EA uniTFTs-Serie

EIGENSCHAFTEN

- 2.0" 320x240 / 2.8" 320x240 / 3.5" 480x320 / 4.3" 480x272
- AACS-Display (all angle color stability, optimierte Beleuchtung und TFT-Panel über den gesamten Blickwinkel)
- Optisch gebondeter kapazitiver Touch (PCAP), superhell mit mehr als 800 cd/m²
- Objektorientierter Bildschirmaufbau
- Objekte zur Laufzeit verändern: Größe, Form, Farbe, Inhalt
- Objekte animieren, bewegen, Alpha-blending
- Zeichensätze, ASCII und Unicode, Chinesisch
- Single supply 3,3 V oder direkt per USB
- Serielle Interfaces: USB, RS-232, SPI, I²C
- 8 digitale frei definierbare I/O eingebaut, auf 136 erweiterbar, 4 analoge Eingänge
- Uhrzeit, Batteriegepuffert (3,5" und 4,3")
- Flash-Speicher als Datenspeicher für Bilder, Fonts, Menüs und Log-Files
- interne Rechenfunktionen und Programmierbarkeit

BESTELLBEZEICHNUNG

2.0" TFT 320x240 Pixel, PCAP, weiße LED-Beleuchtung (Außenmaß: 65x43 mm)	**EA uniTFTs020-ATC**
2.8" TFT 320x240 Pixel, PCAP, weiße LED-Beleuchtung (Außenmaß: 84x58 mm)	**EA uniTFTs028-ATC**
3.5" TFT 480x320 Pixel, PCAP, weiße LED-Beleuchtung (Außenmaß: 100x65 mm)	**EA uniTFTs035-ATC**
4.3" TFT 480x272 Pixel, PCAP, weiße LED-Beleuchtung (Außenmaß: 114x84 mm)	**EA uniTFTs043-ATC**

ZUBEHÖR

Starterset mit 2,8" IPS Display, PCAP, Testboard mit RGB-LED und analogem Eingang	**EA DEMOPACK-RGBANA**
ZIF-Stecker 40 polig 0,5 mm pitch Gegenstecker für FPC-Kabel	**EA WF050-40S**
FPC-Kabel 40 polig 0,5 mm pitch 102 mm lang	**EA KF050-40**
USB-Kabel Typ A -> Mini USB ca. 1 m	**EA KUSB-MINI**

Inhaltsverzeichnis

ALLGEMEINES

Die EA uniTFTs-Serie ermöglicht mit dem integrierten Befehlssatz eine ausgefeilte grafische Darstellung und intuitive Menüsteuerung. Dank dem integrierten Befehlssatz und der Windowsdesignsoftware uniTFTDesigner können nicht nur Elektronikspezialisten sondern z.B. auch Experten aus dem Bereich Design und Benutzerführung das gesamte HMI erstellen.

Die Displaymodule sind mit 3,3 V sofort betriebsbereit, die Ansteuerung erfolgt über die eingebauten seriellen Schnittstellen RS232, SPI, I²C oder USB. Die Module können auch direkt über USB versorgt werden.
Durch die objektorientierte "Programmierung", den umfangreichen Befehlssatz und den bereits integrierten, und jederzeit erweiterbaren Unicode-Schriftsätzen wird "Time-to-Marked" ein Kinderspiel.

Mit einem sehr ähnlichen Befehlssatz wartet die EA uniTFT-Serie auf, welche den High-End Markt mit größeren Modulen bildet:
Aktuell sind 3 verschiedene Größen lieferbar: 5" mit 800x480 Punkten Auflösung und sehr hellen 900 cd/m², sowie 2 verschiedene IPS-Displays mit 7" (1024x600) und 10,1" (1280x800) und jeweils extrem hellen 1000cd/m².

SOFTWARE - ARBEITSWEISE DER EA UNITFTS-SERIE

Die Darstellung auf dem Display erfolgt anhand einer Vielzahl von Befehlen. Die Befehle können entweder per Laufzeit über eine der seriellen Schnittstellen übertragen werden oder auf dem internen Speicher in sogenannten Macros zusammengefasst und fest abgelegt werden. Mithilfe der Befehle werden Grafikobjekte erstellt. Diese Objekte haben unterschiedliche Eigenschaften, wie z.B. Farbe, Form und eingebaute Aktionen. Diese Eigenschaften können jederzeit manipuliert werden, so kann z.B. ein String geändert werden oder aber die Postion eines Touchtasters geändert werden.
Alle nur erdenklichen Objekte können beliebig platziert, bewegt und wieder gelöscht oder auch nur unsichtbar werden. Windows-Zeichensätze werden direkt im Display abgelegt. Dank automatischer ASCII- und Unicode- Umschaltung werden flexibel die unterschiedlichsten Systeme unterstützt; Chinesische Zeichen eingeschlossen. Elegante Effekte zum Ein- und Ausblenden oder Hereinfliegen sind bereits integriert. Über Stylesheets lassen sich durchgängig einheitliche Designs erstellen. Bilder können als JPEG, PNG und viele andere (auch transparent) eingebunden werden. Zusammen mit der integrierten (EA uniTFTs035-ATC and EA uniTFTs043-ATC), batteriegepufferten Zeitbasis lassen sich Ereignisse zusammen mit einem Zeitstempel dokumentieren oder auch Abläufe völlig autark ohne externem Rechner steuern.

