once backward

```
...
#MPD 1,100,<P:macro/MacroProcess.emc>,1,4
...
```

.
. .

M
P
D

1
,
1
0
0
,
"
M
a
c
r
o
P
r
o
c
e
s
s
"
;
1
,
4
.
.
.

Define conditional macro process

#MPC	Process-No, Time, (Condition), <Macroname>, StartNumber(no), EndNumber (StartNumber), Type(1)
-------------	---

Conditional macro processes define an automatic chronological sequence of macros if a condition is fulfilled (true). The process (**Process-No.** 1 ... 10) automatically calls the next macro (<**Macroname**>) in (**Time** in 1/100 s). Several macros can be called (**StartNumber** to **EndNumber** e.g. `#MPC 1,100,(R1<10),"MacroProcess";1,50`)

MacroProcess1 .. MacroProcess50 are called. The type indicates the order of the call:

Type	
1	Cyclic
2	Cyclic backward
3	PingPong
4	PingPong backward
5	Once
6	Once backward

```
...
#MPC 1,100,(R1<10),<P:macro/MacroProcess.emc>
...
```

```
.
.
.
#
M
P
C
1
,
1
0
0
,
(
R
1
<
1
0
)
)
,
"
M
a
c
r
o
P
r
o
c
e
s
s
"
;
.
.
.
```

Define automatic macro process

#MPA	Process-No, Time, (Calculation), <Macroname>, StartNumber(no), EndNumber (StartNumber), Type(1)
------	---

Conditional macro processes define an automatic chronological sequence of macros if the value of the calculation has changed. The process (**Process-No.** 1 ... 10) automatically calls the next macro (<**Macroname**>) in (**Time** in 1/100 s). Several macros can be called (**StartNumber** to **EndNumber** e.g. #MPA 1, 100, (R1<10), "**MacroProcess**";1, 50 MacroProcess1 .. MacroProcess50 are called. The type indicates the order of the call:

Type	
1	Cyclic
2	Cyclic backward
3	PingPong
4	PingPong backward
5	Once
6	Once backward

```

...
#MPA 1,100,(R1),<P:macro/MacroProcess.emc>
...

```

.
 .
 .
 #
 M
 P
 A

 1
 ,
 1
 0
 0
 ,
 (
 R
 1
)
 ,
 "
 M
 a
 c
 r
 o
 p
 r
 o
 c
 e
 s
 s
 "
 ;

Change macro process time

#MPT Process-No, Time

The time (1/100 s) for the macro process (**Process-No** = 0 all) is changed.

Time	
-1	Restart with old interval
0	Stop
>0	Reset time

Define action end macro

#MDA Obj-ID, <Macroname>

After an object animation (**Obj-ID**) has ended, the macro (<**Macroname**>) is called.

```

...
#MDA 1,
...
...
#MDA
1
, "Macro";
...

```

Define I/O-Port macro

#MHP Port

The port macro is called when the status of the **Port** (0 ... 15) changes.

```

...
#MHP 0, <P:macro/Macro.emc>
...
...
#MHP
0
, "Macro";
...

```

Define I/O-Bit macro

#MHB Portpin, Edge, <Macroname>

The bit macro is called when an **Edge** is detected at the **Portpin** (0 ... 127).

Edge	
0	Falling

1	Rising
2	Both edges

```

...
#MHB 16,2,<P:macro/Macro.emc>
...
...
#MHB
16
,
2
,"Macro";
...

```

Define analogue macro

#MHA	Channel, Type, <Macroname>
------	----------------------------

A macro (<Macroname>) is assigned to an A / D input (**Channel** 0 ... 3). **Type**:

Type	
0	Every change
1	Decrement
2	Increment
3	Lower limit 1
4	Upper limit 1
5	Lower limit 2
6	Upper limit 2
7	Leave window
8	Enter window

The parametrization of the analogue input ([limits](#), [hysteresis](#)) is set with the command group '[Analog input](#)'.

```

...
#MHA 0,0,<P:macro/Macro.emc>
...
...
#MHA
0
,
0
,"Macro";
...

```

Define RS232 receive macro

#MHR	BufferSize<Macroname>
------	-----------------------

The macro (<Macroname>) is called when the **BufferSize** (0 = disable) in the master RS232 receive buffer is reached.

```

...
#MHR 42, <P:macro/Macro.emc>
...
...
#MHR
42
, "Macro" ;
...

```

Define second macro (RTC)

```
#MDS <Macroname>
```

The macro (<Macroname>) is called every second.

```

...
#MDS <P:macro/Macro.emc>
...
...
#MDS
"Macro" ;
...

```

Define sound end macro

```
#MHS <Macroname>
```

The macro (<Macroname>) is called when the sound string has finished playing. The definition must be made before playing the jingle. The macro is not called for automatically played jingles (e.g. button) (only after [#HTN](#))

```

...
#MHS <P:macro/Macro.emc>
...
...
#MHS
"Macro" ;
...

```

Define backlight auto-dimming macro

```
#MDL <Macroname>
```

The automatic dimming function of the backlight calls up the specified macro (<Macroname>) when the state changes. Please see command [#XAL](#) for parameter setting concerning time and brightness.

```

...
#MDL <P:macro/Macro.emc>
...
...
#MDL
"Macro" ;
...

```

Define gesture macro

```
#MDG Obj-ID, <MacronameDoubleClick>, <MacronameLongClick>
```

The command defines a gesture macro. The macro <MacronameDoubleClick> is called with a double click, the macro <MacronameLongClick> with a long click

```

...
#MDG 1, <P:macro/MacroDoubleClick>, <P:macro/MacroLongClick>
...
...
#MDG
1
, "MacroDo

```

```
ableClick
"
; "MacroLo
ngClick" ;
...
```

Delete macro definition

#MCD Mask

The command deletes macro definitions by type:

Mask	
0	Second macros (RTC)
2	Process macros
4	Port macros
8	Bit macros
16	Analogue macros
32	Touch/Gesture macros
64	Action macros
128	Delayed macros
256	Backlight macro
512	RS232 receive macro
1024	Sound end macro

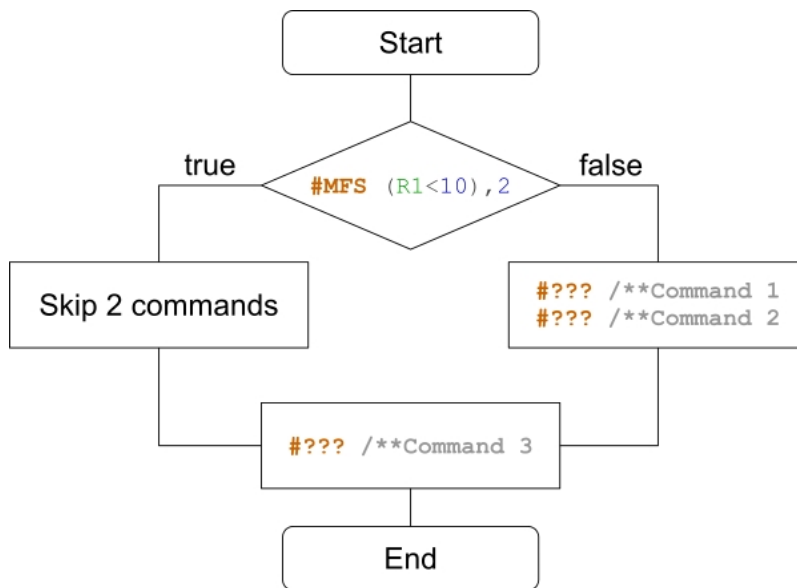
The individual types can be added, e.g. Delete all macros: **Mask** = \$ FFFF

Commands within macros

Skip commands

#MFS (Condition), Commands(1)

If the **Condition** is true, the command skips the defined number of **Commands** (blank lines and comments are ignored) in the macro.



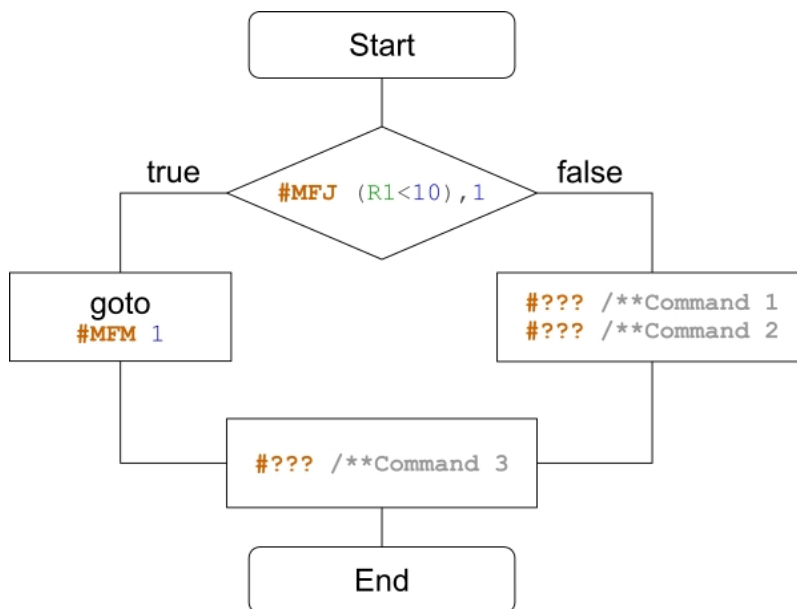
```

...
#MFS (R1<10), 2
#??? /**Comman
d 1
#??? /**Comman
d 2
#??? /**Command 3
...
  
```

Jump

#MFJ	(Condition), Marker-No(0), Delete(0)
------	--------------------------------------

If the **Condition** is true, the command jumps to the marker (**Marker-No.** 0..99) in the macro. A marker can appear multiple times in a macro. The parameter **Delete** deletes the last marker found and searches for the next marker with the same ID in the macro.



```

...
#MFJ (R1<10), 1
#??? /**Comman
d 1
#??? /**Comman
d 2
#MFM 1
#??? /**Command 3
...
  
```

Set jump destination (marker)

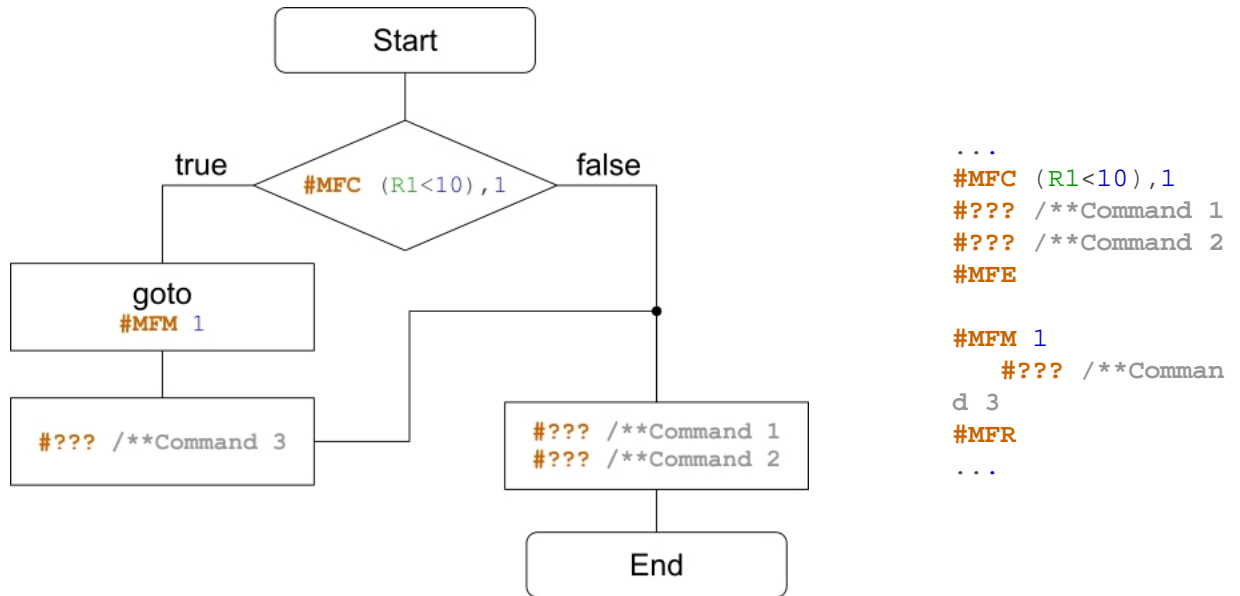
#MFM	Marker-No(0)
------	--------------

The command sets a jump target (**Marker-No.** 0..99) in the macro.

Jump with call

#MFC (Condition), Marker-No(0), Delete(0)

If the **Condition** is true, the command jumps to the marker (**Marker-No.** 0..99) in the macro. A marker can appear multiple times in a macro. A return (**#MFR**) is mandatory to return to the call. The parameter **Delete** deletes the last marker found and searches for the next marker with the same ID in the macro.



Jump to call

#MFR (Condition) (true)

If the **Condition** is true, the command jumps to the call.

Exit macro

#MFE (Condition) (true), <Macroname>

If the **Condition** is true, the macro is exited. Another macro (<**Macroname**>) can be called optionally.

Delete marker

#MFD Marker-No.

The command deletes the last marker with the **Marker-No.**.

Comparison between C-Code and Macro-Code

if-query one-line	
C-Code	Makro-Code
<pre>if(R1<10) //Command else //Command</pre>	<pre>#MFS (R1>=10),3 ???? /**Command #MFS (R1<10),1 ???? /**Command</pre>

if-query multi-line	
C-Code	Makro-Code
<pre>if(R1<10) { //Command 1 //Command 2 } else { //Command 1 //Command 2 }</pre>	<pre>#MFJ (R1>=10),1 ???? /**Command 1 ???? /**Command 2 #MFJ (1),2 #MFM 1 ???? /**Command 1 ???? /**Command 2 #MFM 2</pre>

for-loop	
C-Code	Makro-Code
<pre>for(int i=0;i<10;i++) { //Command 1 //Command 2 }</pre>	<pre>#VRI 0,0 #MFM 1 ???? /**Command 1 ???? /**Command 2 #MFJ (++R0<10),1</pre>

do-loop	
C-Code	Makro-Code
<pre>do { //Command 1 //Command 2 }while(R1<10)</pre>	<pre>#MFM 1 ???? /**Command 1 ???? /**Command 2 #MFJ (R0<10),1</pre>

Function call	
C-Code	Makro-Code

<pre> ... { subfunction(); //Command 1 //Command 2 } void subfunction() { //Function Command 1 //Function Command 2 } </pre>	<pre> #MFC 1,1 #??? /**Command 1 #??? /**Command 2 #MFE /**----- subfunction----- - #MFM 1 #??? /**Function Command 1 #??? /**Function Command 1 #MFR </pre>
--	--

Variables / register #V

Command group to execute calculations and logical operations. With the help of the string files, internationalization (multiple languages) can be realized. There are registers for numbers and strings (can record characters up to 200), integer registers use signed 32-bit, floating-point registers use 23-bit mantissa, 8-bit exponent, 1-bit signed.

String file / multilingualism

Load string file (Variable stringFile Load)	#VFL	StringfileName>
Delete string file (Variable stringFile Delete)	#VFD	<StringfileName> (all)
Send number of loaded string files (Variable stringFile Count)	#VFC	

String register

Set string register (Variable Stringregister Set)	#VSS	String-ID, "String"; "String" [ID+1]; ...
Set string register from position (Variable Stringregister Postion)	#VSP	String-ID, Offset, "New String";
Replace string register from position (Variable Stringregister Replace)	#VSR	String-ID, Offset, "New String";
Cut out and replace sub-string from string register (Variable Stringregister Truncate)	#VST	String-ID, Offset, Number(until end)
Copy sub-string from string register (Variable Stringregister Copy)	#VSC	String-ID Target, String-ID Source, Offset (0), Number(until end)
split string registers in sub-strings (Variable Stringregister dElimiter)	#VSE	String-ID Target Start, String-ID Source, Seperator, Register-ID (=String-ID Target Start)
Set string register with date/time (Variable Stringregister Date)	#VSD	String-ID, "Dateformat"; date (act. time)
Set formatted string register (Variable Stringregister Formated)	#VSF	String-ID, "Formatted string"; Value, Value2, ..., ValueN
Read object strings (Variable Stringregister Object)	#VSO	String-ID, Obj-ID, ...
Send string register (ASCII) (Variable string Send Ascii)	#VSA	String-ID, ...
Send string register (Unicode) (Variable string Send Unicode)	#VSU	String-ID, ...
Sort string register (Variable Quicksort Strings)	#VQS	String-ID Start, String-ID End, Number (0), Offset(0)
Sort codes in string register (Variable Quicksort Codes)	#VQC	String-ID, Direction (1), Number (0), Offset (0)
Last error message in string register (Variable Stringregister Last error)	#VSL	String-ID, Delete(1)
Mix string register (Variable Mix Strings)	#VMS	String-ID Start, String-ID End
Mix codes in string register (Variable Mix Codes)	#VMC	String-ID, Number (0), Offset (0)

Register

Set register (integer) (Variable Register Integer)	#VRI	Register-ID, Value, Value1 [ID+1], ...
Set register (float) (Variable Register Float)	#VRF	Register-ID, Value, Value1 [ID+1], ...
Convert object string (Variable Register Object)	#VRO	Register-ID, Obj-ID, Obj-ID1[ID+1], ...
Convert string register to register (Variable Register dElimiter)	#VRE	Register-ID Start, String-ID Source, Seperator, Register-ID Name
Write register to RTC-RAM (Variable Register rtc Write)	#VRW	ID, Register-ID, Register-ID1, ...
Read register from RTC-RAM (Variable Register rtc Read)	#VRR	ID, Register-ID, Register-ID1, ...
Convert string register as calculation to register (integer) (Variable Calculate Integer)	#VCI	Register-ID, String-ID, String-ID1[ID+1], ...
Convert string register as calculation to register (float) (Variable Calculate Float)	#VCF	Register-ID, String-ID, String-ID1[ID+1], ...
Send register (Variable Register Send)	#VRG	Register-ID, ...
Sort register (Variable Quicksort Register)	#VQR	Register-ID Start, Register-ID End
Mix register (Variable Mix Register)	#VMR	Register-ID Start, Register-ID End

Array

Define array (Integer) (Variable Array Integer)	#VAI	Array-ID, Number, Type(0)
Define array (Float) (Variable Array Float)	#VAF	Array-ID, Number, Type(0)
Delete array (free memory) (Variable Array Delete)	#VAD	Array-ID
Fill array (Variable Array Set)	#VAS	Array-ID, Value(0, all Elements)
Assign values to array elements (with index) (Variable Array Value)	#VAV	Array-ID, Index, Value, Value[Index+1], ...
Assign values to array elements (with current write pointer) (Variable Array Write)	#VAW	Array-ID, Value 1, Value 2, ...
Set writing and/or reading pointer (Variable Array Pointer)	#VAP	Array-ID, WritePointer, ReadPointer(-1)
Sort array (Variable Quicksort Arrays)	#VQA	Array-ID, StartIndex, EndIndex(last Index)
Mix array (Variable Mix Arrays)	#VMA	Array-ID, StartIndex, EndIndex(last Index)

String file / multilingualism

"Hello World" is placed in 4 different languages. It must be ensured that the selected font supports all necessary

characters. In the following example "Arial Unicode MS" was used. The commands below assume that the string files (Chinese.txt, English.txt, Cyrillic.txt and German.txt) are already available on FLASH in the project path in the string's subfolder:

Chinese.txt

HELLO="你好，世界"

English.txt

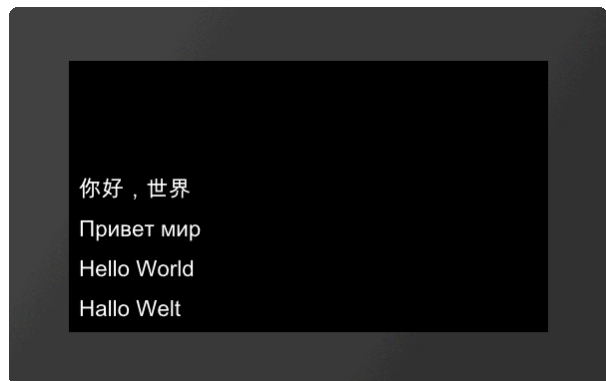
HELLO="Hello World"

Cyrillic.txt

HELLO="Привет мир"

German.txt

HELLO="Hallo Welt"



```

...
#VF
L
<P:
str
ing
/Ge
rma
n.t
xt>

#SS
P
1
,
1
,10
,10
,
7
,!H
ELL
O!;
#VF
D

#VF
L
<P:
str
ing
/En
gli
sh.
txt
>
#SS
P
2
,
1
,10
,50
,
7
,!H
ELL
O!;
#VF
D

```

```
#VF
L
<P:
str
ing
/Cy
ril
lic
.tx
t>

#SS
P
3
,
1
,10
,90
,
7
,!H
ELL
O!;
#VF
D
#VF
L
<P:
str
ing
/Ch
ine
se.
txt
>
#SS
P
4
,
1
,10
,13
0
,
7
,!H
ELL
O!;
#VF
D
...
```

Load string file

#VFL <StringfileName>

Load a set of strings. A maximum of 1000 strings from 8 different files can be loaded at the same time.

Delete string file

#VFD <StringfileName> (alle)

Delete a set of strings or all. The files are physically retained on FLASH so that they can be reloaded.

Send number of loaded strings

#VFC

Places the number of loaded strings in the [send buffer](#). The feedback is structured as follows:

ESC	V	F	C	Number	...
\$1B	\$56	\$53	\$43	16-Bit value	...

1Bh 56h 53h 43h 04h 00h

...
#VFL ...
#VFL ...
#VFL ...
#VFL ...
#VFC
...

String register

Set string register

#VSS String-ID, "String"; "String" [ID+1]; ...

The command saves the **String** in the register set (**String ID** [0 ... 499]).

String-ID	Value	...
0	"Hello World"	...
1	"Test Hello World"	#VSS 0, "Hello World"; "Test "S0;S0" Test";
2	"Hello World Test"	...

Set string register from position

#VSP String-ID, Offset, "New String";

The string of the location **String-ID** is deleted from the position offset and the new data ("**New String**") are added.

String-ID	Value	
0	"Hello Test"	#VSS 0, "Hello World";
1	...	#VSP 0, 6, "Test";
2

Replace string register from position

#VSR String-ID, Offset, "New String";

The string of the location **String-ID** is replaced with the new data ("New String") from the position offset.

String-ID	Value	
0	"Hello Test"	#VSS 0, "Hello World";
1	...	#VSR 0, 6, "Test";
2

Cut out and replace sub-string from string register

#VST String-ID, Offset, Count(until end)

Delete the "left" part of the string register and move the part from **offset** to offset + **count** to the front. If **count** is negative, count is taken as second offset. It is then an area specification.

String-ID	Value	
0	"World"	#VSS 0, "Hello World"; #VST 0, 6, 5
1
2	...	#VSS 0, "Hello World"; #VST 0, 6-10
		...

Copy sub-string from string register

#VSC String-ID Target, String-ID Source, Offset (0), Count(until end)

Copy a substring from the string (**String-ID Source**), starting with the **Offset** and length **Count**, and paste it into another string register (**String ID Target**).

String-ID	Value	
0	"Hello World"	#VSS 0, "Hello World"; #VSC 1, 0, 6, 5
1	"World"	...
2	...	#VSS 0, "Hello World"; #VSC 1, 0, 6-10
		...

Split string register in sub-strings

#VSE String-ID Target Start, String-ID Source, Seperator, Register-ID (=String-ID Target Start)

The string (**String-ID Source**) is split into substrings. The substrings are stored from the **String ID Target Start**. The number of substrings is stored in the **Register-ID**. The **Seperator** parameter specifies the separator.

String-ID	Value
0	"Entry1,Entry2,Entry3"
1	"Entry1"
2	"Entry2"
3	"Entry3"

```

...
#VSS 0, "Entry1, Entry2, Entry3";
#VSE 1, 0, ?, , 10
...

```

Register-ID	Value
10	3

Set string register with date/time

#VSD String-ID, "Dateformat"; date (act. time)

The time is stored in the string register as a formatted string. The presentation is based on the **Dateformat**. The structure is described in more detail in the sub-chapter [Date formats](#).

String-ID	Value
0	...
1	"14:59:30"
2	...

```

...
#VSD 1, "%h:%m:%s";
...

```

Set formatted string register

#VSF String-ID, "Formatted string"; Value, Value2, ..., ValueN

A formatted string is stored in the string register (**String-ID**). If the variable set is repeated, the format string is used again and stored in String-ID + 1

String-ID	Value
0	...
1	"Analog 3420"
2	...

```

...
#VSF 1, "Analog %d"; (analog(0))
...

```

Read object strings

#VSO String-ID, Obj-ID, ...

Object strings (**Obj-ID**) are stored in the string register (**String-ID**). This function is mainly used for EditBoxes.

String-ID	Value	...
0	...	<code>#SED 1, "edit me" /**Default text for</code>
1	"edit me"	<code>EditBox</code>
2	...	<code>#VSO 1,1</code>
		...

Send string register (ASCII)

#VSA String-ID, ...

Place the content of the string register (ASCII formatted) in the [send buffer](#). The feedback is structured as follows:

ESC	V	S	A	String-ID	Length	Char 1	Char 2	...	Char n	...
\$1B	\$56	\$53	\$41	16-Bit value	16-Bit value	8-Bit value	8-Bit value	8-Bit value	8-Bit value	...

```
1Bh 56h 53h 41h 00h 00h 03h 00h 73h 74h 72h ...
#VSS0, "str";
#VSA 0
...
```

Send string register (Unicode)

#VSU String-ID, ...

Place the content of the string register (Unicode formatted) in the [send buffer](#). The feedback is structured as follows:

#	V	S	U	String-ID	Length	Char 1	Char 2	...	Char n	...
\$1B	\$56	\$53	\$55	16-Bit value	16-Bit value	16-Bit value	16-Bit value	16-Bit value	16-Bit value	...

```
1Bh 56h 53h 55h 00h 00h 03h 00h 73h 00h 74h 00h 72h 00h ...
#VSS0, "str";
#VSU 0
...
```

Sort string register

VQS String-ID Start, String-ID End, Number (0), Offset(0)

The area of the string register (**String-ID Start** to **String-ID End**) is sorted. **Number** specifies the area that is considered for the sorting, with number = 0 the entire length is examined. **Offset** specifies the position in the string where the sorting begins.

String-ID	Value before	Value after	
0	"Sort"	"Entry"	...
1	"Test"	"Exit"	<code>#VSS 0, "Sort"; "Test"; "Entry"; "Exit";</code>
2	"Entry"	"Sort"	<code>#VQS 0,3,2,0</code>
3	"Exit"	"Test"	...

Sort codes in string register

VQC String-ID, Direction (1), Number (0), Offset (0)

Codes within the string register (**String-ID**) are sorted. **Number** specifies the area that is considered for the sorting, with number = 0 the entire length is examined. **Offset** specifies the position in the string from which the sorting begins. The direction can also be specified:

Direction	
0	Descending
1	Ascending

String-ID	Value	
0
1	" HWdellloor"	<code>#VSS 1, "Hello World"</code>
2	...	<code>#VQC 1</code>
		...

Last error message in string register

#VSL String-ID, Delete(1)

Save the error messages from the terminal in a string register (**String-ID**). The parameter **Delete** specifies the deletion behavior of the error message:

Delete	
0	Do not delete
1	Delete

Mix string register

#VMS String-ID Start, String-ID End

The content of the registers remains, only the String-ID changes. A new assignment of string ID ↔ content is now available.

Mix codes in string register

#VMC String-ID, Number (0), Offset (0)

The content of a string register (**String-ID**) is interchanged randomly.

Number indicates the number of digits (= 0 complete string), **Offset** the starting point within the register.

Register

Set register (Integer)

#VRI	Register-ID, Value, Value1 [ID+1], ...
-------------	--

The command saves an integer value (32 bits) in the register set (**Register-ID** [0 ... 499]).

Register-ID	Value	
0	10	...
1	42	#VRI 0,10,42,-8
2	-8	...

Set register (float)

#VRF	Register-ID, Value, Value1 [ID+1], ...
-------------	--

The command saves a float value (32 bit) in the register set (**Register-ID** [0 ... 499]).

Register-ID	Value	
0	10.25	...
1	42.39	#VRF 0,10.25,42.39,-8.19
2	-8.19	...

Convert object string

#VRO	Register-ID, Obj-ID, Obj-ID1[ID+1], ...
-------------	---

Object strings are stored in registers. The object string is converted into a numerical value (automatically fitting as an integer or float). This function is mainly used for EditBoxes.

Register-ID	Value	
0
1	42.5	#SED 1, "42.5" /**Default text for EditBox
2	...	#VRO 1,1
2

Convert string register to register

#VRE	Register-ID Start, String-ID Source, Seperator, Register-ID Number
-------------	--

Convert numeric string (**String-ID Source**) to register (**Register-ID Start**). **Separator** specifies the separator between the values. The number of valid values after the conversion is specified in the optional parameter **Register-ID Number**

Register-ID	Value	
0	10	...
		#VSS 0, "10,42.39,-8";
		#VRE 0,0,?, ,10

1	42.39	
2	-8	...
10	3	

Write register to RTC-RAM

#VRW ID, Register-ID, Register-ID1, ...

Buffer a **Register-ID** in the RAM of the RTC. **ID** [0 ... 7] indicates the storage space. The value is retained even after the module is switched off. A RTC needs to be connected (Attention: EA uniTFTs020-ATC and EA uniTFTs028-ATC).

Read register from RTC-RAM

#VRR ID, Register-ID, Register-ID1, ...

Read back a value from the RTC-RAM (**ID**) and transfer it to the register (**Register-ID**). A RTC needs to be connected (Attention: EA uniTFTs020-ATC and EA uniTFTs028-ATC).

Convert string register as calculation to register (integer)

#VCI Register-ID, String-ID, String-ID1[ID+1], ...

Interpret the content of a string register as a calculation string. The result is stored in the register (**Register-ID**)

Register-ID	Vaule	...
0	-8	#VSS 0, " R0+R1 ";
1	50	#VCI 2,0
2	42	...

Convert string register as calculation to register (float)

#VCF Register-ID, String-ID, String-ID1[ID+1], ...

Interpret the content of a string register as a calculation string. The result is stored in the register (**Register-ID**)

Register-ID	Value	...
0	-8.21	#VSS 0, " R0+R1 ";
1	50.89	#VCF 2,0
2	42.68	...

Send register

#VRG Register-ID, ...

Place the contents of the register in the [send buffer](#). The feedback is structured as follows:

ESC	V	R	G	Register-ID	Type	Value	
\$1B	\$56	\$52	\$47	16-Bit value	16-Bit value	32-Bit value	...

```
1Bh 56h 52h 47h 00h 00h 46h 00h 29h 5Ch 03h C1h ...
#VRF 0, ...
#VRG 0
...
```

Sort register (from V1.1)

#VQR	Register-ID Start, Register-ID End
------	------------------------------------

The area of the registers (**Register-ID Start** to **Register-ID End**) are sorted.

Register-ID	Value before	Value after	
0	2	-5	...
1	8	2	#VRI 0, 2, 8, 4, -5
2	4	4	#VQR 0, 3
3	-5	8	...

Mix register (from V1.3)

#VMR	Register-ID Start, Register-ID End
------	------------------------------------

The content of the registers remains, only the Register-ID changes. A new assignment of Register-ID ↔ content is now available.

Array (from V1.4)

Define array (Integer)

#VAI	Array-ID, Number, Type(0)
------	---------------------------

The command defines an integer array (**Array-ID** [0 ... 499]) with the given **Number** of entries. The maximum length of the array respectively if the array has the desired length, can be checked with the calculation [arE\(\)](#). The **Type** specifies the behaviour when writing at the end of the array.

Type	
0	Stop at end
1	Wrap around (ring buffer)

Define array (Float)

#VAF	Array-ID, Number, Type(0)
------	---------------------------

The command defines a float array (**Array-ID** [0 ... 499]) with the given **Number** of entries. The maximum length of the array respectively if the array has the desired length, can be checked with the calculation [arE\(\)](#). The **Type** specifies the behaviour when writing at the end of the array.

Type	
0	Stop at end
1	Wrap around (ring buffer)

Delete array (free memory)

#VAD	Array-ID
------	----------

The command deletes an array (**Array-ID** [0 ... 499]) and releases the memory.

Fill array

#VAS	Array-ID, Value(0, all Elements), element index
------	---

The command fills all elements of the array (**Array-ID** [0 ... 499]) with the given **Value**. If **Value** paramter is not sent, then the whole array is filled with 0. On the other hand the **element index** can specify which elements get the new **value**.

Assign values to array elements (with index)

#VAV	Array-ID, Index, Value, Value[Index+1], ...
------	---

The command assigns new **Values** to array elements, starting with the array **Index**.

Assign values to array elements (with current write pointer)

#VAW	Array-ID, Value 1, Value 2, ...
------	---------------------------------

The command assigns new **Values** to array elements, starting with the current write pointer.

Set write and*or reading pointer

#VAP	Array-ID, WritePointer, ReadPointer(-1)
------	---

The command sets the **Write** and / or **Read pointer** of the array (**Array-ID** [0 ... 499]). If the pointer should remain unchanged, the respective parameter must be set to -1.

Sort array

#VQA	Array-ID, StartIndex, EndIndex(last Index)
------	--

The values of the array (**Array-ID** [0 ... 499]) are sorted in the specified range (**StartIndex** to **EndIndex**).

Shuffle array

#VMA	Array-ID, StartIndex, EndIndex(last Index)
------	--

The command shuffles the values of the array in the specified range (**StartIndex** to **EndIndex**). The values remain unchanged. Only the order (indexes) is adjusted. A new assignment of the array indexes \leftrightarrow values is now available

I/O Port #H

The module has 8 I/O port lines, which can be expanded to up to 136. If the port input pins are changed, macros can be started, see [#MHP](#), and [#MHB](#).

Port-Access (8 I/Os)

Define port (input/output) (Hardware Port Control)	#HPC	Port, I/O, I/O [Port+1], ...
Set port output (Hardware Port Write)	#HPW	Port, State, State [Port+1], ..
Read port inputs (Hardware Port Read)	#HPR	Port (0), Number(1)
Send port information (Hardware Port Information)	#HPI	

Pin-Access

Define port-pin (input/output) (Hardware Bit Control)	#HBC	Portpin, I/O, I/O [Port+1], ...
Set port-pin output (Hardware Bit Write)	#HBW	Portpin, State, State [Port+1], ..
Read port-pin input (Hardware Bit Read)	#HBR	Portpin(0), Number(8)

Port-Access (8 I/Os)

Define port (input/Ausgang)

#HPC	Port, I/O, I/O [Port+1], ...
-------------	------------------------------

The command defines the direction (**I/O**) of the individual port pins bit by bit for an entire **Port** [0 ... 17]:

I/O	
0	Output
1	Input

	Port 0							
	0	1	2	3	4	5	6	7
Input								
Output								

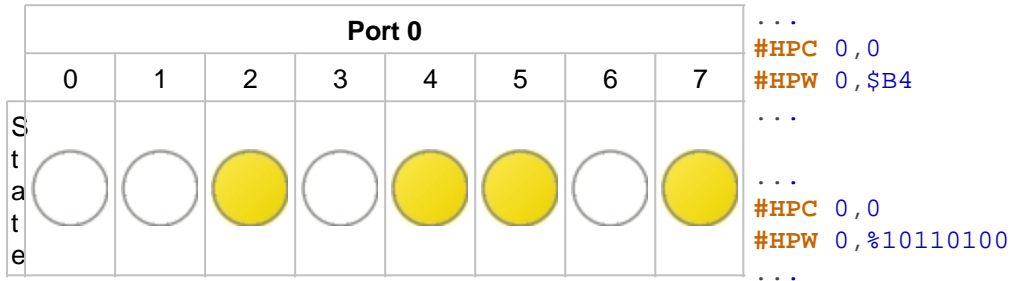
...
#HPC 0, \$E9
 ...
 ...
#HPC 0, %11101001
 ...

Set port output

#HPW	Port, State, State [Port+1], .
-------------	--------------------------------

The command sets the **State** of the outputs bit by bit for an entire **Port**.

State	
0	Low
1	High



Read port inputs

```
#HPR Port (0), Number(1)
```

The command puts the state of one or more (**Number**) of ports (starting with **Port**) in the [send buffer](#). The feedback is structured as follows:

ESC	H	P	R	Port	Number	State 1	State 2	...
\$1B	\$48	\$50	\$52	8-Bit value	8-Bit value	8-Bit value	8-Bit value	...

```
1Bh 48h 50h 52h 00h 02h AAh FFh
...
#HPR 0,2
...
```

See also [port\(a\)](#)

Send port information

```
#HPI
```

Indicates which of the 16 possible port modules are connected (= 1) and places this information in the [send buffer](#). The feedback is structured as follows:

ESC	H	P	I	Returnvalue (1 bit per port)
\$1B	\$48	\$50	\$49	16-Bit value

1Bh 48h 50h 49h 01h 00h

...
#HPI
...

The internal port is not monitored. The command returns only the port expanders that are connected externally.

Pin-Access

Define port-pin (input/output)

#HBC	Portpin, I/O, I/O [Port+1], ...
------	---------------------------------

The command defines the direction (**I/O**) for the **Portpin**:

I/O	
0	Output
1	Input

Set port-pin output

#HBW	Portpin, State, State [Port+1], ..
------	------------------------------------

The command sets the **State** of the output for the **Portpin**.

State	
0	Low
1	High
2	Invert

Read port-pin input

#HBR	Portpin(0), Number(8)
------	-----------------------

The command puts the state of one or more (**Number**) of port pins (starting with port pin) in the [send buffer](#). The feedback is structured as follows:

ESC	H	B	R	Portpi n	Numb er	State 1	State 2	...
\$1B	\$48	\$42	\$52	8-Bit value	8-Bit value	8-Bit value	8-Bit value	

```
1Bh 48h 42h 52h 00h 04h 01h 00h 01h 00h
...
#HBR 0,4
...
```

See also [bit\(a\)](#)

Analogue Input #H

Command group to parametrize and read out the analog input of the module. The module has four 12-bit analog inputs. If the analog input changes, a macro can be started, see [#MHA](#).

Read analogue input (Hardware Analog Read)	#HAR	Channel(0), Number(4)
Set limits/threshold (Hardware Analog Limit)	#HAL	Channel, Limit1, Limit2, Limit1 [Channel+1], Limit2 [Channel+1], ...
Set hysteresis (Hardware Analog Hysteresis)	#HAH	Channel, Hysteresis , Hysteresis [Channel+1], ...

Read analogue input

#HAR	Channel(0), Number(4)
-------------	-----------------------

The command reads out one or more (**Number**) of analog channels (starting with **Channel** [0 ... 4]) and places the value in the [send buffer](#). The feedback is structured as follows:

ESC	H	A	R	Chan nel	Numb er	Value 1	Value 2	...
\$1B	\$48	\$41	\$52	8-Bit value	8-Bit value	16-Bit value	16-Bit value	...

```
1Bh 48h 41h 52h 00h 02h 5Ch 0Dh B8h 06h
...
#HAR 0,2
...
```

See also [analog\(a\)](#)

Set limits/threshold

#HAL	Channel, Limit1, Limit2, Limit1 [Channel+1], Limit2 [Channel+1], ...
-------------	--

For each analog input (**Channel**), 2 threshold values can be set to call macros ([#MHA](#)). The **Limits** are given in ADC counts.

Set hysteresis

#HAH	Channel, Hysteresis , Hysteresis [Channel+1], ...
-------------	---

Set the **Hysteresis** for the respective **Channel** in ADC counts. The default value for each channel is 4. Only after the hysteresis has been exceeded is the respective defined macro is called.

PWM Output #H

Command group for the PWM output

Set the PWM frequency and duty cycle (Hardware PWM Output)	#HFO	Frequency [32-Bit], On Value (no change), Total Value (no change)
Change PWM duty cycle (Hardware PWM DutyCycle)	#HFD	On Value, Total Value (no change)

Set the PWM frequency and duty cycle

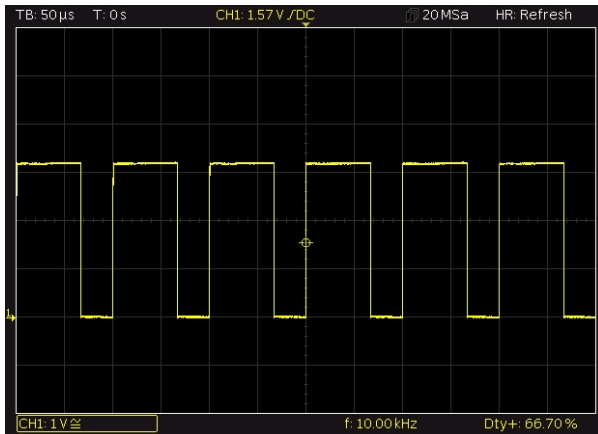
#HFO	Frequency [32-Bit], On Value (no change), Total Value (no change)
-------------	---

Setting the PWM **Frequency** (32-bit value) (2Hz ... 1MHz).

Frequency	
0	Permanent low
1	Permanent high

The two optional parameters **On Value** and **Total Value** set the duty cycle:

$$\text{DutyCycle} = \frac{\text{On Value}}{\text{Total Value}}$$



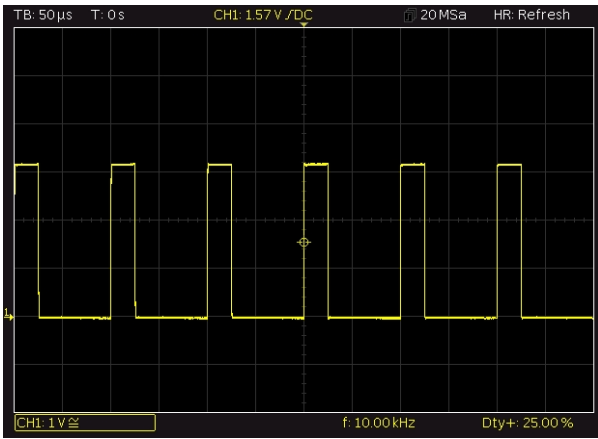
...
#HFO 10000, 2, 3
...

Change PWM duty cycle

#HFD	On Value, Total Value (no change)
-------------	-----------------------------------

The command sets the duty cycle with the two parameters **On Value** and **Total Value**. The frequency is steady:

$$\text{DutyCycle} = \frac{\text{On Value}}{\text{Total Value}}$$



...
#HFD 1,4
...

Serial Master-Interface #H

Command group to use the 3 serial interfaces of the module and use them as master. For example to connect additional peripherals like temperature sensor

Set RS232 baud rate (Hardware Rs232 Parameter)	#HRP	Baudrate [32-Bit]
Set SPI parameters (Hardware Spi Parameter)	#HSP	Frequency, Mode, DataOrder
Set SPI chip select (Hardware Spi Chipselect)	#HSC	ChipSelect
Set I²C parameters (Hardware I2c Parameter)	#HIP	Address, Frequency
Send 8-Bit (ASCII) string (Hardware RS232/SPI/I2c Ascii)	#HRA	"String";
	#HSA	
	#HIA	
Send 16-Bit (Unicode) string (Hardware RS232/SPI/I2c Unicode)	#HRU	"String";
	#HSU	
	#HIU	
Send 32-Bit signed values (Hardware RS232/SPI/I2c Integer)	#HRI	Value, Value1...
	#HSI	
	#HII	
send 32-Bit float values (Hardware RS232/SPI/I2c float)	#HRT	Value, Value1...
	#HST	
	#HIT	
Send binary data (Hardware RS232/SPI/I2c Send binary)	#HRS	Number, Data...
	#HSS	
	#HIS	
Send binary data from register (Hardware RS232/SPI/I2c send values)	#HRX	Type, Register-ID, Number(1)
	#HSX	
	#HIX	
Send binary data from array (Hardware RS232/SPI/I2c send array)	#HRY	Type, Array-ID, StartIndex(0), Number(all elements)
	#HSY	
	#HIY	
Senden file (Hardware RS232/SPI/I2c send File)	#HRF	<Filename>
	#HSF	
	#HIF	
Receive data and place into send buffer (Hardware RS232/SPI/I2c Receive to	#HRR	Number [32-Bit] (max 1024)

buffer)	#HSR	
	#HIR	
Receive 8-bit data and write it to a string register (Hardware RS232/SPI/I2c Bytes to string)	#HRB	String-ID, Number (max 250)
	#HSB	
	#HIB	
	#HRW	
Receive 16-bit data and write it to a string register (Hardware RS232/SPI/I2c Words to string)	#HSW	String-ID, Number (max 250)
	#HIW	
	#HRV	
Receive binary data and write it to a register (Hardware RS232/SPI/I2c Values to register)	#HSV	Type, Register-ID, Number(1)
	#HIV	
	#HRZ	
Receive binary data and write it to an array (Hardware RS232/SPI/I2c Values to array)	#HSZ	Typ, Array-ID, StartIndex(0), Anzahl(alles Elemente)
	#HIZ	

The respective interface can't be used as slave-interface after one of the above commands. The interface gets master functionality for controlling external peripherals.

Set RS232 baud rate

#HRP	Baudrate
------	----------

The command sets the **Baudrate** (32-bit value):

Baudrate	Error
9600	+0.04
19200	-0.08
38400	+0.16
57600	-0.08
115200	+0.64
230400	-0.80
460800	+2.08
921600	-3.68

Set SPI parameters

#HSP	Frequency, Mode, DataOrder
------	----------------------------

The command sets the **Frequency** (15600 ... 1000000 Hz), the SPI **Mode** (0..3) and the **DataOrder** of the master

SPI interface.

DataOrder	
0	MSB first
1	LSB first

Set SPI chip select

#HSC	ChipSelect
------	------------

The command defines the **ChipSelect** setting:

ChipSelect	
0	Low
1	High
2	Automatic (low active)

Set I²C parameters

#HIP	Address, Frequency
------	--------------------

The command sets the **Address** of the bus subscriber to be controlled and the **Frequency** (3900 ... 1000000 Hz).

Send 8-Bit (ASCII) string

#HR A	(RS2 32)	"String";
#HS A	(SPI)	
#HI A	(I ² C)	

The command sends a **String** or individual codes as ASCII value(s) (8 bits per character).

Send 16-Bit (Unicode) string

#HR U	(RS2 32)	"String";
#HS U	(SPI)	
#HI U	(I ² C)	

The command sends a **String** or individual codes as Unicode value(s) (16 bits per character).

Send 32-Bit signed values

#HR (RS2 I 32)	Value, Value1...
#HS (SPI) I	
#HII (I ² C)	

The command sends one or more 32-bit signed integer **Value(s)** (little endian).

send 32-Bit float values

#HR (RS2 T 32)	Value, Value1...
#HS (SPI) T	
#HI (I ² C) T	

The command sends one or more 32-bit float **Value(s)** (little endian).

Send binary data

#HR (RS2 S 32)	Number, Data...
#HS (SPI) S	
#HI (I ² C) S	

The command sends a **Number** of **Data** directly via the master interface. The data are taken over and sent directly, no interpretation, such as calculation interpretation, takes place.

Send binary data from register

#HR (RS2 X 32)	Type, Register-ID, Number(1)
#HS (SPI) X	
#HI (I ² C) X	

The command sends a Number of register entries (**Register-ID**) in binary form via the master interface.