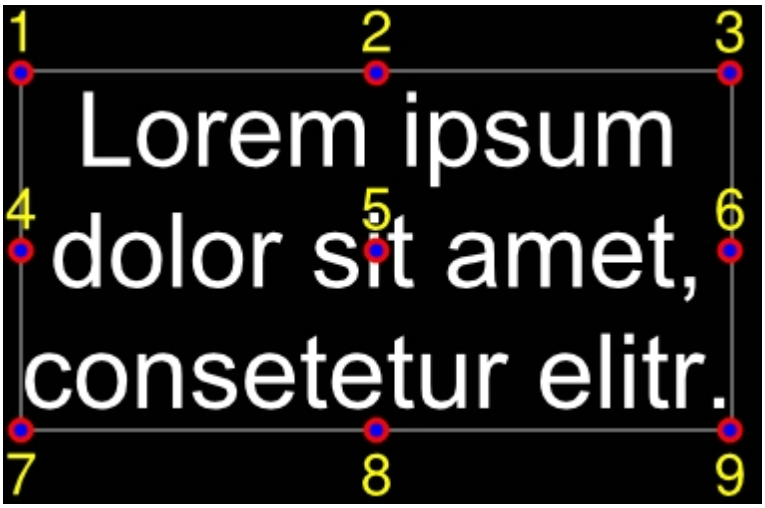
Objekte

Jedes Bild, jeder Text und jeder Button ist ein sogenanntes Objekt. Jedes Objekt muss mit einer Objekt-ID versehen werden, welche es eindeutig identifizierbar macht. Vergibt man eine Objekt-ID, dann wird bei erneuter Vergabe dieser Objekt-ID das vorherige Objekt überschrieben. Über die Objekt-ID können die Eigenschaften eines Objekts jederzeit geändert werden. Es können einfache Grafikobjekte mit der ID 0 direkt auf die Hintergrundebene gerendert werden, sind dann aber nicht mehr veränderbar.
Die Befehle zur Objektverwaltung sind hier zu finden.

Objektposition / Anker

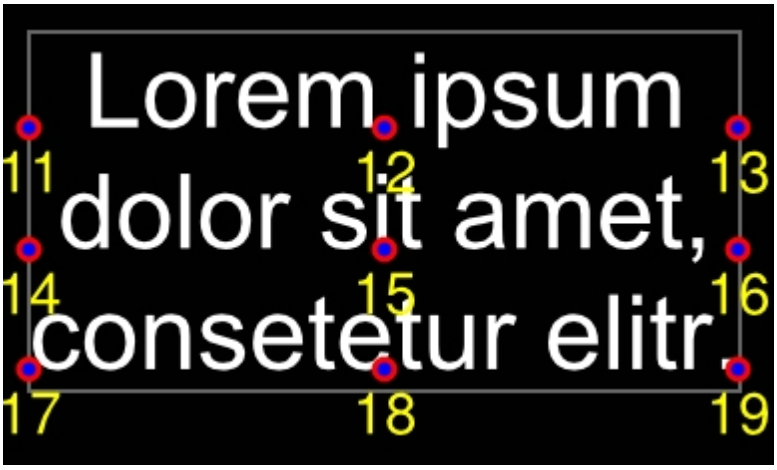
Allgemeine Anker

Die Position eines Objekts ergibt sich durch die Koordinaten (Ursprung links unten) bezogen auf den Objektanker. Jedes Objekt hat 9 unveränderbare Anker.



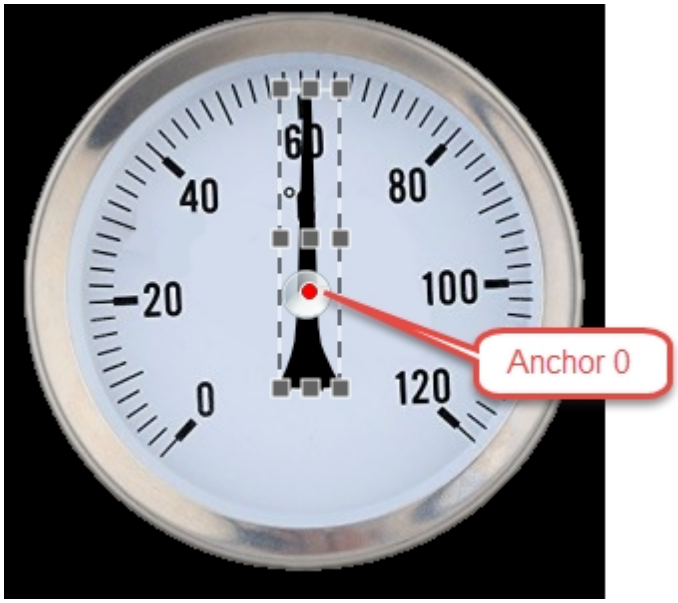
Zeichenketten und Anker

Zeichenketten haben weitere 9 Anker um Objekte (z.B. Gerade für Unterstreichen) an der Textbaseline ausrichten zu können.