Type

7	Signed Byte	1 Byte	little endian
8	Unsigned Byte	1 Byte	
15	Signed Integer	2 Byte	
16	Unsigned Integer	2 Byte	
23	Signed Integer	3 Byte	
24	Unsigned Integer	3 Byte	
31	Signed Integer	4 Byte	
32	Unsigned Integer	4 Byte	
33	Float	4 Byte	
115	Signed Integer	2 Byte	big endian
116	Unsigned Integer	2 Byte	
123	Signed Integer	3 Byte	
124	Unsigned Integer	3 Byte	
131	Signed Integer	4 Byte	
132	Unsigned Integer	4 Byte	
133	Float	4 Byte	

Send binary data from array

#HR Y (RS2 32)	Type, Array-ID, StartIndex(0), Number(all elements)
#HS Y (SPI)	
#HI Y (I ² C)	

The command sends a **Number** of array elements (**Array-ID**), starting with the start index, in binary form via the master interface.

Type			
7	Signed Byte	1 Byte	little endian
8	Unsigned Byte	1 Byte	
15	Signed Integer	2 Byte	
16	Unsigned Integer	2 Byte	
23	Signed Integer	3 Byte	
24	Unsigned Integer	3 Byte	
31	Signed Integer	4 Byte	
32	Unsigned Integer	4 Byte	
33	Float	4 Byte	
115	Signed Integer	2 Byte	big endian
116	Unsigned Integer	2 Byte	
123	Signed Integer	3 Byte	
124	Unsigned Integer	3 Byte	
131	Signed Integer	4 Byte	
132	Unsigned Integer	4 Byte	
133	Float	4 Byte	

Send file

#HR (RS2 F 32)	<Filename>
#HS (SPI) F	
#HI (I ² C) F	

The command sends a file (<Filename>) via the master interface.

Receive data and place it into send buffer

#HR (RS2 R 32)	Number [32-Bit] (max 1024)
#HS (SPI) R	
#HI (I ² C) R	

The command reads a **Number** (32-bit value) of data from the master receive buffer and places it in the [send buffer](#). The feedback is structured as follows:

ESC	H	R/S/I	R	Length	Data 1	Data 2	...	Data n	...
\$1B	\$48	\$52/\$53/\$49	\$52	32-Bit value	8-Bit value	8-Bit value	8-Bit value	8-Bit value	...

See also [mstRA\(\)](#)

Receive 8-Bit data and write it to a string register

#HR (RS2 B 32)	String-ID, Number (max 250)
#HS (SPI) B	
#HI (I ² C) B	

The command reads a **Number** of data from the master receive buffer and writes them to the specified string register (**String-ID**).

See also [mstRA\(\)](#)

Receive 16-Bit data and write it to a string register

#HR (RS2 W 32)	String-ID, Number (max 250)
#HS (SPI) W	
#HI (I ² C) W	

The command reads a **Number** of data from the master receive buffer and writes them to the specified string register (**String-ID**).

See also [mstRA\(\)](#)

Receive binary data and write it to a string register

#HR (RS2 V 32)	Type, Register-ID, Number(1)
#HS (SPI) V	
#HI (I ² C) V	

The command reads a **Number** of data from the master receive buffer and writes them to the specified register (**Register-ID**).

Type			
7	Signed Byte	1 Byte	little endian
8	Unsigned Byte	1 Byte	
15	Signed Integer	2 Byte	
16	Unsigned Integer	2 Byte	
23	Signed Integer	3 Byte	
24	Unsigned Integer	3 Byte	
31	Signed Integer	4 Byte	
32	Unsigned Integer	4 Byte	
33	Float	4 Byte	big endian
115	Signed Integer	2 Byte	
116	Unsigned Integer	2 Byte	
123	Signed Integer	3 Byte	
124	Unsigned Integer	3 Byte	
131	Signed Integer	4 Byte	

132	Unsigned Integer	4 Byte	
133	Float	4 Byte	

See also [mstRA\(\)](#)

Receive binary data and write it to an array

#HR (RS2 Z 32)	Typ, Array-ID, StartIndex(0), Anzahl(alle Elemente)
#HS (SPI) Z	
#HIZ (I ² C)	

The command reads a **Number** of data from the master receive buffer and writes them, starting with the start index, into the specified array (**Array-ID**). An array must be defined before receiving (see [#VAI](#), [#VAE](#)).

Type			
7	Signed Byte	1 Byte	little endian
8	Unsigned Byte	1 Byte	
15	Signed Integer	2 Byte	
16	Unsigned Integer	2 Byte	
23	Signed Integer	3 Byte	
24	Unsigned Integer	3 Byte	
31	Signed Integer	4 Byte	
32	Unsigned Integer	4 Byte	
33	Float	4 Byte	
115	Signed Integer	2 Byte	big endian
116	Unsigned Integer	2 Byte	
123	Signed Integer	3 Byte	
124	Unsigned	3 Byte	

	Integer		
131	Signed Integer	4 Byte	
132	Unsigned Integer	4 Byte	
133	Float	4 Byte	

Sound #H

Command group to play jingles.

Play sound (Hardware Tone Notes)	#HTN	"Sound string"
Stop sound (Hardware Tone Stop)	#HTS	

Play sound / jingle

#HTN	"Sound string"
-------------	----------------

The command plays the specified "**Sound string**". The possibilities of notes can be found on the [bottom](#).

Stop sound

#HTS	
-------------	--

The command stops the currently playing sound file.

Notes

The structure of the note string consists of a divider for subsequent notes/pauses, a possible semitone increase for the next note and the notes themselves:

1..9	Teiler für nachfolgende Noten/Pausen
#	Halbtonerhöhung für nächste Note
C,D,E,F,G, A,B,H	Noten (5. Oktave)
c,d,e,f,g,a, b,h	Noten (6. Oktave)
P	Pause

Time #W

Command group to work with the built-in (EA uniTFTs035-ATC / EA uniTFTs043-ATC) and the externally connected RTC.

Set time (Watch Time Date set)	#WTD	Hour, Minute (act. time), Second (act. time), Day (act. time), Month (act. time), Year (act. time), Adjust (0)
Define object group as watch (Watch Group Clock)	#WGC	Group-ID, HourHand-ID, MinuteHand-ID, SecondHand-ID(none)
Define print format for RTC (Watch Define Format)	#WDF	"Dateformat"
Define month names (Watch Define Month strings)	#WDM	"JAN";"FEB";"MAR";"APR";"MAI";"JUN";"JUL";"AUG";"SEP";"OCT";"NOV";"DEC"
Define day of week names (Watch Define Day strings)	#WDW	"SUN";"MON";"TUE";"WED";"THU";"FRI";"SAT"
Send time (ASCII) (Watch Send Ascii)	#WSA	"Dateformat"; date (act. time)
Send time (Unicode) (Watch Send Unicode)	#WSU	"Dateformat"; date (act. time)
Send time (Binär) (Watch Send Binary)	#WSB	"Dateformat"; date (act. time)
Define base year for time calculation (Watch Define base Year)	#WDY	Year

Set time

#WTD	Hour, Minute (act. time), Second (act. time), Day (act. time), Month (act. time), Year (act. time), Adjust (0)
------	--

The command sets the current time. If the optional parameter **Adjust** is set to 1, the internal crystal will be calibrated the next time (**Adjust** must also be 1).

Define object group as watch

#WGC	Group-ID, HourHand-ID, MinuteHand-ID, SecondHand-ID(none)
------	---

The command converts an existing group into a clock. **HourHand-ID** specifies the Obj ID for the hour hand, **MinuteHand-ID** the Obj ID for the minute hand, **SecondHand-ID** the Obj ID for the second hand.



```
...
#PPP
1,<P:picture/Clock.epg>,120,120,5,200,200,0
#PPP
2,<P:picture/Needle.epg>,120,156,5,6,100,0
#PPP 3,<P:picture/Needle.epg>,120,146,5,6,80,0
#WGC 4,3,2
...
```

Define print format for RTC

```
#WDF "Dateformat"
```

The command changes the [date format](#).

Define month names

```
#WDM "JAN";"FEB";"MAR";"APR";"MAI";"JUN";"JUL";"AUG";"SEP";"OCT";"NOV";"DEC"
```

The command is used to set 12 individual strings for the month names.

Define day of week names

```
#WDW "SUN";"MON";"TUE";"WED";"THU";"FRI";"SAT"
```

The command is used to set 7 individual strings for the weekday names (starting with Sunday).

Send time (ASCII)

```
#WSA "Dateformat"; date (act. time)
```

The command places the date and time as an ASCII string in the [send buffer](#). The feedback is structured as follows:

ESC	W	S	A	ASCII-String	Endin g	...
\$1B	\$57	\$53	\$41		\$00	

See also [year\(\)](#), [month\(\)](#), [day\(\)](#), [weekday\(\)](#), [hour\(\)](#), [minute\(\)](#), [second\(\)](#)

Send time (Unicode)

```
#WSU "Dateformat"; date (act. time)
```

The command places the date and time as a Unicode string in the [send buffer](#). The feedback is structured as follows:

ESC	W	S	U	Unicode-String	Endin g	...
-----	---	---	---	----------------	------------	-----

\$1B	\$57	\$53	\$55		\$00	
------	------	------	------	--	------	--

See also [year\(\)](#), [month\(\)](#), [day\(\)](#), [weekday\(\)](#), [hour\(\)](#), [minute\(\)](#), [second\(\)](#)

Send time (Binär)

#WSB	"Dateformat"; date (act. time)
-------------	--------------------------------

The command puts the date and time in the [send buffer](#) as a signed 32-bit value. The feedback is structured as follows:

ESC	W	S	B	Hour	Minute	Second	Day	Month	Year	Weekday	
\$1B	\$57	\$53	\$42	16-Bit value	16-Bit value	16-Bit value	16-Bit value	16-Bit value	16-Bit value	16-Bit value	...

See also [year\(\)](#), [month\(\)](#), [day\(\)](#), [weekday\(\)](#), [hour\(\)](#), [minute\(\)](#), [second\(\)](#)

Define base year for time calculations (from V1.3)

#WDY	Year
-------------	------

The command changes the base year for the time calculation. Possible values are 1970,1980,1990,2000,2010,2020,2030. The range of values is -68 to +67 years. The preset second count starts on January 1st, 2000 at 0: 0: 0. This is the possible range from 1932 to the end of 2067.

Date formats

Format	
%[h]	Hour
%[m]	Minute
%[s]	Second
%[D]	Day
%[M]	Month
%[Y]	Year
%[W]	Day of week (String)
%[N]	Month (String)

Optional []	
0	Two digits with leading 0 (Default)
1	Minimum 1 digits without leading

	character
2	Two digits with leading spaces
4	Four digits (default for year)

Week string and month string:

Optional []	
0-9	x chars are shown for the week string or month string.

Examples	
"%h:%m:%s";	09:25:04
"%D.%M.%Y";	20.12.2019
"%D %N %Y";	20 December 2019
"%W, %D.%M.%Y";	Friday, 20.12.2019

Files on the internal memory #F

Commands to handle file access.

Folder / Directory

Create folder (File Directory Create)	#FDC	<Path>
Delete folder (File Directory Delete)	#FDD	<Path>, Delete
Set working directory (File Directory Set)	#FDS	<Path>
Send working directory (File Directory Get)	#FDG	
Send all folders and files (directory) (File Directory Read binary)	#FDR	<Pfad> (act. working directory)
Send all folders and files (directory) (ASCII) (File Directory read Ascii)	#FDA	<Pfad> (act. working directory)
Send all folders (directory) (File Directory read dironly List)	#FDL	<Pfad> (act. working directory)

Files

Open file for writing (File Write Open)	#FWO	<FileName>, Position [32-Bit] (End), Truncate(1), size(4096)
Close file (write operation) (File Write Close)	#FWC	Time, Date
Set write position in file (File Write Position)	#FWP	Position [32-Bit]
Write data to file (File Write Data binary)	#FWD	Number [32-Bit], Binary data
Write ASCII-String to file (File Write Ascii)	#FWA	"String"
Write Unicode-String to file (File Write Unicode)	#FWU	"String"
Write register values to file (File Write Register)	#FWR	Register-ID, ...
Write string register to file (File Write Stringregister)	#FWS	String-ID, ...
Write array to file (File Write Array)	#FWY	Array-ID, ...
Open file for reading (File Read Open)	#FRO	<FileName>, Position [32-Bit] (Start)
Close file (read operation) (File Read Close)	#FRC	
Set read position in file (File Read Position)	#FRP	Position [32-Bit]
Read and send data (File Read Data binary)	#FRD	Number [32-Bit] (whole file)

Read ASCII string and write into string register (File Read Ascii)	#FRA	String-ID, ...
Read Unicode string and write into string register (File Read Unicode)	#FRU	String-ID, ...
Load data into register (File Read Register)	#FRR	Register-ID, ...
Load data into string register (File Read Stringregister)	#FRS	String-ID, ...
Read data (8-bit) and write into string register (File Read Bytes to stringregister)	#FRB	String-ID, Number, Number [ID+1],...
Read data (16-bit) and write into string register (File Read Words to stringregister)	#FRW	String-ID, Number, Number [ID+1],...
Read data into array (File Read Array)	#FRY	Array-ID, ...
Delete file (File File Delete)	#FFD	<Filename>

General commands

Send file/folder information (File File Info)	#FFI	<Path> (act. working directory)
Rename file/folder (File File Rename)	#FFR	<Path>, <New Filename>, Replace (0)
Copy file/folder (File File Copy)	#FFC	<Path>, <New Path>, Replace (0)
Move file/folder (File File Move)	#FFM	<Path>, <New Path>, Replace (0)
Change time stamp of file/folder (File change Timestamp)	#FFT	<Path>, Time, Date
Change attributes of file / folder (File change Attribut)	#FFA	<Path>, Attribute
Load file names into string register (File Names Files to stringregister)	#FNF	<Path> (act. working directory), ID (1)
Load subdirectories into string register (File Names Directory to stringregister)	#FND	<Path> (act. working directory), ID (1)

Folder / Directory

Create folder

#FDC	<Path>
-------------	--------

The command creates a new folder. The **<Path>** parameter specifies the name and location.

...

#FDC <Project/NewPath>

...

Delete folder

```
#FDD <Path>, Delete
```

The command deletes a folder. The **<Path>** parameter specifies the name and location.

Delete	
0	Delete folder and content
1	Delete content only

```
...
#FDD <Project/NewPath>, 0
...
```

Set working directory

```
#FDS <Path>
```

The command sets the current working directory. With path </> you reach the root directory.

```
...
#FDS <Project>
...
```

Send working directory

```
#FDG
```

The command places the current working directory in the [send buffer](#). The feedback is structured as follows:

ESC	F	D	G	Path	
\$1B	\$46	\$44	\$47	'String' completed with \$00	...

Send all folders and files (directory)

```
#FDR <Pfad> (act. working directory)
```

The command places all folders and files of the current working directory in the [send buffer](#). The feedback is structured as follows:

ESC	F	D	R	Path/ Filename	Size	Attribute	Time	Date	...	Ending
-----	---	---	---	-------------------	------	-----------	------	------	-----	--------

\$1B	\$46	\$44	\$52	'String' completed with \$00	32-Bit value	8-Bit value	16-Bit value	16-Bit value		\$00
------	------	------	------	------------------------------	--------------	-------------	--------------	--------------	--	------

Attribute	
\$01	Read only
\$02	Hidden
\$04	System
\$20	Archive

```
1Bh 46h 44h 52h 50h 72h 6Fh 6Ah 65h 63h 74h 00h
23h 00h 00h 00h 20h 8Fh 43h 27h 50h ... 00h
...
#FDR
...
```

Send all folders and files (directory) (ASCII)

```
#FDA <Pfad> (act. working directory)
```

The command places all folders and files of the current working directory in the [send buffer](#) as ASCII strings.

ESC	F	D	A	String	Ending	...
\$1B	\$46	\$44	\$41	Size, Attribute, Time, Date, Name	CRLF	

```
←FDAD: 0 B 07.01.2020 08:28:30 Project
D: 0 B 20.12.2019 12:28:56 02_PlacePicture
35 B 07.01.2020 08:28:30 start.emc
...
#FDA
...
```

Send all folders (directory)

```
#FDL <Pfad> (act. working directory)
```

The command places all folders of the current working directory in the [send buffer](#). The feedback is structured as follows:

ESC	F	D	L	Directory-name		Ending
\$1B	\$46	\$44	\$4C	'String' completed with \$00		\$00

Files

Attention:

Flash memories have limited erase / write cycles due to their design. The memory module used in the uniTFTs can typically safely execute 100,000 cycles. In order to write data, a block of memory may have to be erased, typically 30 ms are required for erasing, but it can take up to 400 ms. This must be taken into account in the macro sequence when write file commands are executed.

Open file for writing

```
#FWO <FileName>, Position [32-Bit] (End), Truncate(1), size(4096)
```

The command opens (write only) or creates a file. The **Position** (32-bit value) indicates the write position in the file. The file**size** needs to be set. Resizing is impossible afterwards.

Truncate	
0	Overwrite data
1	Truncate old data

Close file (write operation)

```
#FWC Time, Date
```

The command closes an open file (write operation). The writing process is completed and it is ensured that all data has been written. If the **time** and **date** are specified or the time is set beforehand, the time stamp is set, otherwise 1.1.1980 remains and can be set later.

Set write position in file

```
#FWP Position [32-Bit]
```

The command sets the write **Position** (32-bit value) at the specified position in the file. If the value is <0, the position is calculated from the end of the file

See also [fposW\(\)](#)

Write data to file

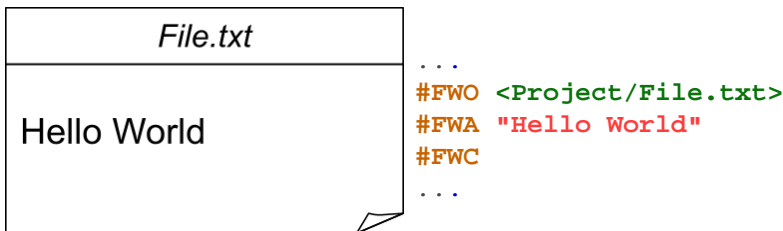
```
#FWD Number [32-Bit], Binary data
```

The command writes a **Number** (32-bit value) of **Binary data** to the opened file. This command is unsuitable for macro programming. It is used to transfer binary data via the interface.

Write ASCII-String to file

#FWA "String"

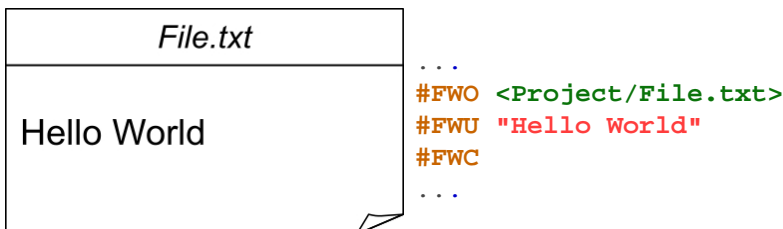
The command writes an ASCII string (8 bits per character) into the open file.



Write Unicode-String to file

#FWU "String"

The command writes a Unicode string (16 bits per character) into the open file.



Write register values to file

#FWR Register-ID, ...

The command writes the register value (**Register-ID**) to the open file. The value can be read out with the [#ERR](#) command. 5 bytes are required for each register.

Write string register values to file

#FWS String-ID, ...

The command writes the content of the string register (**String-ID**) to the open file. The string can be read out with the command [#FRS](#). $2 \cdot (n + 1)$ bytes (n = number of letters) are required for each register.

Write array to file

#FWY Array-ID, ...

The command writes the content of the array (**Array-ID**) to the open file. The string can be read out with the command [#FRY](#). $6 + 4 \cdot n$ Bytes (n = array length) are required for each array..

Open file for reading

#FRO <FileName>, Position [32-Bit] (Start)

The command opens (read only) a file. The **Position** (32-bit value) indicates the read position in the file.

Close file (read operation)

#FRC

The command closes an open file (read operation).

Set read position in file

#FRP Position [32-Bit]

The command sets the read **Position** (32-bit value) at the specific position in the file. If the value is <0, the position is calculated from the end of the file

See also [fposR\(\)](#)

Read and send data

#FRD Number [32-Bit] (whole file)

The command reads a **Number** (32-bit value) of bytes from the open file and places the data in the [send buffer](#). The feedback is structured as follows:

ESC	F	R	D	Number	Data 1	Data 2		Data n	
\$1B	\$46	\$52	\$44	32-Bit value	8-Bit value	8-Bit value		8-Bit value	...

Read ASCII string and write into string register

#FRA String-ID, ...

The command reads an ASCII string (8 bits per character) up to the character "\n" and stores it in a string register (**String-ID**).

Read Unicode string and write into string register

#FRU String-ID, ...

The command reads a Unicode string (16 bits per character) up to the character "\n" and stores it in a string register (**String-ID**).

Load data into register

#FRR Register-ID, ...

The command reads back a register written with [#FRW](#) and saves it in the register (**Register-ID**).

Load data into string register

#FRS String-ID, ...

The command reads back a string register written with [#FRS](#) and saves it in the string register (**String-ID**).

Read data (8-Bit) and write into string register

#FRB String-ID, Number, Number [ID+1],...

The command reads a **Number** (1 ... 250) of bytes and saves it in the string register (**String-ID**).

Read data (16-Bit) and write into string register

#FRW String-ID, Number, Number [ID+1],...

The command reads a **Number** (1 ... 250) of Words and saves it in the string register (**String-ID**).

Read data into array

#FRY Array-ID, ...

The command reads back an array written with [#FWY](#) and saves it in the array (Array-ID).

Delete file

#FFD <Filename>

The command deletes a file (<Filename>)

General commands

Send file information

#FFI <Path> (act. working directory)

The command puts all information about the file (such as time stamp, size) in the [send buffer](#). If the file does not exist, an empty string is returned, the remaining parameters are no longer transferred. The feedback is structured as follows:

ESC	F	F	I	File-name	Size	Attribute	Time	Date	...
\$1B	\$46	\$46	\$49	'String' completed with \$00	32-Bit value	8-Bit value	16-Bit value	16-Bit value	...

See also [fileS\(\)](#), [fileA\(\)](#), [fileT\(\)](#)

Rename file

#FFR <Path>, <New Filename>, Replace (0)

The command changes the specified <Path> to a new name (<New Filename> is only the new name, without path).

Replace	
0	Do not rename if folder / file already exists
1	Delete existing folder / file and rename it

Copy file

#FFC <Path>, <New Path>, Replace (0)

The command copies the specified file (<Path>) to a new location (<New Path>).

Replace	
0	Do not rename if file already exists
1	Delete existing file and rename it

Move file

#FFM <Path>, <New Path>, Replace (0)

The command moves the specified file (<Path>) to a new location (<New Path>).

Replace	
0	Do not rename if file already exists
1	Delete existing file and rename it

Change time stamp of file

#FFT <Path>, Time, Date

The command changes the time stamp of the file (<Path>):

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Time	Hour [0...23]					Minute [0...59]					Second/2 [0...29]					
Date	Year (from 1.1.1980 0:0:0 Hour) [0...127]					Month [1...12]					Day [1...31]					

Example for calculation:

Time = (Hour<<11) + (Minute<<5) + (Second>>1);

Date = ((Year-1980)<<9) + (Month<<5) + Day;

See also [fatT\(datetime\)](#), [fatD\(datetime\)](#), [fattime\(Fat-Time, Fat-Date\)](#)

Change attributes of file

#FFA <Path>, Attribute

The command sets the **Attributes** of the file. The attributes can be set simultaneously with bit coding.

Attribute	
\$01	Read only

\$02	Hidden
\$04	System
\$20	Archive

See also [fileA\(\)](#)

Load file names into string register

#FNF	<Path> (act. working directory), ID (1)
------	---

The command saves all file names from the **<Path>** in the string register (String-ID = **ID ... IDn**). The number is stored in the register (Register ID = **ID**). (You can search with patterns **?/*** e.g. ***.txt** stores all text files)

Load subdirectories into string register

#FND	<Path> (act. working directory), ID (1)
------	---

The command saves all folder names that are in the **<Path>** in the string register (String-ID = **ID ... IDn**). The number is stored in the register (Register ID = **ID**). (You can search with patterns **?/***, e.g. ***New*** stores all directories in which the word "New" occurs)

System commands #X

Settings of the EA uniTFTs-Series.