Spezialfall: Anker 0

Jedes Objekt besitzt zusätzlich einen frei definierbaren Anker. Der Anker 0 kann frei positioniert werden. Bei Kreisen, Ellipsen, Sternen ist der Objektanker 0 gleichzeitig der Konstruktionspunkt.



Der Zeiger soll sich um den Mittelpunkt drehen. Die in dunkelgrau dargestellten 9 Standardanker sind nicht hilfreich in diesem Fall, denn eine Rotation erfolgt immer um einen Anker. Der Anker 0 kann pixelgenau an die gewünschte Position platziert werden (#OAS).

Styles / StyleSheets

Styles ermöglichen ein durchgängiges Design über das komplette Projekt in Bezug auf Farben, Schriften und Buttons. Es gibt
- DrawStyles
- TextStyles
- ButtonStyles

Um ein Objekt oder Text zu platzieren wird ein DrawStyle oder TextStyle benötigt. Der DrawStyle definiert die Strichstärke und die Füllfarbe, der TextStyle den Zeichensatz und die Schriftgröße.

DrawStyle:
Farben, Farbverläufe, Muster und Linien werden im DrawStyle zusammengefasst.
TextStyle:
Das Aussehen einer Zeichenkette wird im TextStyle definiert. Ein TextStyle basiert auf einem DrawStyle für die Farbe und weiteren Definitionen zur Schriftart, Größe und Zeilenabstand.
ButtonStyle:
Touchbuttons und -schalter erhalten ihr Aussehen von einem ButtonStyle, der wiederum aus TextStyles für die Beschriftung und DrawStyles für den Hintergrund des Buttons besteht.

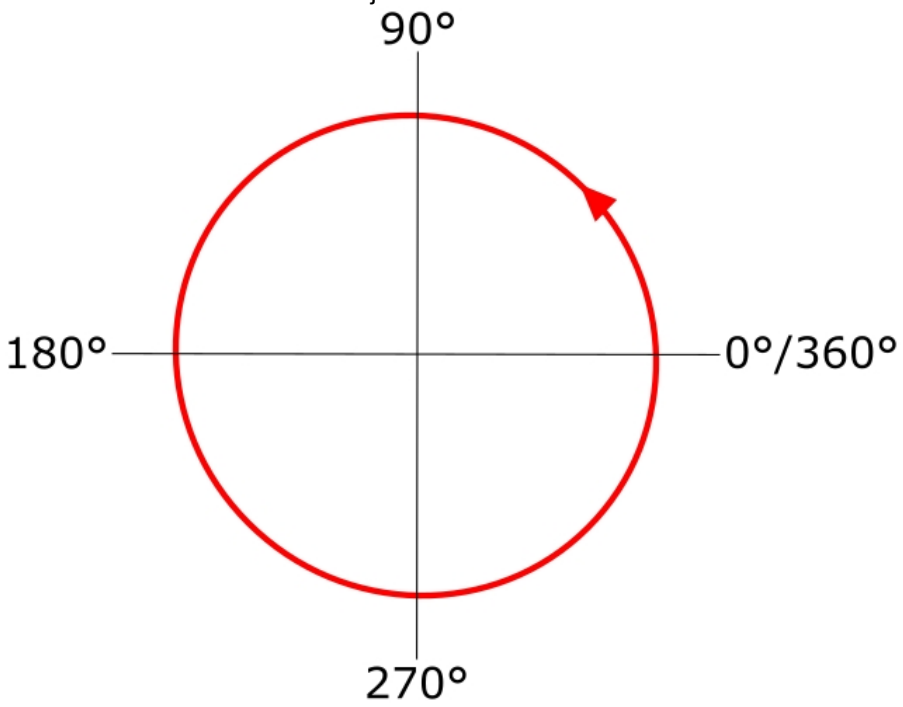
ColorRamp:
Ein Objekt kann mit einer einzelnen Farbe oder einem Farbverlauf gefüllt werden. Dieser Farbverlauf und seine Farben wird in den ColorRamps definiert und kann dann linear oder radial verwendet werden.

Das Windowstool uniTFTDesigner unterstützt StyleSheets welche eine Sammlung von mehreren Draw-, Text- und ButtonStyles sowie ColorRamps enthält.

Die Befehle zu den einzelenen Styles und ColorRamps sind hier zu finden.