Interface setting for communication to external control unit (Slave Interface)

Set slave RS232 parameters (System Configure Rs232 slave)	#XCR	Baudrate [32-Bit], RS485 (no change), Flash(0)
Set slave SPI parameters (System Configure Spi slave)	#XCS	Mode, DataOrder (no change), Flash(0)
Set slave I²C parameters (System Configure I2c slave)	#XCI	Address, Flash(0)

Module commands

Set project path (System Projectpath Set)	#XPS	<Path>
Send project path (System Projectpath Get)	#XPG	
Backlight: set brightness (System Configure backlight Brightness)	#XCB	Brightness, Time (no change), Flash(0)
Backlight: set gradation and frequency (System Configure backlight Frequency)	#XCF	Power, Frequency (no change), Flash(0)
Backlight: state auto-dimming (System Ied AutoState)	#XAS	State
Backlight: set auto-dimming (System Ied Autostate mask)	#XAL	Mask, Time1(60), Brightness1(50), Time2(60), Brightness2(10)
Send ASCII-String (System Send Ascii)	#XSA	"String"
Send Unicode-String (System Send Unicode)	#XSU	"String"
Send Hardcopy (System Hardcopy Send)	#XHS	Format(1), x(0), y(0), Anchor(7), Width(Display width), Height(Display height)
Save hardcopy in file (System Hardcopy File)	#XHF	<Name>, Format(1), x(0), y(0), Anchor(7), Width(Display width), Height(Display height)
Display hard copy as image object (System Hardcopy to Object)	#XHO	Obj-ID, x,y,Anchor, Width, Height
Display object as new image object (System Hardcopy Id to object)	#XHI	Obj-ID, Obj-ID Source
Set display orientation (System Configure Orientation)	#XCO	Orientation
Firmwareversion senden (System Info Verion)	#XIV	
Send module parameters (System Info Display)	#XID	
Send memory overview (System Info Storage)	#XII	
Send RAM memory overview (System Info RAM)	#XIR	

Set Display-Refresh-Rate (System Configure display Update)	#XCU	Option, Flash(0)
Wait command (System Wait hs)	#XXW	Time
Activate/deactivate protocol (System Configure Protocol)	#XCP	Protocol
Reboot (System Firmware reset)	#XFB	Option(0)

Interface setting for communication to external control unit (Slave Interface)

Set slave RS232 parameters

#XCR	Baudrate [32-Bit], RS485 (no change), Flash(0)
-------------	--

The command sets the **Baudrate** (32-bit value):

Baudrate	Error
9600	+0.04
19200	-0.08
38400	+0.16
57600	-0.08
115200	+0.64
230400	-0.80
460800	+2.08
921600	-3.68

The **Flash** parameter determines whether the setting should be saved:

Flash	
0	Do not save settings
1	Save settings

Set slave SPI parameters

#XCS	Mode, DataOrder (no change), Flash(0)
-------------	---------------------------------------

The command sets the SPI **Mode** (0..3) and the **DataOrder** of the slave SPI interface.

DataOrder

0	MSB first
1	LSB first

The **Flash** parameter determines whether the setting should be saved:

Flash	
0	Do not save settings
1	Save settings

Set slave I²C parameters

#XCI	Address, Flash(0)
------	-------------------

The **Address** of the slave I²C interface is set with the command. By default, the module can be addressed with the address \$ DE. The **Flash** parameter determines whether the setting should be saved:

Flash	
0	Do not save settings
1	Save settings

Module commands

Set project path

#XPS	<Path>
------	--------

The command defines the project path. The module automatically searches for file names under this path, e.g. Macros. Paths are then specified with <P: ...>.

Send project path

#XPG	
------	--

The command places the current project path in the [send buffer](#). The feedback is structured as follows:

ESC	X	P	G	Path	
\$1B	\$58	\$50	\$47	'String' completed with \$00	...

Backlight: set brightness

#XCB	Brightness, Time (no change), Flash(0)
------	--

The command specifies the **Brightness** of the backlight [0 ... 150] in %. The parameter **Time** (in 1/100 s) indicates

how quickly the brightness is reached. In the delivery state, the brightness is 100% and changes within 1 second (time = 100). If the brightness exceeds 100%, a derating of the life-time must be expected - we recommend using this setting only for a short time, e.g. in direct sunlight. The Flash parameter determines whether the setting should be saved:

Flash	
0	Do not save settings
1	Save settings

Backlight: Set gradation and frequency

#XCF	Power, Frequency (no change), Flash(0)
------	--

The brightness levels of the backlight are determined using a power function. Depending on the area of application, it makes sense to have more levels in the low range (night vision). For this, the parameter **Power** must be increased. The default value is 10. The PWM **Frequency** [5000 ... 65535] of the backlight can also be changed if there are indifferences with ambient light. Default: frequency = 5000. The Flash parameter determines whether the setting should be saved:

Flash	
0	Do not save settings
1	Save settings

Backlight: state auto-dimming

#XAS	State
------	-------

The command sets the **State** of the automatic backlight dimming. Default dimming is deactivated:

State	
0	Off: no dimming
1	On: Auto dimming active
2	Manual retrigger

Backlight: set auto-dimming

#XAL	Mask, Time1(60), Brightness1(50), Time2(120), Brightness2(10)
------	---

If automatic dimming is activated (see #XAS), the command uses the **Mask** parameter to set which events retrigger the countdown:

Mask	
\$01	Touch

\$02	USB
\$04	RS232
\$08	SPI
\$10	PC
\$20	Master RS232

The mask bits can be set simultaneously with bit decoding. **TimeX** specifies in seconds when the new brightness (**BrightnessX**) should be set. The new brightness value is specified relative to the current brightness (0..100).

Send ASCII-String

#XSA	"String"
-------------	----------

The command places a string or individual codes as ASCII values (8 bits per character) in the send buffer.

Send Unicode-String

#XSU	"String"
-------------	----------

The command places a string or individual codes as Unicode values (16 bits per character) in the send buffer.

Send Hardcopy

#XHS	Format(1), x(0), y(0), Anchor(7), Width(Display width), Height(Display height)
-------------	--

The command takes a screenshot of the position (**x, y, Anchor**) and places it in the [send buffer](#). Depending on the format in which the picture was requested, a header and the data are returned.

Format	
1	BMP 24-Bit
2	BMP 16-Bit
3	BMP 8-Bit greyscale
11	epg 32-Bit
12	epg 16-Bit
13	epg 8-Bit greyscale
21	epg 32-Bit compressed
22	epg 16-Bit compressed
23	epg 8-Bit greyscale compressed

The feedback is structured as follows:

ESC	X	H	S	Header	Data	...
\$1B	\$58	\$48	\$53

Save hardcopy in file

#XHF	<Name>, Format(1), x(0), y(0), Anchor(7), Width(Display width), Height(Display height)
-------------	--

The command takes a screenshot of the position (**x, y, Anchor**) and writes it to a file (**<Name>**).

Format	
1	BMP 24-Bit
2	BMP 16-Bit
3	BMP 8-Bit greyscale
11	epg 32-Bit
12	epg 16-Bit
13	epg 8-Bit greyscale
21	epg 32-Bit compressed
22	epg 16-Bit

	compressed
23	epg 8-Bit greyscale compressed

Display hardcopy as image object

#XHO	Obj-ID, x,y,Anchor, Width, Height
-------------	-----------------------------------

The command creates a screenshot of the position (**x, y, Anchor**) with the size (**Width, Height**) and displays it as a new image object with the **Obj-ID**.

Display object as new image object

#XHI	Obj-ID, Obj-ID Source
-------------	-----------------------

The command creates a new image object with the **Obj-ID** from the source object (**Obj-ID Source**) and displays it.

Set display orientation

#XCO	Orientation
-------------	-------------

The command defines the orientation (0, 90, 180, 270) of the display. The default is 0° Landscape.

Send firmware version

#XIV	
-------------	--

The command places the firmware version and the detected touch panel in the [send buffer](#). The feedback is structured as follows:

ESC	X	I	V	Version string	
\$1B	\$58	\$49	\$56	'String' completed with \$00	...

See also [version\(\)](#)

Send module parameters

#XID	
-------------	--

The command puts module parameters (including resolution and interface settings) in the [send buffer](#). The feedback is structured as follows:

ESC	X	I	D	Width	Height	Color depth	Touch	VideoWidth	VideoHeight	
\$1B	\$58	\$49	\$44	16-Bit value	16-Bit value	8-Bit	8-Bit	16-Bit value	16-Bit value	...

							value	value			
--	--	--	--	--	--	--	-------	-------	--	--	--

Touch	
\$00	No touch
\$03	Resistive touch
\$07	capacitive touch
\$0F	Simulator

See also [scrW\(\)](#), [scrH\(\)](#), [touchT\(\)](#), [vidW\(\)](#), [vidH\(\)](#)

Memory overview

#XII	
-------------	--

The command places the size and free space of the internal FLASH in the [send buffer](#). The feedback is structured as follows:

ESC	X	I	I	Total	Free	...
\$1B	\$58	\$49	\$53	32-Bit value	32-Bit value	...

See also [memST\(\)](#), [memSF\(\)](#)

Send RAM memory overview

#XIR	
-------------	--

The command places the size and free space of the object RAM in the [send buffer](#). The feedback is structured as follows:

ESC	X	I	R	Total	Free	...
\$1B	\$58	\$49	\$52	32-Bit value	32-Bit value	...

See also [memRT\(\)](#), [memRF\(\)](#)

Set Display-Refresh-Rate

#XCU	Option, Flash(0)
-------------	------------------

The command sets the display refresh rate. The parameter **Option** is set to 3 by default.

Option	
0	No display update
1	One-time display update

2..10 00	Cyclical display update (time in 1 / 100s)
---------------------	--

The **Flash** parameter determines whether the setting should be saved:

Flash	
0	Do not save settings
1	Save settings

Wait command

#XXW	Time
-------------	------

The command interrupts the execution of commands for the set **Time** (in 1/100s). We recommend this command only for debugging purposes during development.

Activate/deactivate protocol

#XCP	Protocol
-------------	----------

The command activates or deactivates the small / short **Protocol**.

Protocol	
0	Deactivate
1	Activate

Reboot

#XFB	Option(0)
-------------	-----------

The module can be restarted with the command:

Option	
0	Normal reset
1	Testmode
2	Disable PowerOnMakro
3	Disable Default
4	Boot menu
5	Reserved

Answer / Feedback

The module places information in its send buffer after requests or touch events. Below the individual responses from the send buffer are explained.

The responses are binary encoded unless otherwise stated:

<ESC> = 0x1B, the size (number of bits) of each parameter are given in the explanation of each response.

The module works with little-endian (Intel-format), that means the least significant byte is transferred first.

EditBox

EditBox content	<ESC> SEU	Obj-ID, Content
-----------------	--------------	-----------------

Touch

State of button/switch	<ESC> TQS	Obj-ID, State
Radiogroup active switch	<ESC> TQR	Obj-ID, Group-ID
Keyboard key	<ESC> TQK	Obj-ID, Code
Bargraph- /Instrument value	<ESC> TQI	Obj-ID, Value
Menu entry	<ESC> TQM	Obj-ID, ItemNumber
ComboBox entry	<ESC> TQC	Obj-ID, ItemNumber
SpinBox entry	<ESC> TQB	Obj-ID, ItemNumber

Variables/Registers

Number of string files loaded	<ESC> VFC	Number
Content from string register (ASCII)	<ESC> VSA	String-ID, Length, Char1, ..., Char n
Content from string register (Unicode)	<ESC> VSU	String-ID, Length, Char1, ..., Char n
Output register value	<ESC> VRG	Register-ID, Type, Value

I/O Port

Number of port blocks	<ESC> HPI	Available
Read port	<ESC> HPR	Port, Number, State 1, State 2, ...
Read port-pin	<ESC> HBR	Portpin, Number, State 1, State 2, ...

Analogue Input

Value of analogue input	<ESC>	Channel, Number, Value 1, Value 2, ...
-------------------------	-------	--

	HAR	
--	-----	--

Master interfaces

RS232 data	<ESC> HRR	Length, Data 1, Data 2, ..., Data n
SPI data	<ESC> HSR	Length, Data 1, Data 2, ..., Data n
PC data	<ESC> HIR	Length, Data 1, Data 2, ..., Data n

RTC

Time ASCII output	<ESC> WSA	ASCII-String
Time Unicode output	<ESC> WSU	Unicode-String
Time binary output	<ESC> WSB	Hour, Minute, Second, Day, Month, Year, Weekday

File access

Current working directory	<ESC> FDG	Path
All folders and files from directory (binary output)	<ESC> FDR	Directory-/File-name, Size, Attribute, Time, Date, ...
All folders and files from directory (ASCII output)	<ESC> FDA	String
All folders from directory (ASCII output)	<ESC> FDL	Name 1, Name 2, ..., Name n
File information	<ESC> FFI	File-name, Size, Attribute, Time, Date
Data from file	<ESC> FRD	Number, Data 1, Data 2, ..., Data n

System commands

Current project path	<ESC> XPG	Path
Version information	<ESC> XIV	Version string
Display information	<ESC> XID	Width, Height, Color depth, Touch, VideoWidth, VideoHeight
RAM memory information	<ESC> XIR	Total, Free
Memory information	<ESC> XII	Total, Free
Hardcopy	<ESC> XHS	Header, Data

EditBox content

ESC	S	E	U	Obj-ID	Content	
\$1B	\$53	\$45	\$55	16-Bit value	'String' completed with \$00	...

The content of the EditBox (16 bits per character) is transferred. The string ends with a \$ 00. The feedback is triggered by keyboard code 13 (\$0D).

State of button/switch

ESC	T	Q	S	Obj-ID	State	
\$1B	\$54	\$51	\$53	16-Bit value	16-Bit value	...

The state of a button / switch (**Obj-ID**) is transferred. Which events (down, up, drag) lead to the sending of the feedback is set with the **#TCR** command. If no responses are to be transmitted via the serial interface, this is done with **#TCR 0, 0, Obj-ID**.

State	
1	Up (not pressed)
2	Down (pressed)

Radio group active switch

ESC	T	Q	R	Obj-ID	Group-ID	
\$1B	\$54	\$51	\$52	16-Bit value	16-Bit value	...

The active switch (**Obj-ID**) of a radio group (**Group-ID**) is transmitted with every change. If no answers are to be transmitted via the serial interface, this is done with **#TCR 0, 0, Obj-ID1, ..., Obj-IDn** (all object IDs of the Radiogroup elements).

Keyboard key

ESC	T	Q	K	Obj-ID	Code	
\$1B	\$54	\$51	\$4B	16-Bit value	16-Bit value	...

The last key pressed (**Code**) on the keyboard (**Obj-ID**) is output. The prerequisite is that the keyboard is not connected to an EditBox. If no responses are to be transmitted via the serial interface, this is done with **#TCR 0, 0, Obj-ID**.

Bargraph- /Instrument value

ESC	T	Q	I	Obj-ID	Value	
\$1B	\$54	\$51	\$49	16-Bit value	16-Bit value	...

The new value of the BarGraph / instrument (**Obj-ID**) is output. Which events (down, up, drag) lead to the sending of the feedback is set with the **#TCR** command. If no responses are to be transmitted via the serial interface, this is done with **#TCR 0, 0, Obj-ID**.

Menu entry

ESC	T	Q	M	Obj-ID	ItemNumber	...
\$1B	\$54	\$51	\$4D	16-Bit value	16-Bit value	...

The selected menu item (**ItemNumber**) is output. Which events (down, up, drag) lead to the sending of the feedback is set with the **#TCR** command. If no responses are to be transmitted via the serial interface, this is done with **#TCR 0, 0, Obj-ID**.

ComboBox entry

ESC	T	Q	C	Obj-ID	ItemNumber	...
\$1B	\$54	\$51	\$43	16-Bit value	16-Bit value	...

The selected SpinBox entry (**ItemNumber**) is output. Which events (down, up, drag) lead to the sending of the feedback is set with the **#TCR** command. If no responses are to be transmitted via the serial interface, this is done with **#TCR 0, 0, Obj-ID**.

SpinBox entry

ESC	T	Q	C	Obj-ID	ItemNumber	...
\$1B	\$54	\$51	\$43	16-Bit value	16-Bit value	...

The selected SpinBox entry (**ItemNumber**) is output. Which events (down, up, drag) lead to the sending of the feedback is set with the **#TCR** command. If no responses are to be transmitted via the serial interface, this is done with **#TCR 0, 0, Obj-ID**.

Number of string files loaded

ESC	V	F	C	Number	...
\$1B	\$56	\$53	\$43	16-Bit value	...

The number of strings used from the string files is output.

Content from string register (ASCII)

ESC	V	S	A	String-ID	Length	Char 1	Char 2	...	Char n	...
\$1B	\$56	\$53	\$41	16-Bit value	16-Bit value	8-Bit value	8-Bit value	8-Bit value	8-Bit value	...

The content (8 bits per character) of the string register (**String-ID**) and the **Length** are output. The string does not end with \$00.

Content from string register (Unicode)

ESC	V	S	U	String-ID	Length	Char 1	Char 2	...	Char n	...
\$1B	\$56	\$53	\$55	16-Bit value	16-Bit value	16-Bit value	16-Bit value	16-Bit value	16-Bit value	...

The content (16 bits per character) of the string register (**String-ID**) and the **Length** are output. The string does not end with \$00.

Output register value

ESC	V	R	G	Register-ID	Type	Value	...
\$1B	\$56	\$52	\$47	16-Bit value	16-Bit value	32-Bit value	

The contents of the register (**Register-ID**) and the **Type** are output:

Type	
'I'	Integer
'F'	Float

Number of port blocks

ESC	H	P	I	Available
\$1B	\$48	\$50	\$49	16-Bit value

All available **Addresses** of the connected port blocks are output. Internally there is a block with the address 0, so that \$01 is returned without external hardware.

read port

ESC	H	P	R	Port	Number	State 1	State 2	...
\$1B	\$48	\$50	\$52	8-Bit value	8-Bit value	8-Bit value	8-Bit value	

The status (**State 1**) of the port is output. If the **Number** is > 1, the states following the port module are sent (**State 2, State n**).

Read port-pin

ESC	H	B	R	Portpin	Number	State 1	State 2	...
\$1B	\$48	\$42	\$52	8-Bit value	8-Bit value	8-Bit value	8-Bit value	

The state (**State 1**) of the port pin is output. If the **Number** is > 1, the states following the port module are sent (**State 2, State n**).

Value of analogue input

ESC	H	A	R	Channel	Number	Value 1	Value 2	...
\$1B	\$48	\$41	\$52	8-Bit value	8-Bit value	16-Bit value	16-Bit value	

The value (**Value 1**) of the analog channel is output. If the **Number** is > 1, the measured values following the channel are sent (**Value 2**).

RS232 data

ESC	H	R	R	Length	Data 1	Data 2	...	Data n	...
\$1B	\$48	\$52	\$52	32-Bit value	8-Bit value	8-Bit value	8-Bit value	8-Bit value	...

The data (**Data 1**, **Data 2**, ..., **Data n**) that were received via the master RS232 interface are output. **Length** indicates how much data is sent.

SPI data

ESC	H	S	R	Length	Data 1	Data 2	...	Data n	...
\$1B	\$48	\$53	\$52	32-Bit value	8-Bit value	8-Bit value	8-Bit value	8-Bit value	...

The data (**Data 1**, **Data 2**, ..., **Data n**) that were received via the master SPI interface are output. **Length** indicates how much data is sent.

I²C data

ESC	H	I	R	Length	Data 1	Data 2	...	Data n	...
\$1B	\$48	\$49	\$52	32-Bit value	8-Bit value	8-Bit value	8-Bit value	8-Bit value	...

The data (**Data 1**, **Data 2**, ..., **Data n**) that were received via the master I²C interface are output. **Length** indicates how much data is sent.

Time ASCII output

ESC	W	S	A	ASCII-String	Endin g	...
\$1B	\$57	\$53	\$41		\$00	

The requested time is transmitted in the set format as ASCII. The string ends with a \$00.

Time Unicode output

ESC	W	S	U	Unicode-String	Endin g	...
\$1B	\$57	\$53	\$55		\$00	

The requested time is transmitted in the set format as Unicode. The string ends with a \$00.

Time binary output

ESC	W	S	B	Hour	Minute	Second	Day	Month	Year	Weekday	
\$1B	\$57	\$53	\$42	16-Bit value	16-Bit value	16-Bit value	16-Bit value	16-Bit value	16-Bit value	16-Bit value	...

The requested time is transmitted in binary format in the set format. Day of the week = 0 means Sunday

Current working directory

ESC	F	D	G	Path	
\$1B	\$46	\$44	\$47	'String' completed with \$00	...

The current working directory is output.

All folders and files from directory (binary output)

ESC	F	D	R	Verzeichnis/ Dateiname	Size	Attribute	Time	Date		Ending
\$1B	\$46	\$44	\$52	'String' completed with \$00	32-Bit value	8-Bit value	16-Bit value	16-Bit value	...	\$00

All folders and files in the current working directory are output.

Attribute	
\$01	Read only
\$02	Hidden
\$04	System
\$20	Archive

The time and date give the time stamp of the last change to the file.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Time	Hour [0...23]					Minute [0...59]					Second/2 [0...29]					
Date	Year (from 1.1.1980 0:0:0 Hour) [0...127]					Month [1...12]					Day [1...31]					

All folders and files from directory (ASCII output)

ESC	F	D	A	String	Ending	
\$1B	\$46	\$44	\$41	Size, Attribute, Time, Date, Name	CRLF	...

All folders and files in the current working directory are output as ASCII strings.

All folders from directory (ASCII output)

ESC	F	D	L	Directory-name		Ending
\$1B	\$46	\$44	\$4C	'String' completed with \$00		\$00

All folder names in the current working directory are output as ASCII strings.

Folder/File information

ESC	F	F	I	Directory-/File-name	Size	Attribute	Time	Date	
\$1B	\$46	\$46	\$49	'String' completed with \$00	32-Bit value	8-Bit value	16-Bit value	16-Bit value	...

Folder / file information is output.

Attribute	
\$01	Read only
\$02	Hidden
\$04	System
\$20	Archive

The time and date give the time stamp of the last change to the file.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Time	Hour [0...23]						Minute [0...59]						Second/2 [0...29]		
Date	Year (ab 1.1.1980 0:0:0 Uhr) [0...127]								Month [1...12]			Day [1...31]			

Data from file

ESC	F	R	D	Number	Data 1	Data 2		Data n	
\$1B	\$46	\$52	\$44	32-Bit value	8-Bit value	8-Bit value		8-Bit value	...

The data from the file are output. **Number** indicates the length of the file.

Current project path

ESC	X	P	G	Path	
\$1B	\$58	\$50	\$47	'String' completed with \$00	...

The current project path is output.

Version information

ESC	X	I	V	Version string	...
\$1B	\$58	\$49	\$56	'String' completed with \$00	

The version information of the display is output (e.g. "EA uniTFT V1.4 with capacitive touch")

Display information

ESC	X	I	D	Width	Height	Color depth	Touch	VideoWidth	VideoHeight	...
\$1B	\$58	\$49	\$44	16-Bit value	16-Bit value	8-Bit value	8-Bit value	16-Bit value	16-Bit value	

Display information is output.

Touch	
\$00	No touch
\$03	Resistive touch
\$07	Capacitive touch
\$0F	Simulator

RAM memory information

ESC	X	I	R	Total	Free	...
\$1B	\$58	\$49	\$52	32-Bit value	32-Bit value	

RAM memory information is output.

Memory information

ESC	X	I	I	Total	Free	...
\$1B	\$58	\$49	\$53	32-Bit value	32-Bit value	

Storage information is output.

Hardcopy

ESC	X	H	S	Header	Data	...
\$1B	\$58	\$48	\$53	

A hard copy of the display content is output. The **Header** and **Data** depend on the selected format.