Koordinatensystem und Winkel

Das Koordinatensystem bezieht sich direkt auf die Displayauflösung des Moduls, beispielsweise beim EA uniTFTs028-A einen Bereich von 320 x 240. Der Ursprung 0|0 liegt in der linken unteren Ecke. Damit ergibt sich für die Koordinaten ein Umfang von 0..319 bzw. 0..239.
Winkel werden im mathematischen Drehsinn (gegen den Uhrzeigersinn) angegeben. 0° befindet sich horizontal rechts. Außer bei Instrumenten sind Objekte in 90° Schritten drehbar:

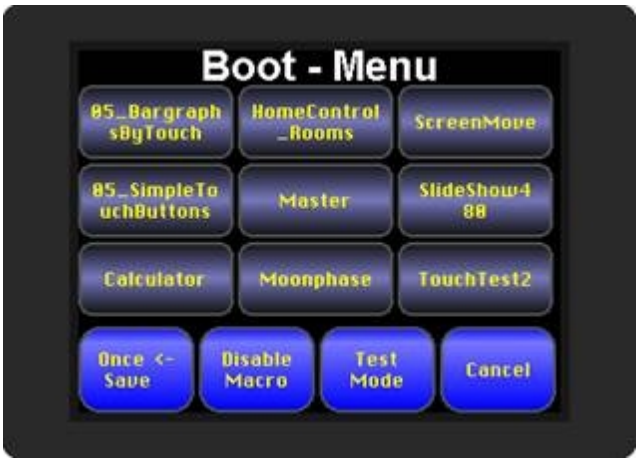


Mehrsprachigkeit - Stringfiles

In einer immer mehr zusammenwachsenden Welt mit internationalem Equipment ist die Unterstützung verschiedener Sprachen ein Muss. Die EA uniTFTs-Serie bietet mit der Unicode-Unterstützung die Basis um alle Schriften und Zeichen einschließlich Chinesisch darzustellen.
Weiterhin unterstützen sog. Stringfiles eine Umschaltung der Sprache während des Betriebs. Stringfiles sind Textfiles als eine Art Datenbank der anzuzeigenden Strings. In den Makrofiles wird für Strings auf ein Index referenziert. In den Stringfiles ist dieser Index zu finden und wird dann automatisch mit dem entsprechenden Eintrag / Text in der richtigen Sprache dargestellt.
Genauere Beschreibung sind bei dem Befehl #VFL oder bei den Beispielen zu finden.

Bootmenü

Auf dem integrierten Speicher können mehrere Projekte gespeichert werden. Welches Projekt automatisch gestartet wird legt das File "start.emc" fest. Um ein anderes Projekt zu laden kann entweder die Datei verändert bzw. ersetzt werden. Alternativ gelangen Sie über das Touchpanel in das Bootmenü:
Beim Einschalten oder sofort nach einem Reset streichen Sie in kurzen Abständen mehrmals über das Touchpanel.



Um Missbrauch durch den Bediener zu vermeiden, kann das Bootmenü deaktiviert werden. Hierfür muss auf dem Speicher im Root-Verzeichnis die Datei "bootmenu.off" mit leeren Inhalt erzeugt werden. Das kann auch durch uniTFTs Befehle erfolgen:
#FWO</bootmenu.off>
#FWC

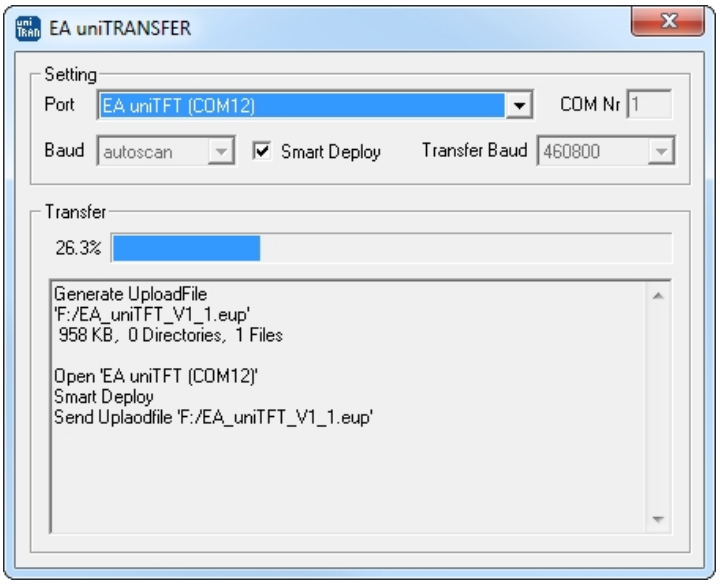
Neben der Projektauswahl kann im Bootmenü ein **Testmode** gestartet werden, um Informationen über das Modul angezeigt zu bekommen: Version, Protokollstatus, Baudrate, SPI Mode, I²C Bus Adresse etc.