Functions an Calculations

The EA uniTFTs-Series can process small mathematical functions internally at runtime. In addition, with logical operators and options, they offer the possibility to make decisions, similar to an if-statement. In order to be able to evaluate user inputs or optimize the screen layout, calculation commands are also available that can work with object properties, such as bar graph value, last touch position or object width and position. Most functions are available as both integer and floating-point calculations. Care must be taken to stay in the respective number range, or to convert with the cast operator (float or int).

The module works with little-endian (Intel-format), that means the least significant byte is transferred first.

Attention:

Calculations need to be run inside brackets (...). See [command syntax](#) for further details.

signed Integer (32 Bit)	Float IEEE 754 (32-Bit)
-------------------------	-------------------------

Mathematical functions

Arithmetical functions	+, -, * , /, ()		
Absolute value x	abs(x)		
Sign of x (-1, 0, 1): x<0, x==0, x>0	sign(x)		
Modulo x%y	mod(x,y)		
Power x^y	pow(x,y)		
Root	sqrt(var)		
Truncate decimal points	trunc(var)		
Round decimal points	round(var)		
Logarithm (log)	log(var)		
Natural logarithm (ln, Basis e)	ln(var)		
Exponential function base e	exp(var)		
Degrees to rad	rad(deg)		
Rad to degrees	deg(rad)		
Sine	sin(deg)		
Cosine	cos(deg)		
Tangent	tan(deg)		
Arcsine	asin(var)		
Arc cosine	acos(var)		
Arctangent	atan(var)		
Arctangent, right quadrant	atan(y,x)		
Minimum	min(a,b,c...)		

Maximum	max(a,b,c...)		
Average	avg(a,b,c...)		
Random value sv <= x <= ev (max 65535)	rand(sv,ev)		
Random value 0 <=x <= ev (max 65535)	rand(ev)		
Random value 0<= x<=1000	rand()		

Register increment / decrement

pre-/post- increment	++Rx, Rx++		
pre-/post- decrement	--Rx, Rx--		

Cast Integer ↔ Float

Integer calculation, return Float	int(calculation)		
Float calculation, return Integer	float(calculation)		

Bit Operatoren

Low Byte	loB(x)		
High Byte	hiB(x)		
Low Word	loW(x)		
High Word	hiW(x)		
Low Byte from High Word	hwloB(x)		
High Byte from High Word	hwHiB(x)		
AND	&		
OR			
NOT	~		
XOR	^		
Shift left / right	<<, >>		
Set bit (Bit-Nr. 0..31)	bitS(value, Bit-No.)		
Clear bit(Bit-Nr. 0..31)	bitC(value, Bit-No.)		
Exor bit (Bit-Nr. 0..31)	bitX(value, Bit-No.)		
Test bit (Bit-Nr. 0..31) returns true or false	bitT(value, Bit-No.)		

Logical Operators

AND	&&		
OR			
NOT	!		
Equal, not equal	==, !=		
Less, less than or equal	<, <=		
Greater, Greater than or equal	>, >=		

Decision

If-Then-Else-function	ifte(condition, value true, value false)		
-----------------------	--	--	--

Object commands, general

Width (without transform)	objW(id)		
Height (without transform)	objH(id)		
Position X (actual anchor, for groups: relative to parent object)	objX(id)		
Position Y (actual anchor, for groups: relative to parent object)	objY(id)		
Screen-Position X (given anchor, also for groups: Screen coordinates)	objX(id, anchor)		
Screen-Position Y (given anchor, also for groups: Screen coordinates)	objY(id, anchor)		
Scaled width	objSW(id)		
Scaled height	objSH(id)		
Shear X	objSX(id)		
Shear Y	objSY(id)		
Rotation	objR(id)		
Opacity	objO(id)		
Layer	objL(id)		
Read actual used style	objC(id)		
Read active anchor	objA(id)		
Object exists?	objE(id)		
Object visible?	objV(id)		
Get Obj-ID from screen-coordinates	objXY(x,y)		
Get Obj-ID from screen-coordinates. If onlyrect is true, then fast calculation is done and only the object's rectangular size is checked (transparencies are ignored)	objXY(x,y, onlyrect)		

Object commands menu

Get last valid menu item	objML(id)		
Get active shown menu item (0=closed)	objMV(id)		
Check state of item (1=checked, 0=unchecked)	objMC(id, item)		
Get enable state (1=enabled, 0=disabled)	objME(id, item)		

Object commands ComboBox

Get last valid item	objCL(id)		
Get active shown item (0=closed, -1=no item visible))	objCV(id)		

Get enable state (1=enabled, 0=disabled)	objCE(id, item)		
--	-----------------	--	--

Object commands SpinBox

Get last valid items	objBL(id)		
Get last valid item of box no in spinbox group	objBL(id, boxnr)		
Get active shown items	objBV(id)		
Get active shown item of box no in spinbox group	objBV(id, boxnr)		
Get enable state (1=enabled, 0=disabled)	objBE(id, item)		

Object commands StringBox

Visible lines	objTV(id)		
Number of paragraphs (= number of lines without AutoWrap)	objTA(id)		
Number of lines (after AutoWrap)	objTN(id)		
Number of strings (after AutoWrap)	objTN(id, Absatz nr)		
Visible top line	objTL(id)		
String line start (after AutoWrap)	objTL(id, Absatz nr)		
Paragraph from line number	objTS(id, line nr)		

Object properties

Read user value integer (#OUI)	objUI(id)		
Read user value float (#OUI)	objUF(id)		

Object properties Bargraph/Instrument

Current value	objIV(id)		
Value drawn on the screen (not necessarily the same for animations as objIV (id))	objID(id)		
EndValue	objIE(id)		
StartValue	objIS(id)		

Object properties paths

Length	pathL(id)		
X-Coordinate of a path with distance	pathX(id,distance)		
Y-Coordinate of a path with distance	pathY(id,distance)		
Tangent angle of a point on a path	pathR(id,distance)		

Touch functions

Touch button/switch state =1 unpressed =2 pressed	butS(id)		
Radio-group: Active switch (id)	butR(id)		
Last used touch button	butI()		

Letzter Keyboard code	butC()		
Last entered keyboard code)	touchA()		
Last touch position X (Down- or Drag-Event)	touchX()		
Last touch position Y (Down- or Drag-Event)	touchY()		
Touch position X of touch point number no (Down- or Drag-Event))	touchX(nr)		
Touch position X of touch point number no (Down- or Drag-Event)	touchY(nr)		

Decomposition of input elements (Menu)

Get root from item	menR(item)		
Get menu from item	menM(item)		
Get sub-menu from item	menS(item)		
Make item from root, menu, sub-menu	menRMS(r,m,s)		

Decomposition of input elements (SpinBox)

Entry box 1 (8 Bit value)	spin1(item)		
Entry box 2 (8 Bit value)	spin2(item)		
Entry box 3 (8 Bit value)	spin3(item)		
Entry box 4 (8 Bit value)	spin4(item)		
Make item from individual spinbox values ((e4<<24) (e3<<16) (e2<<8) e1)	spinE(e1,e2,e3,e4)		

I/O Ports

Port state (a = port expander 0..15)	port(a)		
Port-pin state (a = Pin number 0..127)	bit(a)		

Analogue input

Analog value (a=0..3))	analog(a)		
------------------------	-----------	--	--

RS232 Master interface

Length of received data bytes	mstRA()		
-------------------------------	---------	--	--

Timer

Set timer start value (10 ms Timer)	timer(startvalue)		
Read timer (10 ms Timer)	timer()		

Time functions/ RTC

Time and date are calculated by the module internally in seconds with epoch = 1.1.2000 at 00:00:00 o'clock with SINT32 (= datetime-value). Thus, the maximum period used by the module is from 1932 - 2067. The base date can be changed with the **#WDY** command. To calculate time periods, it's always necessary to convert the time/date to seconds first, then carry out the calculation. The result can then be converted back again to minutes, hours, day, month and year.

Convert current date into datetime-value	date()		
Convert number of days into seconds	date(D)		

Convert day + month + year (1932 - 2067) into datetime-value	date(D,M,Y)		
Convert actual time into seconds (starting with 0:00:00)	time()		
Convert number of hours into seconds (? h-3600)	time(h)		
Convert hours and minutes into seconds (? h-3600 + m-60)	time(h, m)		
Hour, minutes and seconds into seconds (? h-3600 + m-60 + s)+ s)	time(h, m, s)		
Convert current date and time into datetime-value	datetime()		
Hour+min+sec+day+month+year into datetime-value	datetime(h,m,s,D,M,Y)		
Actual year	year()		
Calculate year from datetime-value	year(a)		
Actual month	month()		
Calculate month from datetime-value	month(a)		
Actual day	day()		
Calculate day from datetime-value	day(a)		
Actual week day (0-6=Sunday..Saturday)	weekday()		
Calculate week day from datetime-value	weekday(a)		
Actual hour	hour()		
Calculate hour from datetime-value	hour(a)		
Actual minute	minute()		
Calculate minute from datetime-value	minute(a)		
Actual second	second()		
Calculate seconds from datetime-value	second(a)		

String register functions

Length of string (Register no)	strL(nr)		
ASCII-Code from string position	strA(nr, offset)		
Unicode-Code from string position	strU(nr, offset)		
Convert numerical string to SINT32 or float.	strV(nr)		
Comparison of two string registers =0 both strings are the same >0 first unequal character in n1 is greater then the one in n2 <0 first unequal character in n1 is smaller then the one in n2	strC(n1, n2)		
Compare the first len characters of two strings	strC(n1, n2, len)		

Compare len characters of two strings, starting with offset	strC(n1, n2, len, offset)		
Compare len characters of two strings, starting on different offsets	strC(n1, n2, len, offset1, offset2)		
Search for code in string from left =0 not found >0 Offset of the first code found	strFL(nr, code)		
Search for code in string from left starting with offset	strFL(nr, code, offset)		
Search for code in string from right	strFR(nr, code)		
Search for code in string from right starting with offset	strFR(nr, code, offset)		
Search for string from another register (n2) in string (n1)	strFS(n1, n2)		
Search for string from another register (n2) in string (n1) from offset.	strFS(n1, n2, offset)		
Check if code is a character	isAL(code)		
Check if code is digit or character ist	isAN(code)		
Check if code is a small letter	isLO(code)		
Check if code is a big letter	isUP(code)		
Check if code is whitespace	isWS(code)		
Check if code is a digit	isDD(code)		
Check if code is a hexadecimal number	isDH(code)		
Check if code is a binary number	isDB(code)		

Array functions

Get max. elements for a new array	arE(-1)		
Get array element count (0 = not exist)	arE(id)		
Get array value	arV(id, index)		
Get next array value from read index and increment read index	arV(id)		
Get array read index	arR(id)		
Get array write index	arW(id)		

Color commands

Get red channel from a 24 bit RGB value	getR(x)		
Get green channel from a 24 bit RGB value	getG(x)		
Get blue channel from a 24 bit RGB value	getB(x)		
Combine 3 single color channels to one 24 bit RGB value	RGB(R, G, B)		

Read 24 bit RGB value from a color ramp	rampRGB(nr, offset)		
Read opacity value from a color ramp	rampO(nr,offset)		
Read 24 bit RGB value from a display-pixel	tftRGB(x,y)		

File and directory commands

File exists? (<Path/filename> in string register no)	fileE(nr)		
File size? (<Path/filename> in string register no))	fileS(nr)		
File attribute? (<Path/filename> in string register no)	fileA(nr)		
Get fat-time of file (<Path/filename> in string register no))	fileT(nr)		
Get fat-date of file (<Path/filename> in string register no)	fileD(nr)		
Convert datetime-value to fat-time	fatT(datetime)		
Convert datetime-value to fat-date	fatD(datetime)		
Convert fat-time and date to datetime-value	fattime(Fat-Time, Fat-Date)		
Get actual read-pointer position (#FRO)	fposR()		
Get actual read-pointer position calculated from file end (=negativ)	fposR(-1)		
Get actual file size of opened read file	fposR(1)		
Get actual write-pointer position (#FWO)	fposW()		
Get actual write-pointer position calculated from file end (=negativ)	fposW(-1)		
Get actual file size of opened write file	fposW(1)		
Get actual maximum file size	fposW(2)		
Get maximum file size of file from string register no (from V1.4)	fileM(nr)		
Get storage total size	memST()		
Get storage free size	memSF()		
Get object RAM total size	memRT()		
Get object RAM free size	memRF()		
Get object RaM max block size	memRB()		

Module commands

Firmware version	version()		
Last frame rate (fps)	fps()		
Get touch type (=0 no, =1 resistive, =2 PCAP))	touchT()		
Screen width (#XCV)	scrW()		

Screen height (#XCV)	scrH()		
Screen width of hardware, not depending on (#XCV)	scrW(1)		
Screen height of hardware, not depending on (#XCV)	scrH(1)		
Video width	vidW()		
Video height	vidH()		
Number of video objects	vidC()		
Get actual backlight brightness	ledB()		
Get backlight autostate brightness (state=0..2)	ledB(status)		
Get backlight autostate status	ledS()		
Errorstring available?	error()		
Errorstring available? copy Errorstring to Stringregister nr (#VSL)	error(nr)		
Errorstring available? copy Errorstring to Stringreg nr and clear Errorstring	error(nr,1)		

Priority list of all operators

12	()	Parentheses / function call (highest priority))
11	++	Register increment
	--	Register decrement
	+	Sign
	-	Sign
	!	Logical NOT
	~	Bitwise NOT
10	*	Multiplication
	/	Division
9	+	Addition
	-	Subtraction
8	<<	Shift left
	>>	Shift right
7	<	Less than
	<=	Less than or equal
	>	Greater than

	>=	Greater than or equal
6	==	Equal
	!=	Unequal
5	&	Bitwise AND
4	^	Bitwise XOR
3		Bitwise OR
2	&&	Logical AND
1		Logical OR (lowest priority))

HARDWARE

The EA uniTFTs-Series consists of a TFT-Display with LED backlight, driven by an integrated driving circuit, which is dimmable using software commands. In 24/7 operation the backlight can be dimmed automatically to increase the LED life-time and save energy.

The module is designed to work with 3.3 VDC. Serial data transfer is possible through RS232, SPI, I²C or direct via USB protocol.

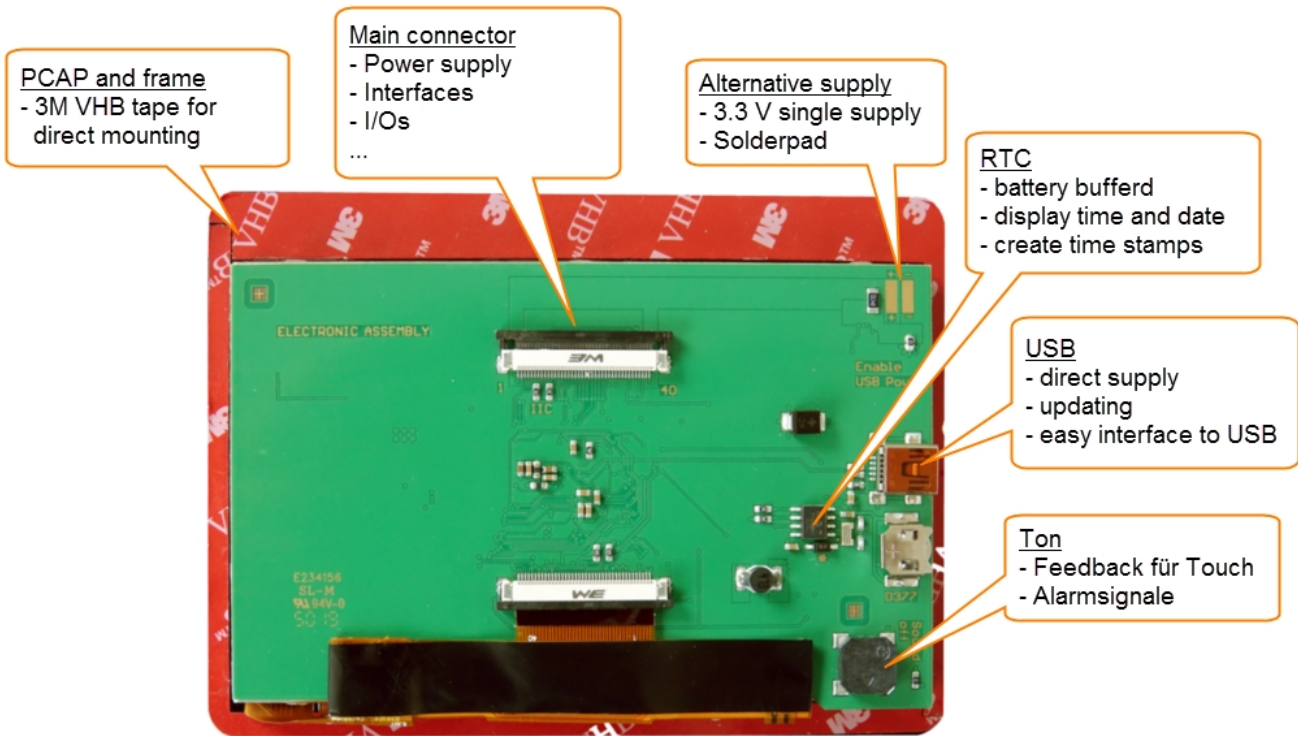
For simple control tasks, the module has 8 freely usable I/Os (expandable up to 136), 4 analogue inputs, one PWM output and 3 serial interfaces (RS232, SPI and I²C).

The modules do have an integrated capacitive touch panel. By touching the display you can enter data and make adjustments via menu or bar graph. The labelling, size and shape of the "keys" is flexible and can also be changed during runtime (different languages, icons). The drawing of the individual "keys", as well as the labelling is completely taken over by the built-in software. The capacitive touchpanel has an robust glass surface that can also be operated with thin gloves.

Front view (example EA uniTFTs043-ATC)



Rear view (example EA uniTFTs043-ATC)





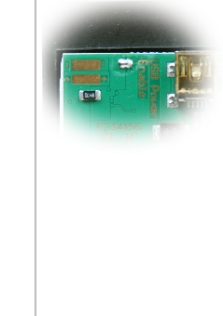
Pin assignment

Pin assignment for ZIF connector. It's an FPC connector with 40 positions and 0.5 mm pitch. Bottom contact.

Pin	Symbol	I/O	Description	
1	GND		Ground 0 V	
2	VDD		Power Supply 3.3 V	Power supply or 3.3V output at USB operation (max. 200mA)
3	$\overline{\text{RES}}$	I	$\overline{\text{Reset}}$	internal Pull-Up: (10..75 k Ω)
4	$\overline{\text{CS}}$	I	SPI: Chip Select	internal Pull-Up: (1 M Ω)
5	MOSI	I	SPI: MOSI	
6	MISO	O	SPI: MISO	
7	CLK	I	SPI: CLK	internal Pull-Up: (1 M Ω)
8	RxD	I	RS232: Receive Data	internal Pull-Up: (1 M Ω)
9	TxD	O	RS232: Transmit Data	
10	DE	O	RS485: Transmit Enable	
11	SDA	I/O	I ² C: Serial Data	internal Pull-Up: (10 k Ω); Pull-Up resistors can be changed
12	SCL	I	I ² C: Serial Clock	internal Pull-Up: (10 k Ω)
13	$\overline{\text{SBUF}}$ $\overline{\text{TESTMODE}}$	I	Low: Data available in send buffer PowerOn Low: Test mode	internal Pull-Up: (10 k Ω)
14	$\overline{\text{DPROT}}$	I	High: Small-/Shortprotokoll active Low: deactivated	internal Pull-Up: (10 k Ω)
15	A/D 0	I	Analog Input 0	internal Pull-Down: (1 M Ω)
16	A/D 1	I	Analog Input 1	
17	A/D 2	I	Analog Input 2	
18	A/D 3	I	Analog Input 3	
19	I/O 0.0	I/O	I/O 0.0 (Bit 0)	internal Pull-Up: (1 M Ω), Reset-state: Tri-State, default: High
20	I/O 0.1	I/O	I/O 0.1 (Bit 1)	
21	I/O 0.2	I/O	I/O 0.2 (Bit 2)	
22	I/O 0.3	I/O	I/O 0.3 (Bit 3)	
23	I/O 0.4	I/O	I/O 0.4 (Bit 4)	
24	I/O 0.5	I/O	I/O 0.5 (Bit 5)	
25	I/O 0.6	I/O	I/O 0.6 (Bit 6)	
26	I/O 0.7	I/O	I/O 0.7 (Bit 7)	
27	PWM	O	PWM-Output	
28	DNC	---	Do not connect	Reserved for future use
29	DNC	---	Do not connect	
30	DNC	---	Do not connect	
31	DNC	---	Do not connect	
32	DNC	---	Do not connect	
33	DNC	---	Do not connect	
34	BUZZER	O	Sound	PWM output for external speaker

Power supply

The modules can be powered in three different ways:

	Ziff-Connector	Solderpadas	USB
Spannung	3.3 V	3.3 V	5 V (Power over USB)
USB Lötbrücke	open	open	closed
			

Attention:

In order to avoid fault currents, the "USB Power Enable" solder bridge must be set correctly. The solder bridge is closed by default. The internal voltage regulator is now active and generates 3.3 V from the connected USB supply. If an additional 3.3 V is now supplied externally, fault currents occur.

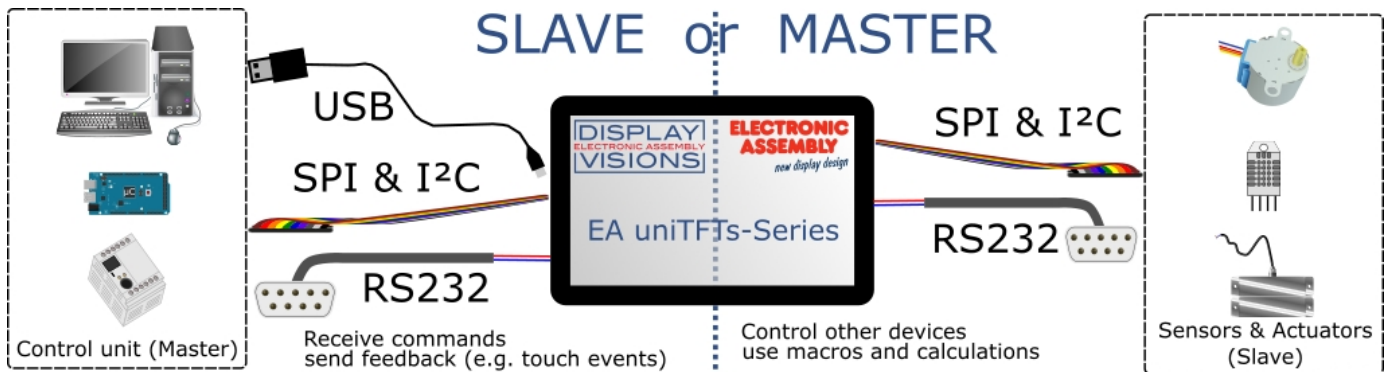
Serial interfaces

The module provides 4 serial interfaces, including RS232, SPI, I²C and USB. In addition to the USB interface, the other interfaces can change your behaviour:

They can either be used to connect to an external host, i.e. to a higher-level controller, or used as a master interface. By default, all interfaces are parameterized as slaves and accept the [commands](#).

Parameterized as a master interface, it enables the control of external sensors and actuators. The display module behaves here as a master.

As already described, the interfaces behave as slave interfaces by default and accept commands. However, as soon as a master interface command ([#H...](#)) is executed, the interface gets master functions.



RS232

RS232 is a standard for a serial interface.

The EA uniTFT provides one RS232 interface, that can be operated as slave (default) or as master: As slave interface it is used to communicate with the display. All data sent to the display are interpreted as a command (with and w./o. Small-/Short-Protocol). If you would like to send and receive any data via RS232 to other devices then you have to use it as master. Those are handled via [#H commands](#).

The transmission is serially asynchronous. Thus the data is converted into a bit stream and transmitted. There is no clock signal, so transmitter and receiver need to work with the same data rate (so-called baud rate). RS232 is a voltage interface, such that data is transmitted using changing voltage levels. In the PC world and industrial controls, levels of + 12V and - 12V are defined as standard. With boards or micro-controllers levels of 0V and VDD (in the case of EA uniTFTs-Series 3.3 V) are common. To adjust the signal levels, there are some possibilities in the form of level shifters (e.g., ICL232, MAX202). RS232 consists of "listening" and "talking" lines that are crossed between the two parties.

In the EA uniTFTs-Series, the data format is fixed to 8-N-1:

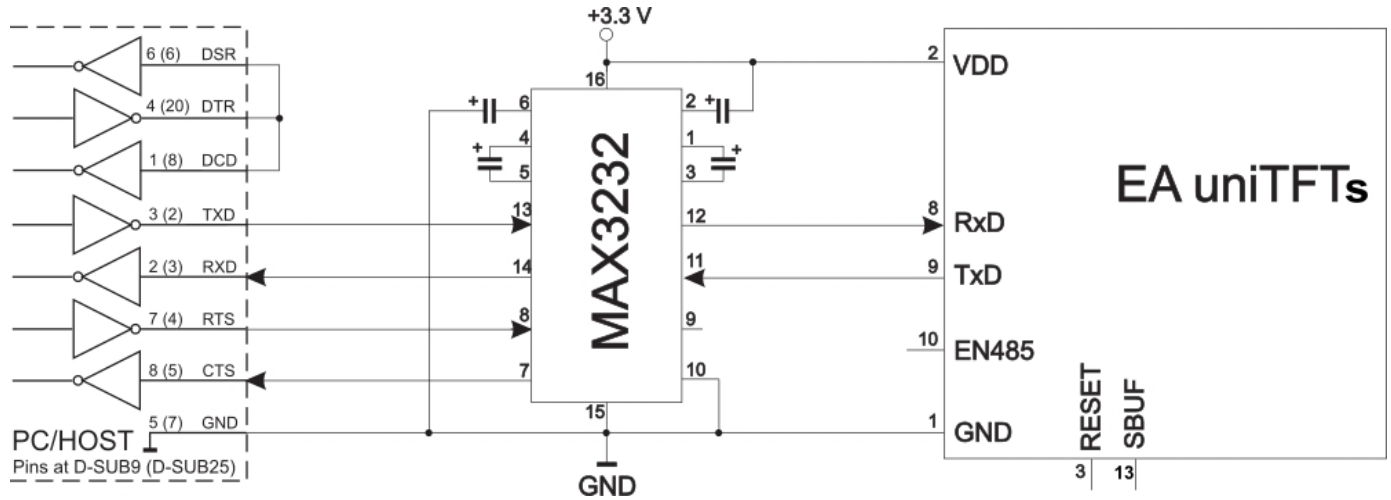


The EA uniTFTs-Series works with the following baud rates:

Baud	Error	Baud	Error
9600	+0.04	115200	+0.64
19200	-0.08	230400	-0.80
38400	+0.16	460800	+2.08
57600	-0.08	921600	-3.68

The parameters (baud rate) are set using command [#XCR](#) (higher-level control unit), and the master interface is set with the command [#HRP](#). Those definition can be done in start.emc e.g.

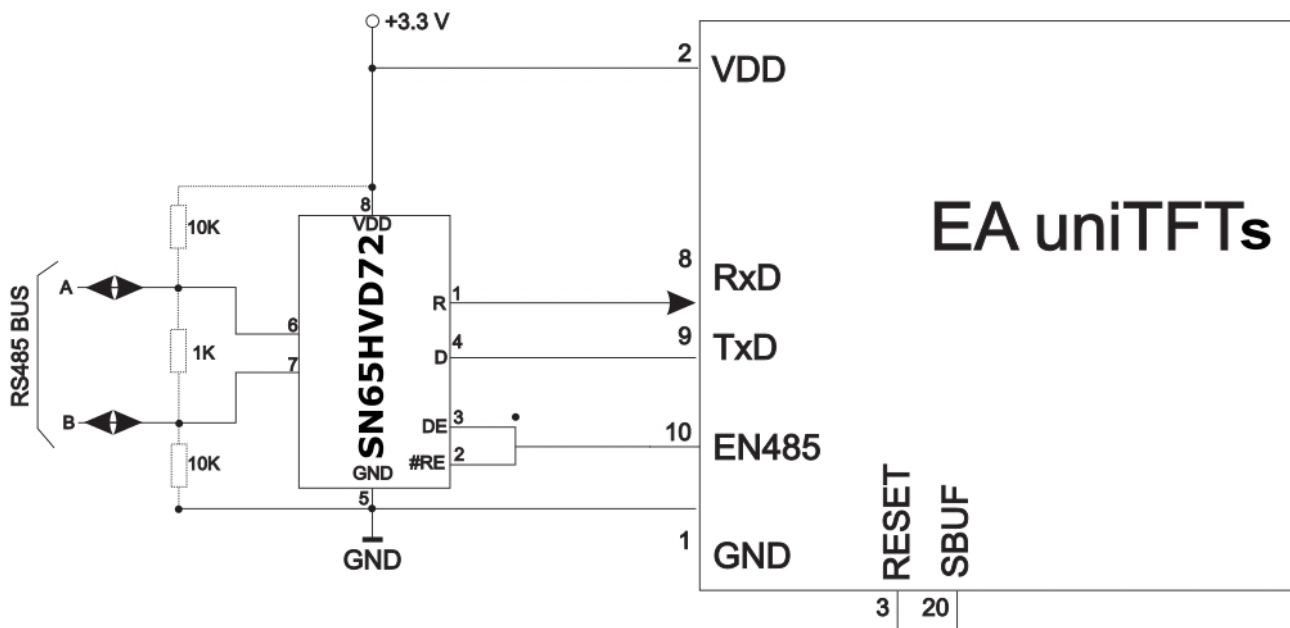
Application notes



RS232 V24 - Interface to a PC (EA uniTFTs)

RS485 / RS422 interface

With this simple external IC a communication to any RS-485 and RS-422 can be done.



RS485 - Interface to a PLC (EA uniTFT)

SPI

The **S**erial **P**eripheral **I**nterface is a bus system for serial synchronous data transfer.

The EA uniTFT provides a SPI interface: AS default the interface has Slave functionality and is used to communicate with the display. All data sent to the display are interpreted as a command (with and w./o. Small-/Short-Protocol). Would you like to send and receive any data via SPI to other devices like temperature sensor, then you have to use the Master interface. Those are handled via [#H commands](#).

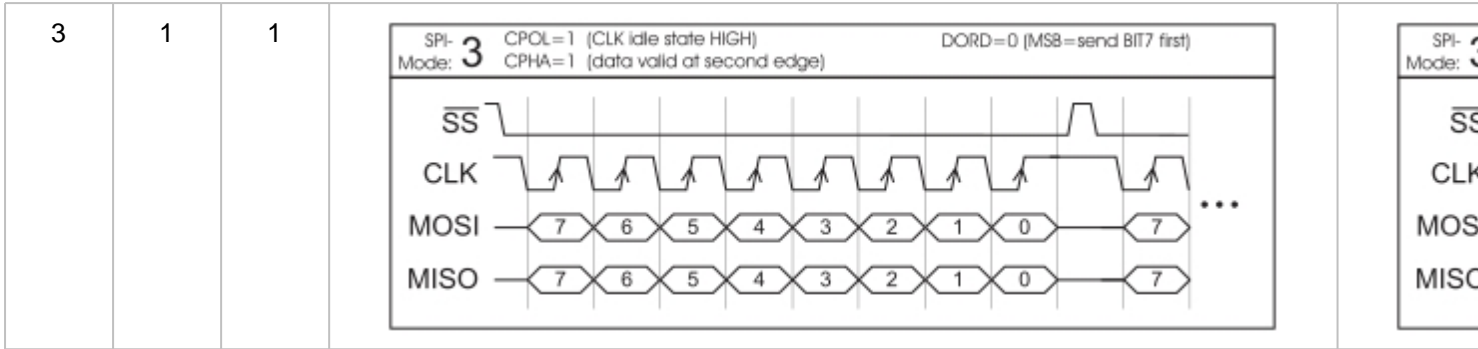
The SPI is working with 4 lines:

- MOSI (**M**aster **O**ut → **S**lave **I**n) or SDO (Serial Data Out) or DO
- MISO (**M**aster **I**n ← **S**lave **O**ut) or SDI (Serial Data In) or DI
- SCK (**S**erial **C**lock) - Shift clock
- SS (**S**lave **S**elect → Addressing) or CS (Chip Select)

SPI works with a bidirectional transmission principle, meaning that data is exchanged between the connected devices at the same time. The communication is controlled by the master using the SCK line.

The protocol for data transfer is not defined in SPI, therefore there are different configuration possibilities, which are defined by the parameters Clock Polarity, Clock Phase and Data Order. The default setting is SPI mode 3 with DORD = 0. The commands **#XCS** and **#HSP** (master interface) set the mode 0..3. Alternatively the command can be stored directly into the boot file <start.emc>.

Mode	CPOL	CPHA	DORD (0) - MSB First	DORD (1)
0	0	0	<p>SPI-Mode: 0 CPOL=0 (CLK idle state LOW) CPHA=0 (data valid at first edge) DORD=0 (MSB=send BIT7 first)</p>	<p>SPI-Mode: 0</p>
1	0	1	<p>SPI-Mode: 1 CPOL=0 (CLK idle state LOW) CPHA=1 (data valid at second edge) DORD=0 (MSB=send BIT7 first)</p>	<p>SPI-Mode: 1</p>
2	1	0	<p>SPI-Mode: 2 CPOL=1 (CLK idle state HIGH) CPHA=0 (data valid at first edge) DORD=0 (MSB=send BIT7 first)</p>	<p>SPI-Mode: 2</p>



The maximum clock frequency is 1 MHz. The module needs some time to prepare data for transfer. That means a wait cycle (no activity on the SCK-line) of at least **50 μs** is required before reading data.

I²C

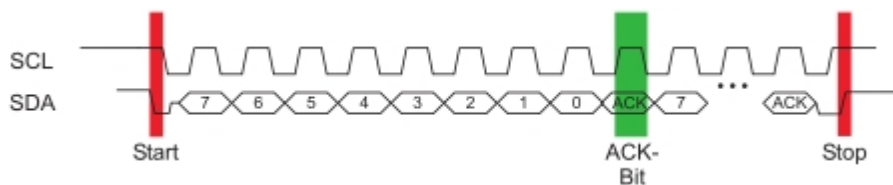
I²C stands for **I**nter-**I**ntegrated **C**ircuit and is a serial data-bus developed by Phillips.

The EA uniTFT provides one I²C interface: As default the interface is parametrized as Slave and used to communicate with the display. All data sent to the display are interpreted as a command (with and w./o. Small-/Short-Protocol). Would you like to send and receive any data via I²C to any other device like temperature sensor, then you have to use the Master functionality (pins 43 and 44). Those are handled via [#H commands](#).

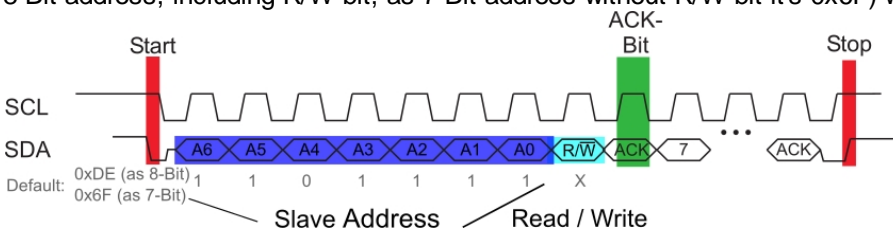
The bus is a Master-Slave implementation and needs 2 signal lines:

- SCL (**S**erial **C**lock **L**ine)
- SDA (**S**erial **D**ata **L**ine)

The electrical specification defines that both lines are terminated with a pull-up resistor at VDD, because all devices connected to the bus have open collector outputs. The bus clock is always given by the master, which controls the entire communication:



After the start condition, the slave address follows. In this case, bit 0 is the so-called R/W bit and determines whether the slave should be read (1) or data is transmitted (0). The data exchange takes place until the master executes the stop condition. More detailed information can be found in the I²C specification. The default I²C bus address is 0xDE (as 8-Bit address, including R/W bit, as 7-Bit address without R/W bit it's 0x6F) when writing to the slave unit.



The command [#XCI](#) and [#HIP](#) (master interface) can change the I²C write address to any other address. Alternatively the command can be written directly into the boot file <start.emc>.

The maximum frequency in slave mode is 400 kHz, the master interface is capable up to 1 MHz. The module needs some time to prepare data for transfer. That means a wait cycle (no activity on the SCL-line) of at least **50 μs** is required before reading data.

USB

The **Universal Serial Bus** is a serial bus system for interfacing a PC with other peripherals. It's based on differential data transfer. The bus topology is a strict master-slave communication (Exception: On the Go devices). In the case of EA uniTFTs-Series the PC/Master needs to coordinate the communication. The module has a CDC (Communications Device Class) and is found by Windows PC's as a virtual COM-Port:

Description	Value
Device Class	2
USB Vendor ID	0x2DA9
USB Product ID	0x2454
Device description	EA uniTFT

To program the module, adjust settings or to perform initial tests, we recommend using the USB interface. It's easy to connect, the transfer rate is fast and no interface parameters need to be specified. The driver for Windows can be downloaded on our web-page: http://www.lcd-module.de/fileadmin/downloads/EA_CDCdriver_V5_2.zip

Attention:

A [protocol](#) has to be used in USB CDC mode. It's impossible to use the USB interface without a protocol, which means pin 22 (primary connector) must not be set to GND. The high-speed connection of USB leads to buffer overflow, which are prevented by the protocol.

Touch-panel

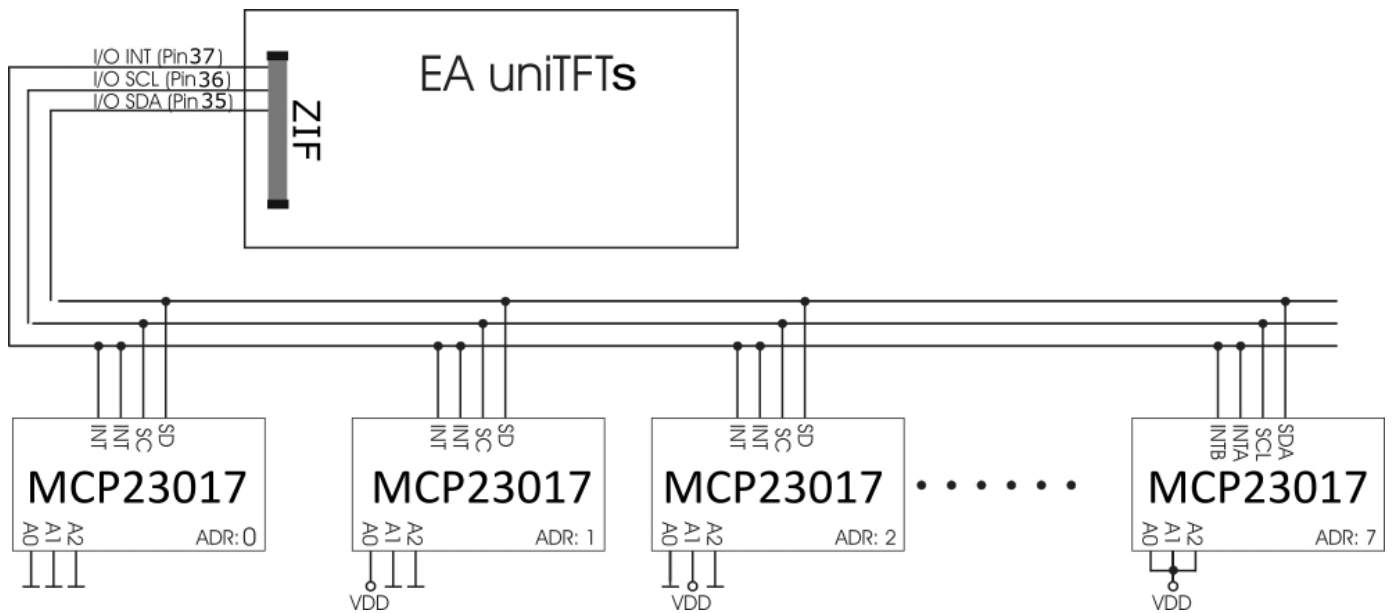
The modules all do have a optically bonded capacitive touchpanels, which is used for mounting, too. By touching the display you can enter data and adjust settings via menus or bar graphs. The labelling of the "keys" is flexible and can also be changed during runtime (different languages, icons). The drawing of the individual "keys" as well as the labelling is completely handled by the built-in software.

I/O - digital in- and outputs

The module has 8 digital I/Os (CMOS level, non-floating). The input range is 0... 3.3 V. All 8 I/Os have a weak pull-up at 1 M Ω and are set as inputs after reset.

Remark: The logic is not designed for time-critical operations; i.e. it is not a real-time operating system

By using one or more external (max. 8) MCP23017-E (16 I/Os per IC), the total number of I/Os can be expanded up to 136. Therefore the port-expanders are connected to pins 35-37 (see application example).



The maximum power of the MCP23017-E is 700 mW in total. The maximum current load for a single pin is 25 mA, which makes it possible to directly operate a low current LED. If a higher load is required, the I/O current must be amplified with suitable circuitry, e.g. through an external transistor. For more details see [Electrical characteristics](#)

The overview of the software commands for the I/Os can be found under the chapter '[I/O Port](#)'.

Analogue input

The module uses 4 analogue inputs with a resolution of 12 bit and an input range of 0 V...VDD. The input range can be arbitrarily expanded with the help of external voltage dividers or amplifiers. Every single input is referenced to GND and has an input resistance of about 1 M Ω . The absolute accuracy is 11 bits, as reference VDD/2 is used.

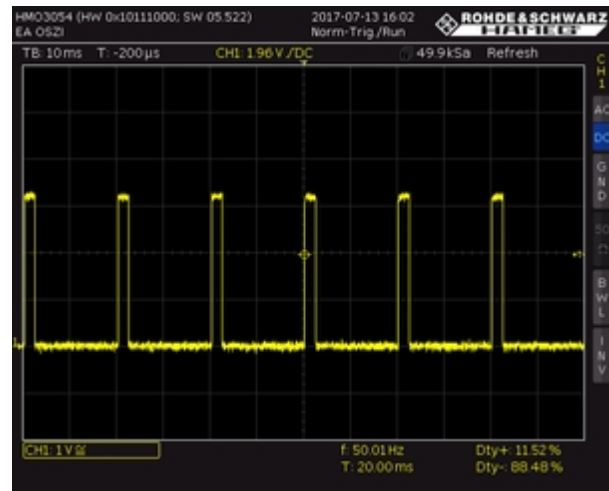
This enables the display to measure analogue voltages, e.g. to display or save the values for further processing. The exceeding or undershooting of a threshold can also be used to trigger an alarm.

The overview of the software commands for the analogue inputs can be found under the chapter ['Analogue Input'](#)



PWM output

The module has the option of controlling external components via a PWM signal (pulse width modulation). At constant frequency (adjustable from 2 Hz to 1 MHz #HFO), the duty cycle of a rectangular pulse is changed. Modulation changes the ratio between the on- and off-time and thus the characteristics of the output signal. In this way, electromechanical components such as motors can be driven or even a quasi-analogue voltage can be generated. The variation of the duty cycles supports a low engine speed/voltage with a short start-up time or a high motor speed/voltage with a long start-up time. The output levels are at 0V and VDD.



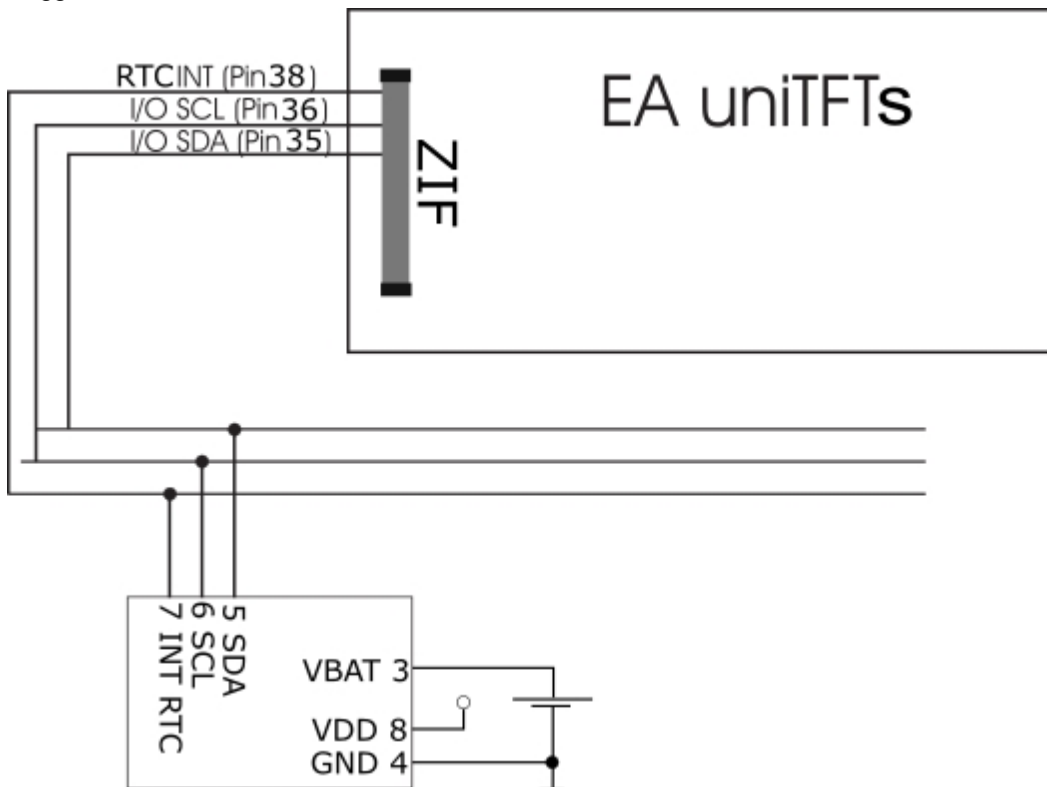
Time / RTC

EA uniTFTs035-ATC and EA uniTFTs043-ATC has a built-in RTC clock. It provides a time-stamp for log files and the time and date can be displayed on the screen directly. On delivery, the time is set to Central European Time (CET / MEZ). Depending on the location it may be necessary to set the device to local time (#WTD). In the event of a voltage drop, or when the module is switched off, the clock is powered by a button cell (D377), so that the correct time is retained.

Due to component tolerances and temperature fluctuations, deviations of up to 0.02% are possible. The deviation can be reduced by repeatedly adjusting the time (#WTD).



For the smaller displays EA uniTFTs020 and uniTFTs028 an external RTC can be connected. We suggest the **MCP7940N**:



Memory

The module has a built-in flash memory. The size is 31 MByte.

This memory is used to store all data, whether generated at runtime, e.g. log files, or pre-loaded as project data, such as macro files, pictures, animations and icons.

Attention:

Flash memories have limited erase / write cycles due to their design. The memory module used in the uniTFTs can typically safely execute 100,000 cycles. In order to write data, a block of memory may have to be erased, typically 30 ms are required for erasing, but it can take up to 400 ms. This must be taken into account in the macro sequence when write file commands are executed.