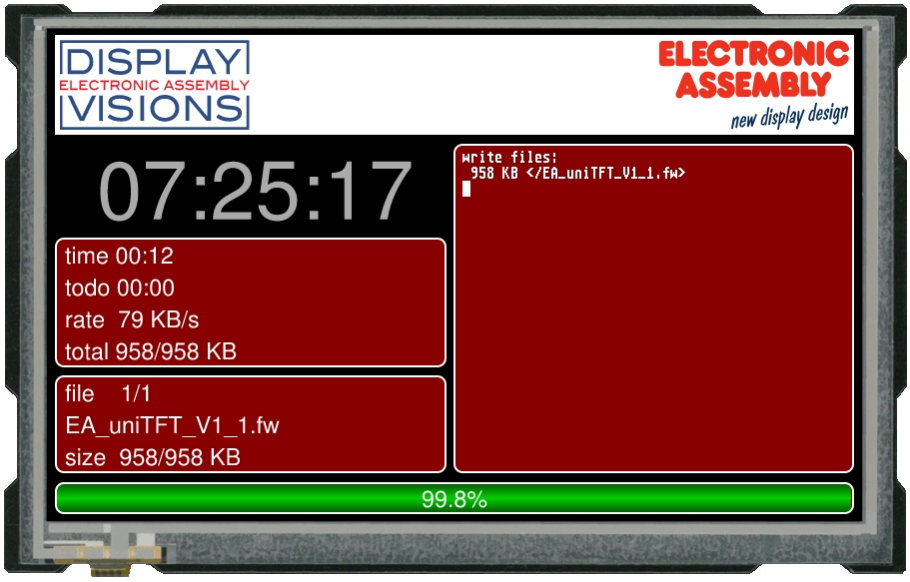
Firmwareupdate

Um die neuesten Features der EA uniTFTs-Serie nutzen zu können kann es notwendig sein, die interne Firmware des Moduls zu erneuern. Dies kann auf zwei Varianten erfolgen: entweder über die seriellen Schnittstellen oder mit Hilfe der SD-Card.

Firmwareupdate per serielle Schnittstelle und Windows PC:

- speichern Sie die Firmwaredatei (z.B. EA_uniTFTs_V1_1.fw) auf der Festplatte
- verbinden Sie das EA uniTFT mit Ihrem PC
- starten Sie uniTRANSFER.exe (zu finden in den Tools der uniTFTDesugner-Installation) und wählen die korrekte Schnittstelle als Verbindung zum EA uniTFTs aus
- schieben Sie die neue Firmwaredatei aus dem Windows-Explorer per Drag&Drop auf das Fenster "uniTRANSFER.exe"





- Nach der Übertragung muss manuell ein Reset ausgeführt werden, beim erneuten Start wird die Firmware geladen. **Bitte schalten Sie das Gerät während des Updates nicht aus**.
- Das Firmwarefile wird nach der Installation gelöscht, sofern es nicht das Dateiattribut "Schreibgeschützt" hat.

Firmwareupdate per serieller Schnittstelle:
Die Firmwaredatei kann mit einem beliebigen System zum EA uniTFT übertragen werden. Dazu übertragen Sie den

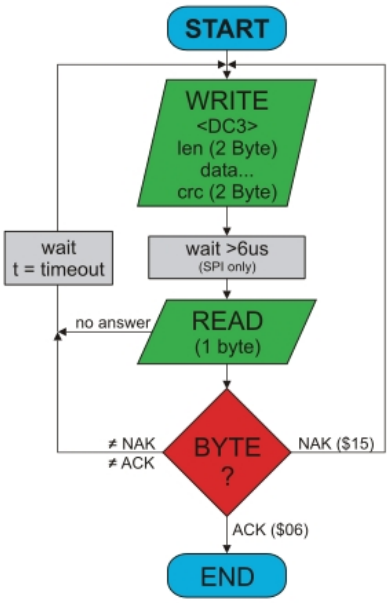
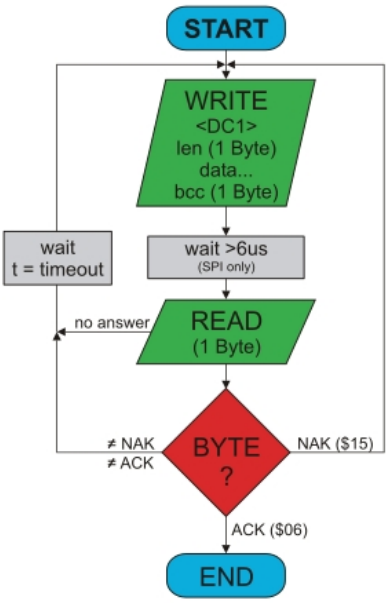
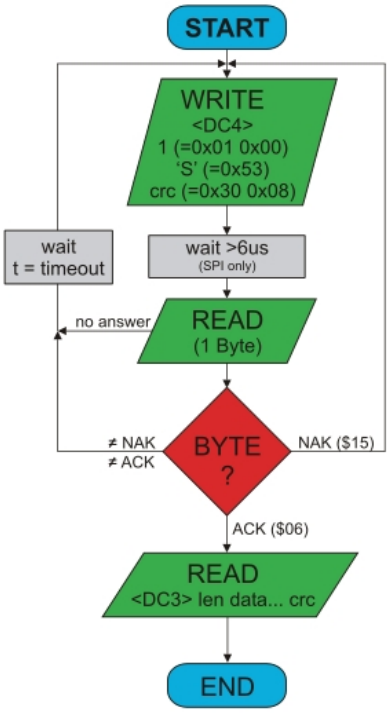
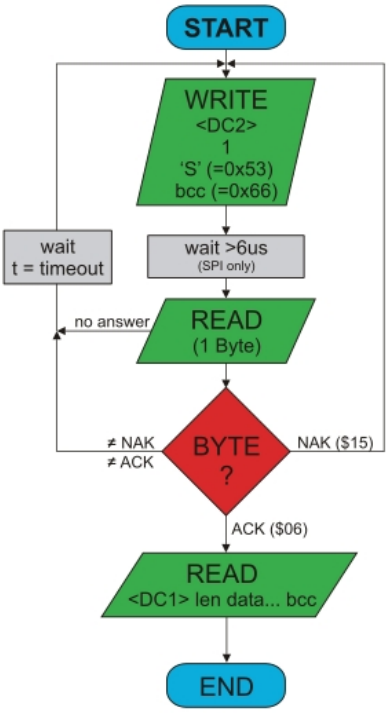
Inhalt der *.fw Datei 1:1 (mit Protokoll in einzelnen Paketen) zum EA uniTFT. Auf dem Modul kann mitverfolgt werden wieweit die Übertragung abgeschlossen ist. Nach erfolgreicher Übertragung erfolgt automatisch ein Datencheck. Bei korrekten Daten wird der Programmiervorgang automatisch gestartet. **Bitte schalten Sie das Gerät während des Updates nicht aus**.

DATENÜBERTRAGUNG / PROTOKOLL

Egal über welche der 4 seriellen Schnittstellen die Daten von der übergeordneten Steuerung übertragen werden ist das Übertragungsprotokoll identisch. Die Hardwarebeschaltung ist unterschiedlich und kann unter dem Punkt "Serielle Interfaces" nachgelesen werden.
Die Datenübertragung ist jeweils eingebettet in einen festen Rahmen mit Prüfsumme. Die EA uniTFTs-Serie quittiert dieses Paket mit dem Zeichen <ACK> (=0x06) bei erfolgreichem Empfang oder <NAK> (=0x15) bei fehlerhafter Prüfsumme oder Empfangspufferüberlauf. In Falle eines <NAK> wird das komplette Paket verworfen und muss nochmal gesendet werden. Ein <ACK> bestätigt lediglich die korrekte Übertragung. Ein Syntax-Check erfolgt nicht.
Es sind zwei unterschiedliche Protokolle implementiert, das "Short Protokoll" und das "Small Protokoll". Das Short Protokoll arbeitet mit einer CRC16 Prüfsumme und erlaubt deutlich größere Datenpakete. Während das Small Protokoll hauptsächlich aus Kompatibilitätsgründen zur Serie EA eDIPxxx implementiert wurde.
Die Anzahl der Nutzdaten pro Paket kann max. 2042 Byte bzw. 255 Byte betragen. Befehle die größer sind (z.B. Bilder oder File schreiben #FWD ...) müssen auf mehrere Pakete aufgeteilt werden. Die Nutzdaten in den einzelnen Paketen werden nach korrektem Empfang vom Displaymodul wieder zusammengefügt.

Hinweis:
Das <ACK> muss aktiv eingelesen werden (SPI und I²C). Empfängt der Hostrechner keine Quittierung (weder <ACK> noch <NAK>), so ist mindestens ein Byte verloren gegangen. In diesem Fall muss die eingestellte Timeoutzeit abgewartet werden, bevor das Paket komplett wiederholt wird.

Das Protokoll kann für erste Tests an der seriellen Schnittstelle deaktiviert werden. Hierfür ist der Pin 14 auf low zu legen (siehe Pinbelegung).

	Short protocol	Small protocol
Send		
Receive		

Short Protokoll Befehle

1. Befehle / Daten zum Modul senden

Dieser Protokoll-Befehl überträgt Daten zum Display. Es können mehrere Grafikbefehle in ein Protokollpaket verpackt werden. Sind die Daten größer als die maximale Paketgröße können die Daten auf mehrere Pakete aufgeteilt werden. Das Modul fügt die einzelnen Datenpakete wieder zusammen.
Das Modul arbeitet mit little-endian (Intel-Format), das niederwertige Byte wird zuerst übertragen.