Elektrische Spezifikation EA uniTFTs020-ATC

Value	Condition	min.	typ.	max.	Unit
Supply current 3.3 V	Backlight 0%		113		mA
	Backlight 100%		222		mA
	Backlight 150%		284		mA
Supply current USB (5 V)	Backlight 0%		82		mA
	Backlight 100%		158		mA
	Backlight 150%		201		mA
Brightness (100%)	with PCAP	700	850		cd/m ²

Elektrische Spezifikation EA uniTFTs028-ATC

Value	Condition	min.	typ.	max.	Unit
Supply current 3.3 V	Backlight 0%		113		mA
	Backlight 100%		236		mA
	Backlight 150%		309		mA
Supply current USB (5 V)	Backlight 0%		84		mA
	Backlight 100%		169		mA
	Backlight 150%		219		mA
Brightness (100%)	with PCAP	700	780		cd/m ²

Elektrische Spezifikation EA uniTFTs035-ATC

Value	Condition	min.	typ.	max.	Unit
Supply current 3.3 V	Backlight 0%		115		mA
	Backlight 100%		269		mA
	Backlight 150%		368		mA
Supply current USB (5 V)	Backlight 0%		84		mA
	Backlight 100%		192		mA
	Backlight 150%		260		mA
Brightness (100%)	with PCAP	480	600		cd/m ²

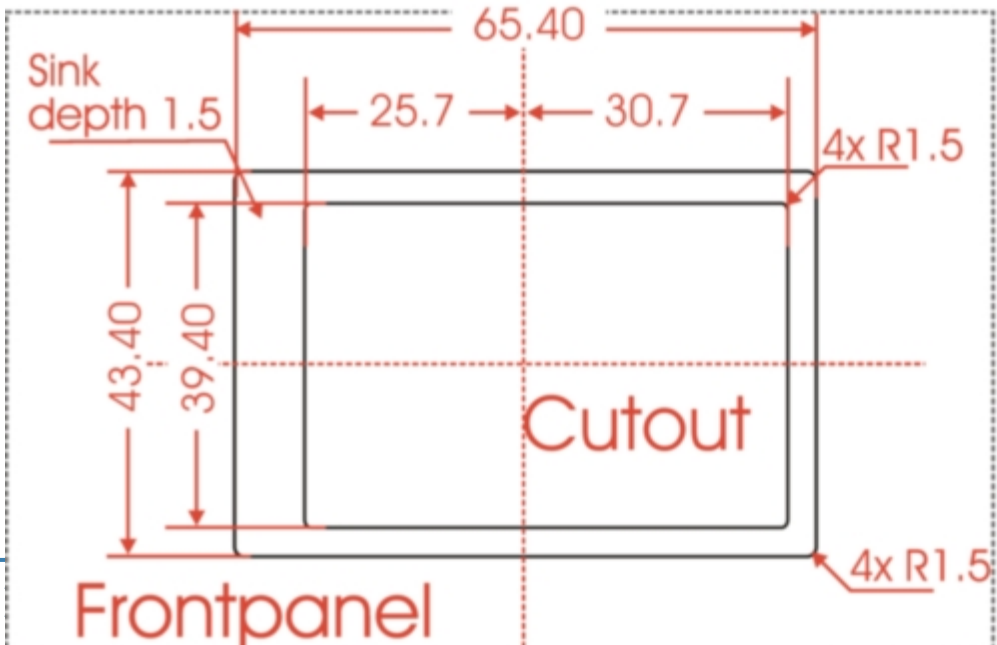
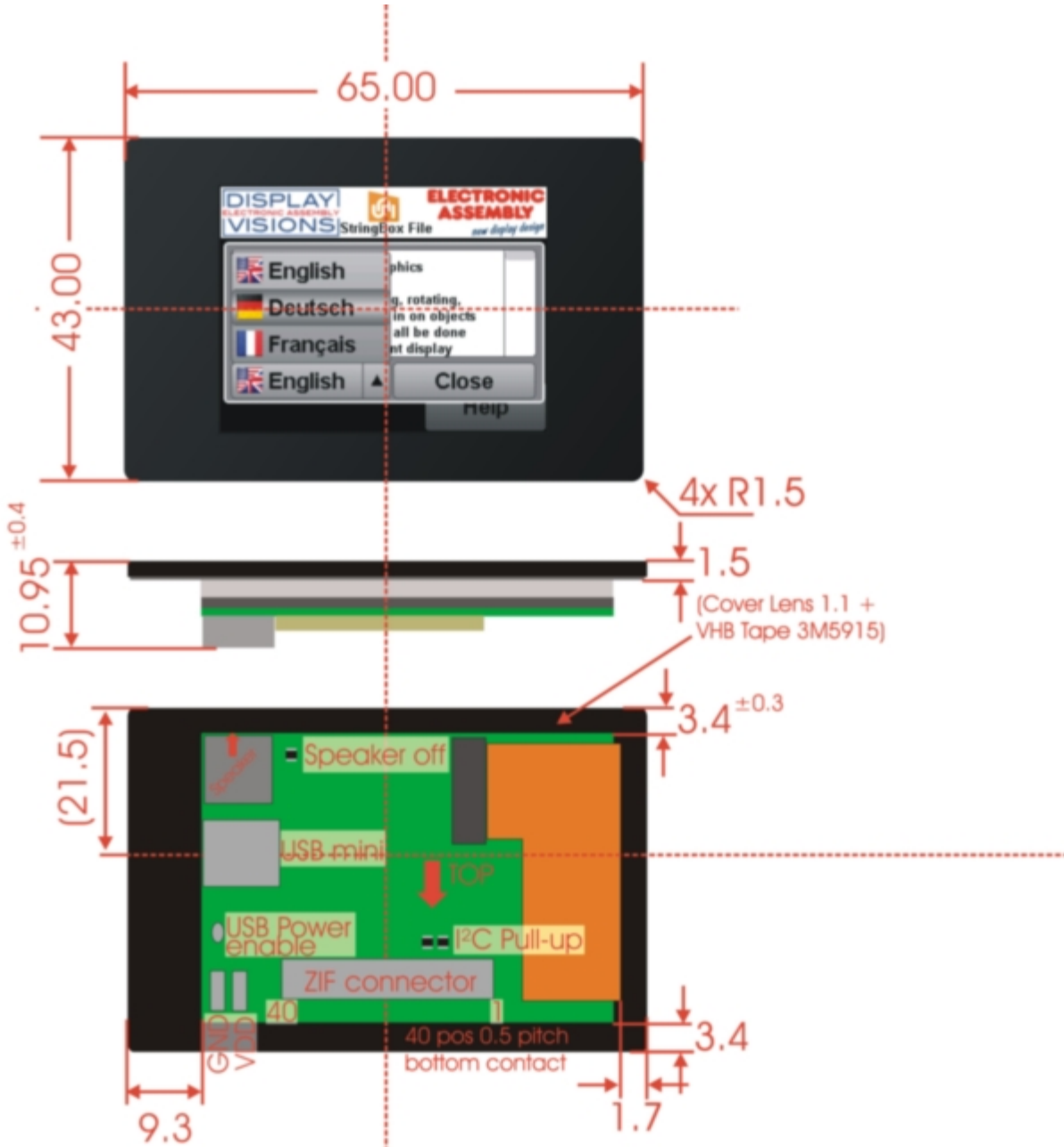
Elektrische Spezifikation EA uniTFTs043-ATC

Value	Condition	min.	typ.	max.	Unit
Supply current 3.3 V	Backlight 0%		131		mA
	Backlight 100%		362		mA
	Backlight 150%		562		mA
Supply current USB (5 V)	Backlight 0%		89		mA
	Backlight 100%		252		mA
	Backlight 150%		382		mA
Brightness (100%)	with PCAP	750	820		cd/m ²

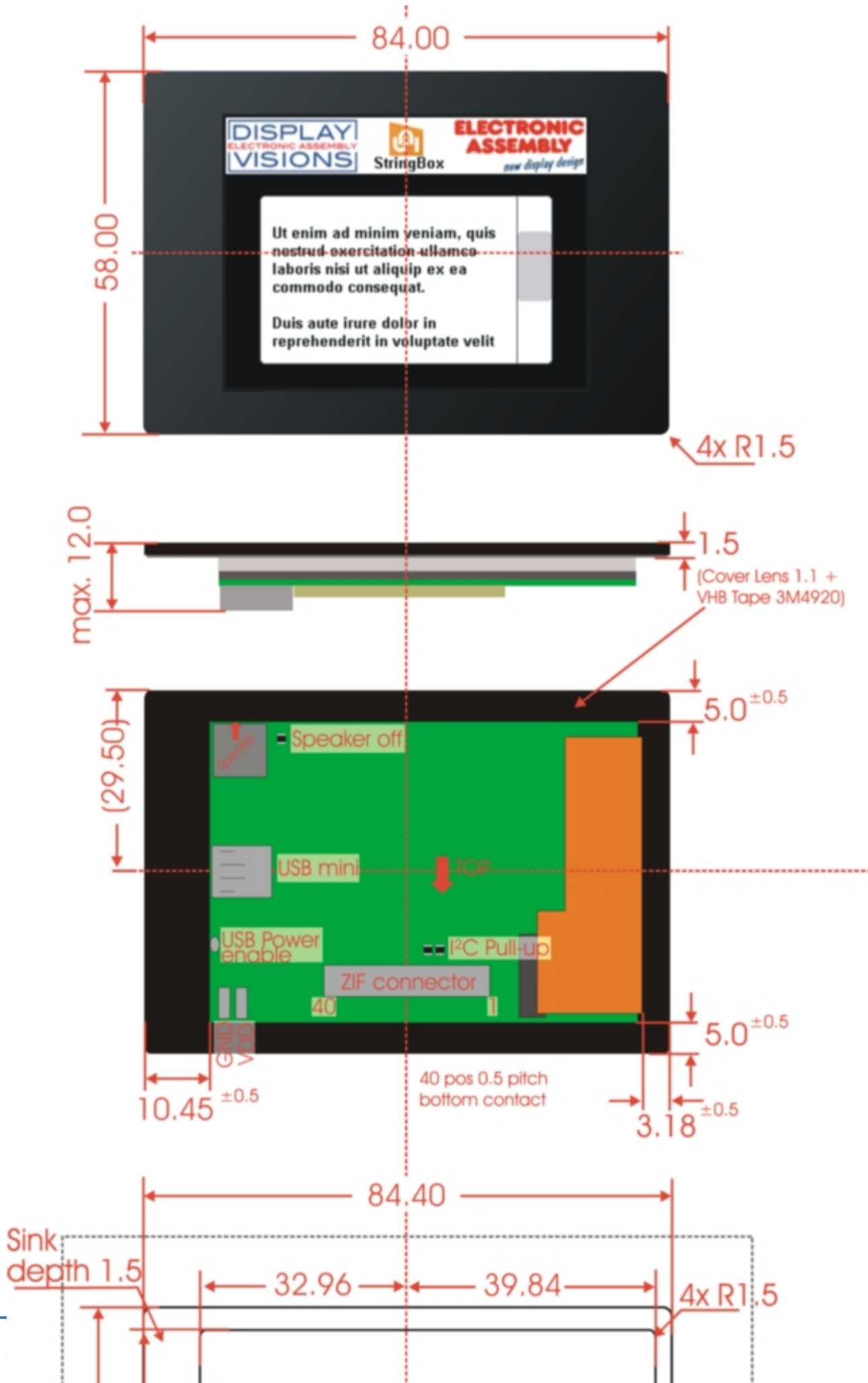
Elektrische Spezifikation Allgemein

Value	Condition	min.	typ.	max.	Unit
Operating temperature		-20		70	°C
Storage temperature		-30		80	°C
Storage humidity	@ 60°C			90	% RH
Operating voltage		3.1	3.3	3.5	V
Input low voltage (except USB, I/O)		-0,3	0	0.3*VDD	V
Input high voltage (except USB, I/O)		VDD*0.7		VDD+0.3	V
Output low voltage (except USB, I/O)		-	-	0,4	V
Output high voltage (except USB, I/O)		VDD-0.5	-	-	V
Output current I/O	single	-	-	2	mA
	all	-	-	16	mA (total)
I ² C-bus pull-up				10	k

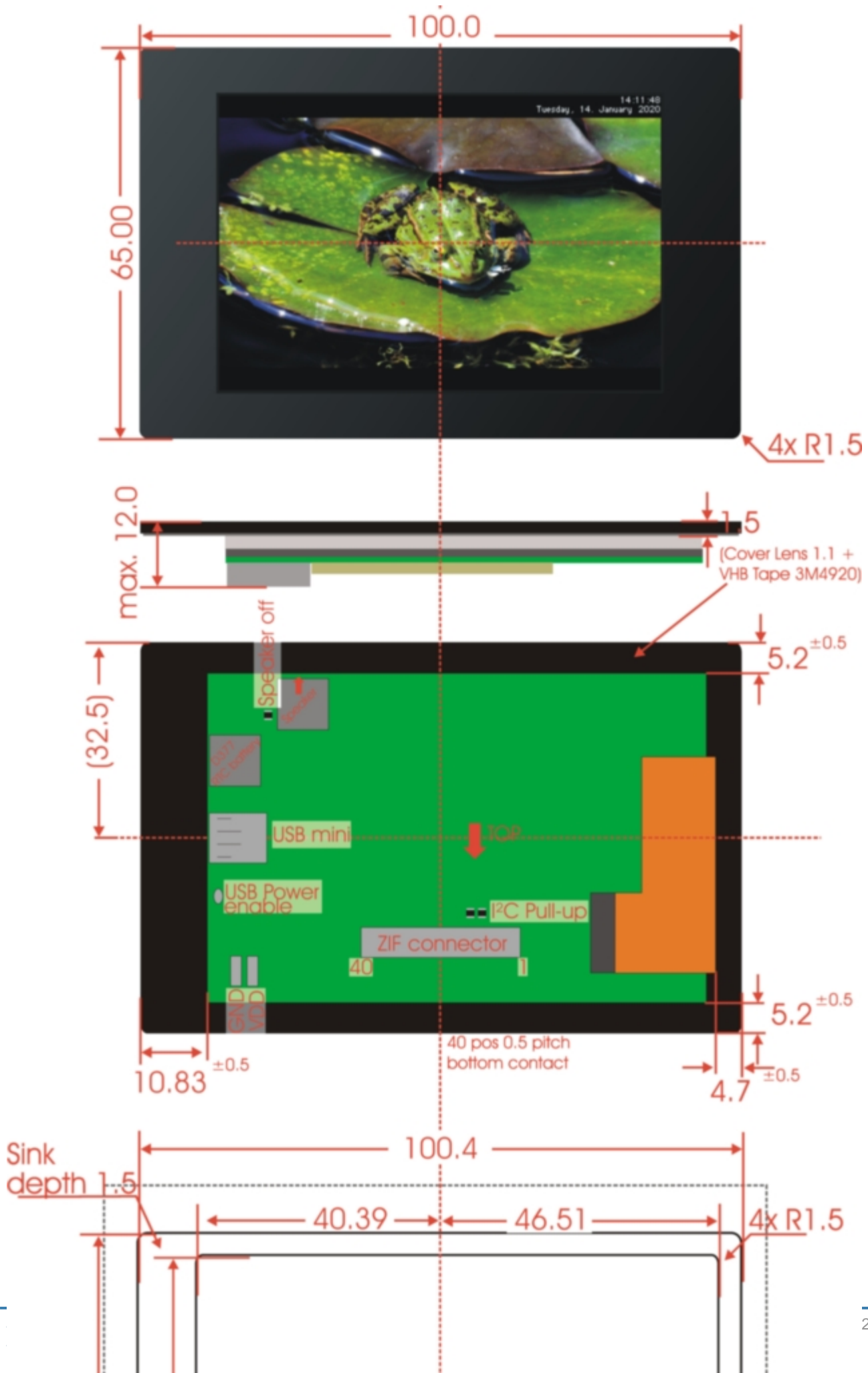
Dimension EA uniTFTs020-ATC



Dimension EA uniTFTs028-ATC



Dimension EA uniTFTs035-ATC



Dimension EA uniTFTs043-ATC

uniTFTDESIGNER - DESIGNSOFTWARE

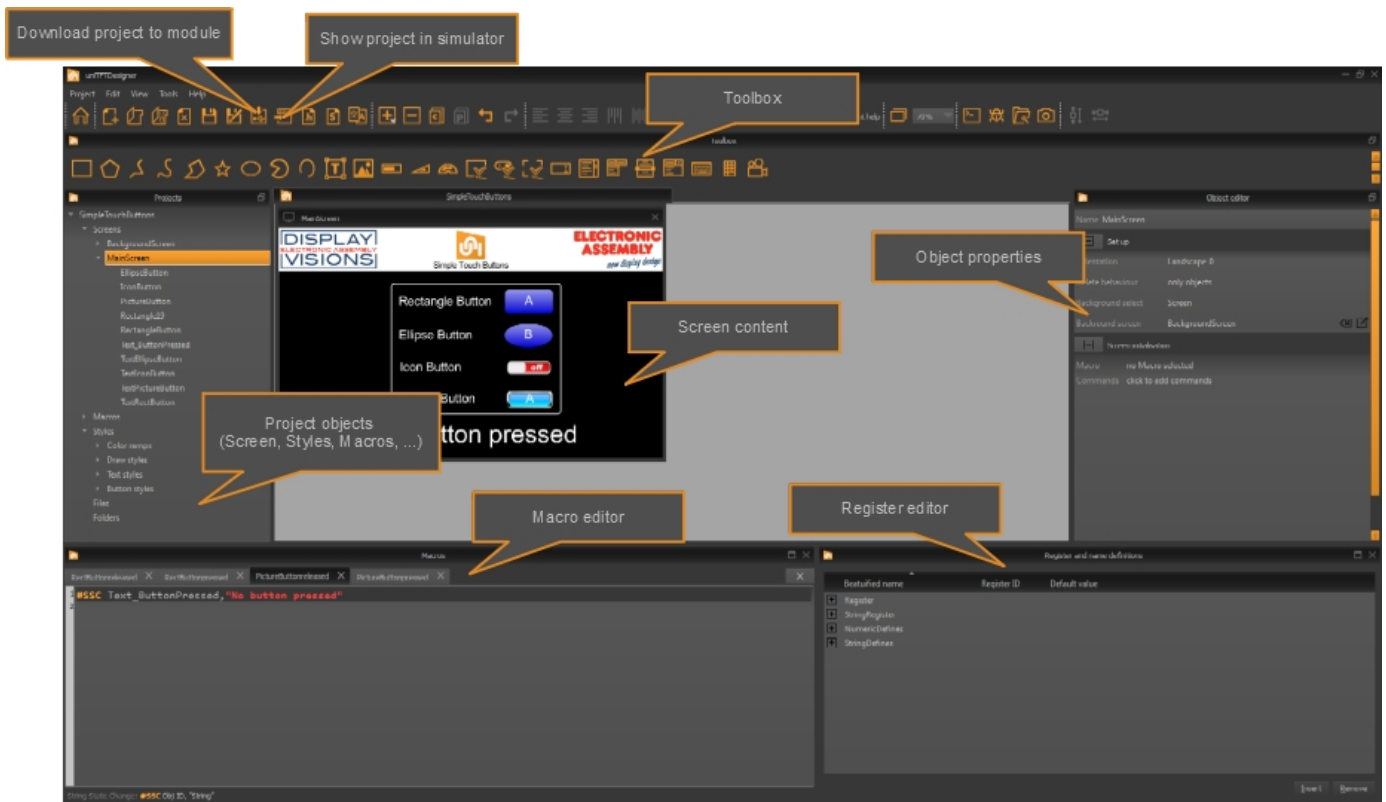
The Windows design software uniTFTDesigner (WYSIWYG) makes it easy to create complete screen layouts. With the help of the macro editor, functional sequences can be defined. The properties of all objects (position, size, angle) are easily adjustable.

The touch functionality is also supported by uniTFTDesigner, so you can create radio groups, sliders, bar-graphs and simple touch-buttons. Touching a button may switch to a new screen or start a macro. An integrated simulator shows immediately the real screen including functionality. Also the digital and analogue inputs and outputs will be simulated.

A comprehensive debug-function and the integrated help function round off this package.



The surface



Help engine and explanations

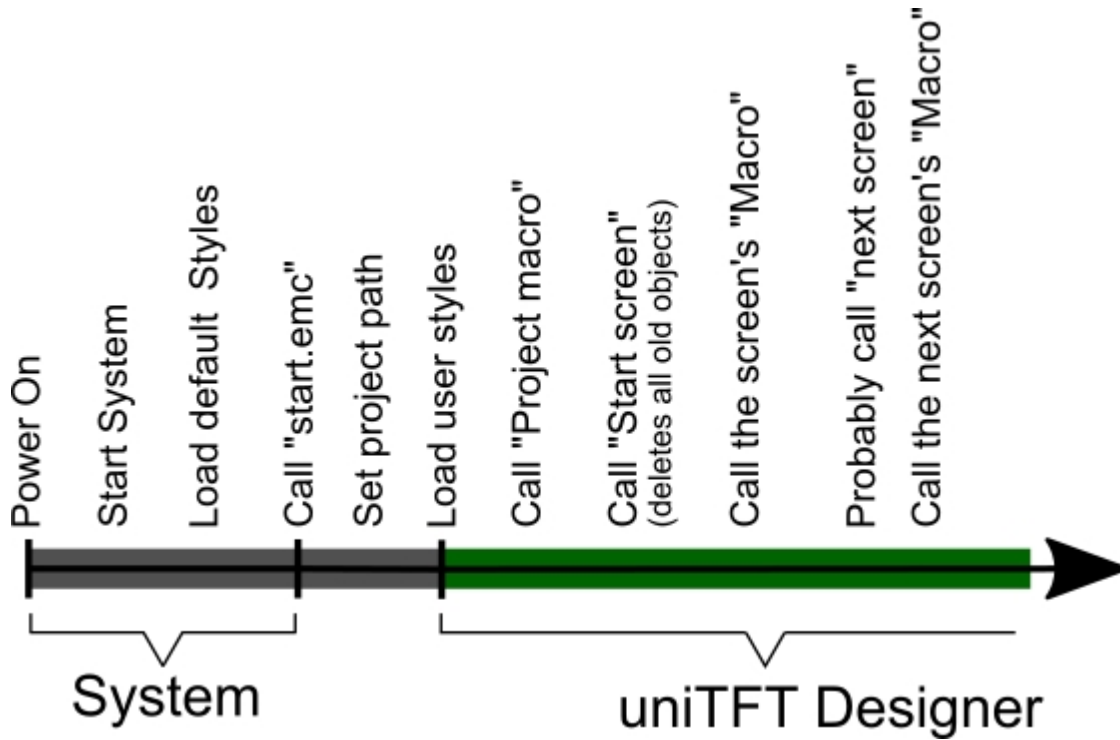
The menu item "Help" shows information about the version status (about dialogue), as well as this help file (or press "F1").

In the Macro Editor, you can access the specific help for the respective command by pressing the key combination F1.

A large selection of example projects can be found on the home/welcome-screen.

Processing order: Macros, screen

The processing of macros and screens follows the following chart:



Note: uniTFTDesigner always deletes all definitions and objects ([#ODI.0](#)) and starts a new screen afterwards - except this is disabled by "Delete behaviour" in the screen properties or limited to delete objects only.