Modul empfängt	DC3 0x13	Länge (16 Bit) 0xXX 0xXX	Data..... 0x....	crc (16 Bit) 0xXX 0xXX
Modul sendet	ACK 0x06			

Beispiel: #XCB20 ändert z.B. die Helligkeit auf 20%. Dazu gehört am Ende ein [LF] bzw. 0x0A als Abschlusskennung
Das Paket beginnt also mit DC3 und einer Längenangabe der Nutzdaten. Am Ende steht eine Prüfsumme (CRC16/CCITT) über das komplette Paket. Hier finden Sie einen Online-CRC-Calculator.
In Hex: **13 07 00 23 58 43 42 32 30 0A 3D CD** (hier als Datei zum Download; diese kann z.B. per Drag-and-Drop einfach auf terminal.exe geschoben werden)

Beispiel: #XCB80 ändert z.B. die Helligkeit auf 80%.
In Hex: **13 07 00 23 58 43 42 38 30 0A FC 0A** (hier als Datei zum Download)

2. Inhalt des Sendepuffers anfordern

Fallen Daten im Modul an, werden diese im Sendepuffer des Moduls gespeichert. Über die seriellen Schnittstellen können die Daten angefordert werden. Ob Daten verfügbar sind kann über den Pin 20 SBUF herausgefunden werden, oder aber die übergeordnete Steuerung fragt zyklisch die Daten ab (polling).

Modul empfängt	DC4 0x14	Länge (16 Bit) 0x01 0x00	'S' 0x53	crc (16 Bit) 0x30 0x08
Modul sendet	ACK 0x06			
Modul sendet	DC3 0x13	Länge (16 Bit) 0xXX 0xXX	Data..... 0x....	crc (16 Bit) 0xXX 0xXX

3. Letztes Datenpaket wiederholen

Ist ein Empfangenes Packet des Moduls fehlerhaft (falsche Länge oder Prüfsumme) kann es erneut angefordert werden:

Modul empfängt	DC4 0x14	Länge (16 Bit) 0x01 0x00	'R' 0x52	crc (16 Bit) 0x11 0x18
Modul sendet	ACK 0x06			
Modul sendet	DC3 0x13	Länge (16 Bit) 0xXX 0xXX	Data..... 0x....	crc (16 Bit) 0xXX 0xXX

4. Pufferinformationen anfordern

Mit diesem Befehl wird nachgefragt, ob Nutzdaten zur Abholung bereit stehen (= Pin20 SBUF) und wie voll der

Empfangspuffer des Displays bereits ist.

Modul empfängt	DC4 0x14	Länge (16 Bit) 0x01 0x00	'I' 0x49	crc 0x
Modul sendet	ACK 0x06			
Modul sendet	DC4 0x14	Länge (16 Bit) 0x04 0x00	Sendepuffer Bytes ready (16 Bit) 0xXX 0xXX	Empfangspuffer Bytes free (16 Bit) 0xXX 0xXX	crc 0x

5. Protokolleinstellungen

Hierüber lässt sich die maximale Paketgröße welche das Display senden darf begrenzen. Voreingestellt ist eine Paketgröße mit bis zu 2042 Byte Nutzdaten. Weiterhin lässt sich der Time-out in 1/1000s einstellen. Der Time-out spricht an, wenn einzelne Bytes verloren gegangen sind. Danach muss das gesamte Paket nochmals übertragen werden.

Modul empfängt Defaultwerte	DC4 0x14	Länge (16 Bit) 0x05 0x00	'D' 0x44	Paketgröße Sendepuffer (16 Bit) 0xFA 0x07 (=2042 Byte)	Time-out (16 Bit) in ms 0xD0 0x07 (=2 Sekunden)	crc (16 0x98 0
Modul sendet	ACK 0x06					

6. Protokollinformationen

Abfrage der Protokolleinstellungen (siehe 5.).

Modul empfängt	DC4 0x14	Länge (16 Bit) 0x01 0x00	'P' 0x50		
Modul sendet	ACK 0x06				
Modul sendet	DC4 0x14	Länge (16 Bit) 0x06 0x00	Maximale Paketgröße Sendepuffer (16 Bit) 0xFA 0x07 (=2042 Byte)	Paketgröße Sendepuffer (16 Bit) 0xXX 0xXX	Tim 0x

7. RS485 Adresse selektieren / deselektieren

Mit diesem Befehl lässt sich ein Modul am RS485-Bus selektieren oder deselektieren. Per Default ist ein Modul mit der Adresse 7 immer aktiv.

Modul empfängt Defaultwerte	DC4 0x14	Länge (16 Bit) 0x03 0x00	'A' 0x41	'S' (=Selektieren) 'D' (=Deselektieren) 0x53 oder 0x44	RS485-Adresse 0xXX	crc (16 0xXX 0
Modul sendet	ACK 0x06 ----	→ Selektieren → Deselektieren				

8. RS485 Verzögerung Enable Signal

Einige RS485 benötigen eine gewisse Zeit um das Enable-Signal zu verändern und z.B. vom schreibenden in den lesenden Modus umzuschalten. Um eine erfolgreiche Kommunikation mit diesen Geräten zu ermöglichen kann mit diesem Befehl die Umschaltung in den schreibenden Modus verzögert werden.