Short cuts

Short cuts

For faster use of uniTFT Designer there are some short cuts:

Global short cuts

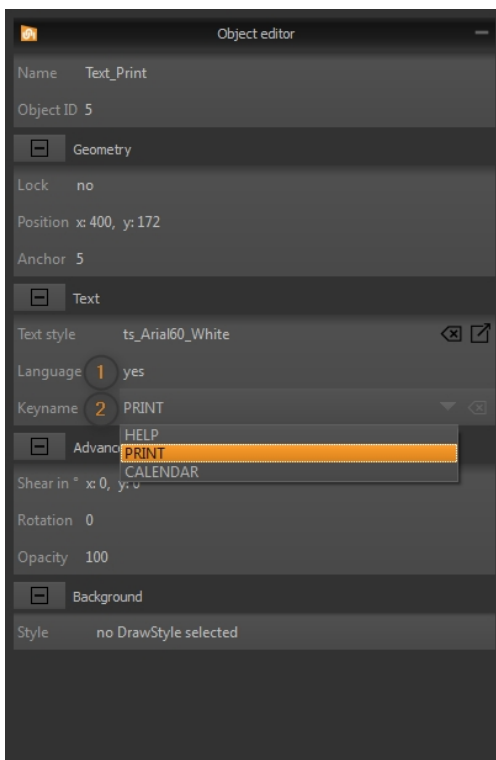
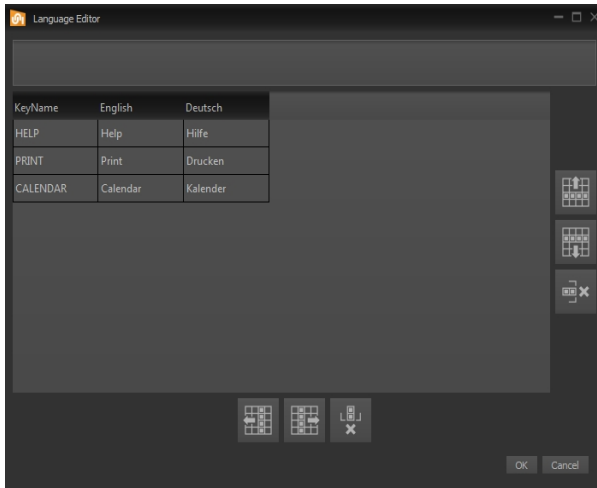
Open Help	F1
Open command help	F1
Close	Ctrl + Q / Alt + F4
New project	Ctrl + N
Open project	Ctrl + O
Close project	Ctrl + F4
Save project	Ctrl + S
Save project as	Ctrl + Shift + S
Create backup zip-file	Ctrl + Shift + Z
Load backup zip-file	Ctrl + Shift + B
Load Recovery-Point	Ctrl + Shift + R
Copy (objects, screens, styles usw.)	Ctrl + C
Paste	Ctrl + V
Select all	Ctrl + A
Zoom in (Screen)	Ctrl + +
Zoom out (Screen)	Ctrl + -
Undo	Ctrl + Z
Re-do	Ctrl + Y
Edit	F2
Deploy to hardware (EA uniTFTs-Series)	F5
Deploy to simulator	F6
Interface setting hardware / Simulator path	Ctrl + Alt + P
Open terminal / console	Alt + T
Default View (reset to default view)	Ctrl + 0
Start uniEXPLORER	Ctrl + Alt + E
Start Hardcopytool	Ctrl + Alt + H
Start Debugger	Ctrl + Alt + D

WYSIWYG - Graphic short cuts

select / deselect multiple items	Shift + left mouse button
select / deselect stacked items	Ctrl + left mouse button
select / deselect object within group	Alt + left mouse button
Layer: one up	Ctrl + arrow up
Layer: one down	Ctrl + arrow down
Layer: to top	Ctrl + Shift + arrow up
Layer: to bottom	Ctrl + Shift + arrow down
Group	Ctrl + G
Un-group	Ctrl + U
Move object by 1 pixel	Arrow keys
Move object by grid size	Shift + Arrow keys
Alignment: Item(s) Top	Ctrl + Alt + T
Alignment: Item(s) Bottom	Ctrl + Alt + B
Alignment: Item(s) Left	Ctrl + Alt + L
Alignment: Item(s) Right	Ctrl + Alt + R
Alignment: Item(s) Vertical	Ctrl + Alt + V
Alignment: Item(s) Horizontal	Ctrl + Alt + H
Alignment: Item anchors horizontal	Ctrl + Alt + 1
Alignment: Item anchors vertical	Ctrl + Alt + 2
Space: Vertical	Shift + Alt + V
Space: Horizontal	Shift + Alt + H
Space: to Grid vertical	Shift + Alt + 1
Space: to Grid horizontal	Shift + Alt + 2

Language Editor

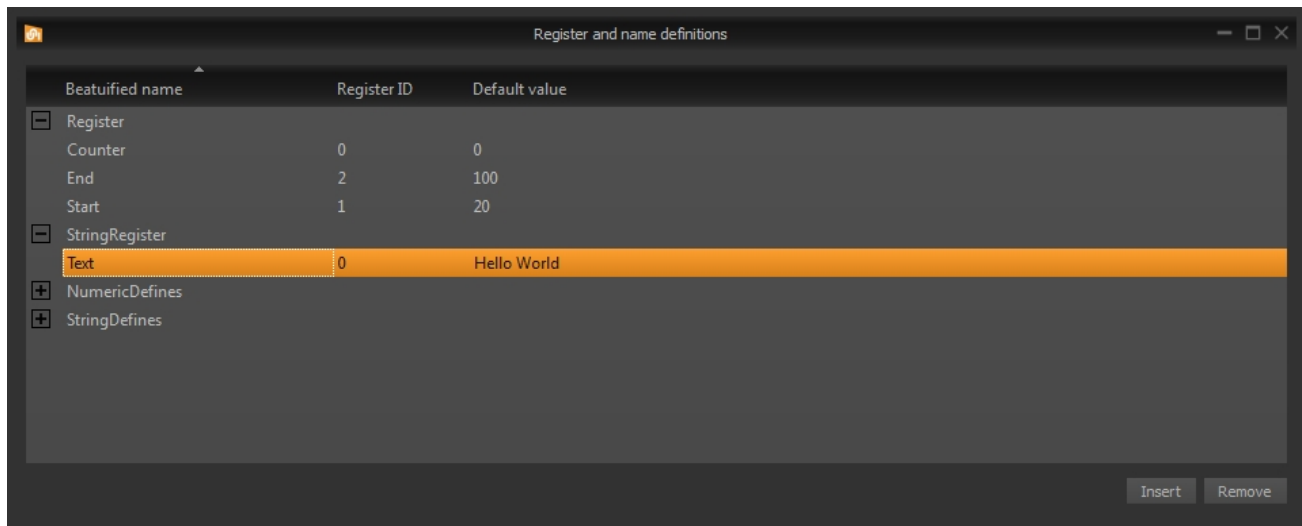
The uniTFTDesigner supports multi-language of the EA uniTFTs-Series. In the language editor (**Project -> Language Editor**) multiple languages and KeyNames can be defined together with the corresponding translation. The file (Language.csv) containing the data can be found in the data folder of the project. You can use the file to hire a translation agency.



1. To use multiple languages together with objects (e.g. Text, Button, SpinBox, ...) the property "Language" need to be activated in the object editor.
2. Now the KeyName can be selected and is translated during runtime to the set language.

Register Editor

In the Register Editor (**View -> Workspace Panels -> Register and name definitions**) beautified names can be assigned to registers and string registers. Default values can be set, too. Additionally numeric and string defines can be set. Use it like defines during compile time. Beautified names can be used in the macro editor instead of ObjectID's.



Macro Editor

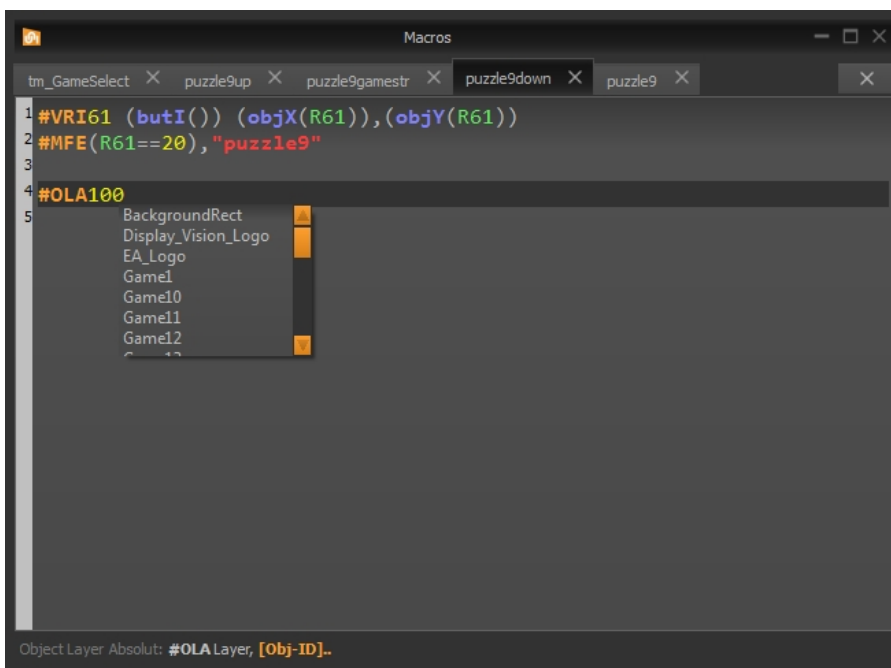
In the Macro Editor (**View -> Workspace Panels -> Macros**) command sequences are written in function groups the so called macros. It makes sense to edit and define all objects that need to be calculated and to use macros for all non-graphic commands.

An advantage is the syntax highlighting to see commands and parameters clearly structured. Also comments (starting with **/****) can be inserted.

Vorteilhaft sind das Syntaxhighlighting um Befehle und Parameter klar strukturiert zu erkennen. Auch Kommentare (beginnend mit **/****) können eingefügt werden.

All object, macro and register names available in the project and also the built-in calculations are suggested to match the parameter (**Ctrl + space**).

The short command help in the status bar is useful as a short information. With the shortcut **F1** the help for the respective command is automatically displayed.

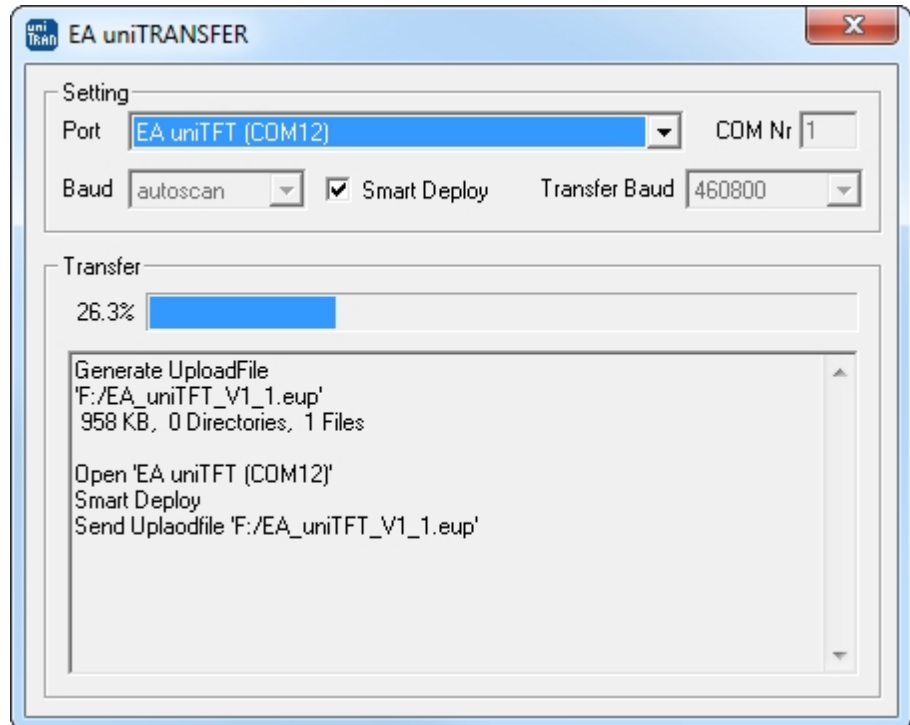


TOOLS FOR WINDOWS

Besides the design-software [uniSKETCH](#) there are a number of other Windows-Tools. Among them is the tool [EA uniTRANSFER](#) which can transfer projects to the EA uniTFTs-Series. For documentation purposes, it is often very helpful to take a screen capture to illustrate different situations. Here the tool [EA Hardcopy](#) can be of help. The most powerful tool is the [EA uniTFT simulator](#), which simulates the real hardware on the PC.

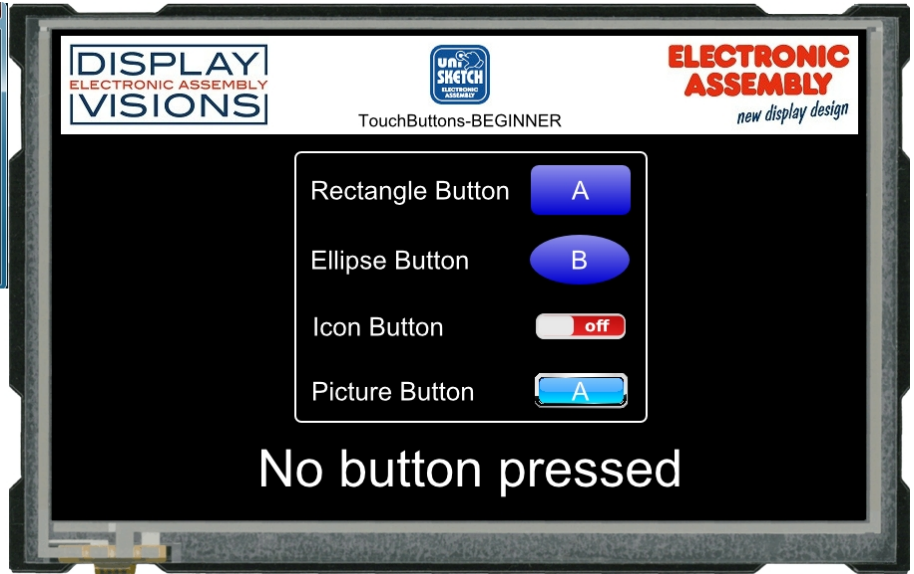
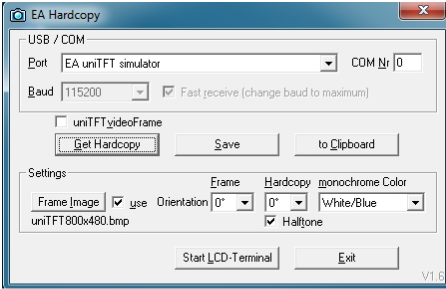
EA uniTRANSFER

After the port settings have been selected correctly, uniTRANSFER can copy any files to internal FLASH. To download projects, it's sufficient to drag the project folder to the window by drag'n'drop. A progress bar in the program provides information about the status of the transmission. On the display itself further information is visible. The checkbox "Smart Deploy" can be activated to transfer files and projects as fast as possible. It compares creation time and file size between module and data source. If these are different, the file is replaced otherwise it remains and will not be copied. This saves a lot of time on large files, such as fonts or pictures. EA uniTRANSFER creates a *.eup file. This file contains all transmission data as well as commands for programming the FLASH memory. You can also transfer the created upload file *.eup under any other system to the EA uniTFT. For this you transfer the content of the *.eup file 1:1 (with [protocol](#) in packets), no further commands are necessary.



EA Hardcopy

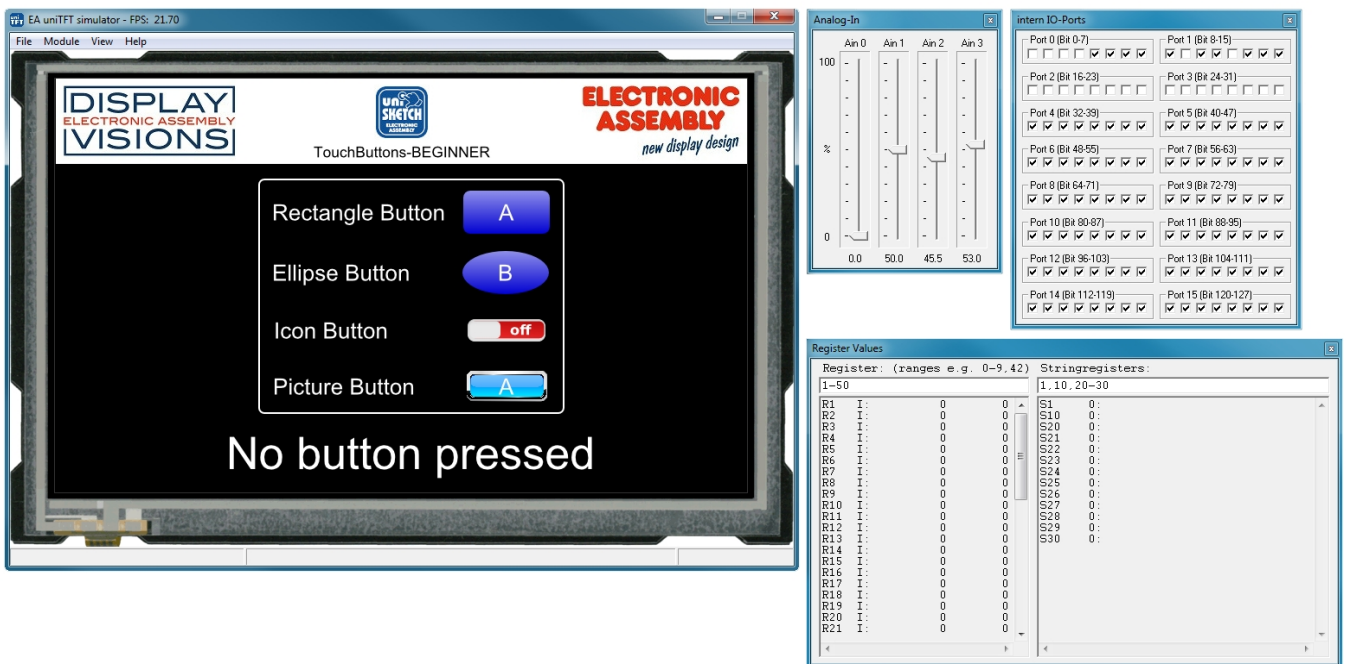
T
h
e
h
a
r
d
c
o
p
y
t
o
o
l
i
s
s
u
i
t
a
b
l
e
f
o
r
c
r
e
a
t
i
n
g
a
m
e
a
n
i
n
g
f
u
l
l
d
o
c
u
m
e
n
t



a
t
t
i
o
n
o
f
t
h
e
a
p
p
l
i
c
a
t
i
o
n
.

EA uniTFT simulator

T
h
e
s
i
m
u
l
a
t
o
r
c
a
n
b
e
c
a
l
l
e
d
i
r
e
c
t
l
y
f
r



o
m
u
n
i
T
F
T
D
e
s
i
g
n
e
r
(
F
6
)
a
n
d
s
i
m
u
l
a
t
e
s
b
e
h
a
v
i
o
r
o
f
t
h
e
p
r
o
j
e
c
t
h
a
r
d
w
a

r
e
.
l
n
a
d
d
i
t
i
o
n
t
o
i
n
p
u
t
o
p
t
i
o
n
s
s
u
c
h
a
s
p
o
r
t
s
a
n
d
a
n
a
l
o
g
i
n
p
u
t
s
,
e
.
g
.

t
h
e
c
o
m
p
u
t
e
r
,
s
o
w
n
R
S
2
3
2
i
n
t
e
r
f
a
c
e
c
a
n
b
e
u
s
e
d
a
s
M
a
s
t
e
r
R
S
2
3
2
o
r
S
I
a

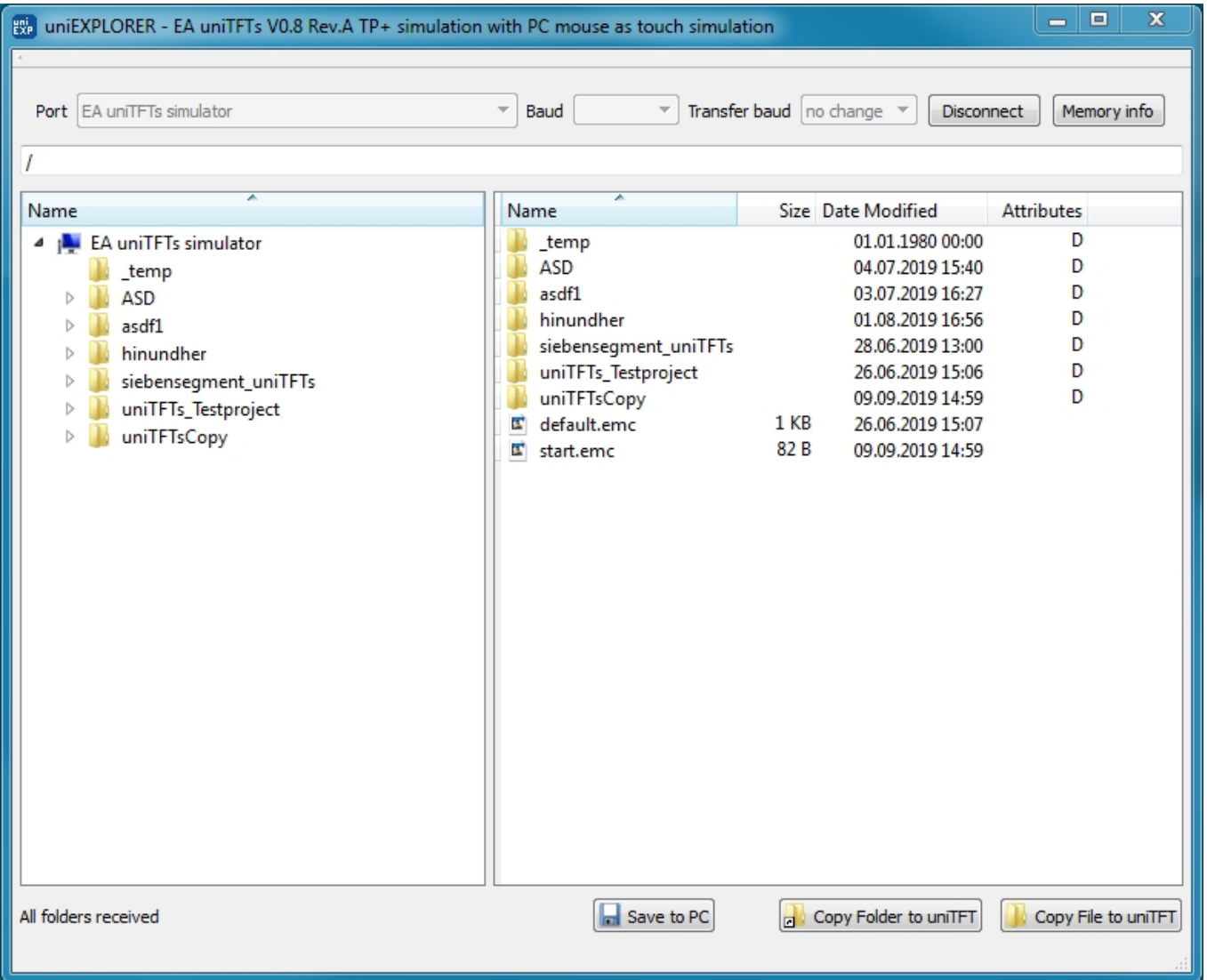
V
e
R
S
2
3
2
i
n
t
e
r
f
a
c
e
.
A
d
e
b
u
g
f
u
n
c
t
i
o
n
a
n
d
o
n
l
i
n
e
d
i
s
p
l
a
y
o
f
t
h
e
r
e
g
i
s
t

e
r
s
f
a
c
i
l
i
t
a
t
e
s
t
h
e
d
e
v
e
l
o
p
m
e
n
t
o
f
m
a
c
r
o
f
i
l
e
s
. Y
o
u
c
a
n
a
l
s
o
s
e
t
b
r
e
a

k
p
o
i
n
t
s
a
n
d
s
t
e
p
t
h
r
o
u
g
h
s
i
n
g
l
e
l
i
n
e
s
.

EA uniEXPLORER

T
h
e
u
n
i
E
X
P
L
O
R
E
R
i
s
a
c
c
o
m
f
o
r
t
a
b
l
e
t
o
o
l
t
o
s
e
e
a
l
l
f
i
l
e
s
o
n
t
h
e
m
o
d
u
l



e

REVISION HISTORY

EA uniTFTs-Series Firmware

Date	Version	Info
	1.0	First release

uniTFTs-Simulator

Date	Version	Info
	1.0	First release

uniTFTDesigner - Windowstool

Date	Version	Info
	1.0	First release

Helpfile

Date	Version	Info
	1.0	First release