Modul empfängt Defaultwerte	DC4 0x14	Länge (16 Bit) 0x03 0x00	'T' 0x54	Verzögerung in 10 us 0x00 0x00	crc (16 Bit) 0xE9 0x7E
Modul sendet	ACK 0x06				

9. Schnittstelle exklusiv anfordern

Alle 4 seriellen Schnittstellen werden parallel und gleichwertig behandelt. Um zu gewährleisten, dass eine Abfolge von Protokollpaketen ohne Unterbrechung durchgeführt wird, können die anderen seriellen Schnittstellen deaktiviert werden und die Schnittelle exklusiv angefordert werden. Dies ist zum Beispiel bei einem Projektupdate über USB sinnvoll.

Modul empfängt	DC4 0x14	Länge (16 Bit) 0x02 0x00	'G' 0x47	0x00 = Freigabe 0x01 = Anfordern	crc (16 Bit) 0xXX 0xXX
Modul sendet	ACK 0x06				
Modul sendet	DC4 0x14	Länge (16 Bit) 0x01 0x00	Aktiv (16 Bit) 0x00 = alle 0x01 = RS232 0x02 = SPI 0x03 = IIC 0x04 = USB		crc (16 Bit) 0xXX 0xXX

10. Break-Kommando, Ausführung Unterbrechen / Beenden

Falls in einem Makro eine Dauerschleife programmiert wurde oder eine Unterbechung des normalen Ablaufs gewünscht wird, kann mit diesem Befehl gezielt unterbrochen und beendet werden. Auch dieser Befehl eignet sich vor allem für Update-Vorgänge.

Modul empfängt Defaultwerte	DC4 0x14	Länge (16 Bit) 0x02 0x00	'C' 0x43	break 0x01 = Wait command 0x02 = aktuelles Makrofile 0x04 = Sendepuffer löschen 0x08 = Empfangsbuffer leeren 0x10 = Makrodefiniton (z.B. Portmakros) 0xFF = Alles Unterbrechen und Beenden	crc (16 Bit) 0xXX 0xXX
Modul sendet	ACK 0x06				

11. Hardware Reset

Das Modul wird mit diesem Protokollbefehl neu gestartet. Je nach Parameter wird nach dem Reset eine andere Startoption gewählt.

Modul empfängt Defaultwerte	DC4 0x14	Länge (16 Bit) 0x02 0x00	'B' 0x42	Option 0x00 = Normaler Neustart 0x01 = Neustart im Testmode 0x02 = Neustart ohne 'start.emc' 0x03 = Neustart ohne default Styles 0x04 = Bootmenü anzeigen (Projektauswahl) 0x05 = Reserved 0x06 = Mass Storage Mode (ab V1.2)	crc (16 Bit) 0xXX 0xXX
Modul sendet	ACK				

	0x06

CRC-Berechnung

Für die Berechnung der Prüfsumme wird eine zyklische Redundanzprüfung (CRC) eingesetzt. Eine gängige und bekannte CRC-Prüfung ist die CRC-CCITT. Als Startwert wird 0xFFFF verwendet. Im Folgenden sehen Sie eine typische C-Implementierung. Die Funktionen müssen extern aufgerufen werden. Die Prüfsumme muss mit dem Startwert vorbelegt werden.

```
//-------------------------------------------------------------------------
//Funktion: buffer2crc16()
//input:    ptr Datum, ptr auf CRC, Blocklänge
//output:   ---
//Beschr:   CRC-CCITT eines Speicherbereichs
//-------------------------------------------------------------------------
void buffer2crc16(UBYTE *dat, UINT16 *pCRC, UINT32 anz)
{
  while(anz--)
    crc16(*dat++, pCRC);
}

//-------------------------------------------------------------------------
//Funktion: sp_crc16()
//input:    Datum, ptr auf CRC
//output:   ---
//Beschr:   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 Protokoll

1. Befehle / Daten zum Modul senden

Dieser Protokoll-Befehl überträgt Daten zum Display. Es können mehrere Grafikbefehle in ein Protokollpaket verpackt werden. Sind die Daten größer als die maximale Paketgröße können die Daten auf mehrere Pakete aufgeteilt werden. Das Modul fügt die einzelnen Datenpakete wieder zusammen.

Modul empfängt	DC1 0x11	Länge (8 Bit) 0xXX	Data..... 0x....	bcc (8 Bit) 0xXX
Modul sendet	ACK 0x06			

Beispiel: #XCB25 ändert z.B. die Helligkeit auf 25%. Dazu gehört am Ende ein [LF] bzw. 0x0A als Abschlusskennung
Das Paket beginnt also mit DC1 und einer Längenangabe der Nutzdaten. Am Ende steht eine Prüfsumme (8 Bit Summe, Modulo 256) über das komplette Paket. Hier finden Sie einen Online-CRC-Calculator.
In Hex: **11 07 23 58 43 42 32 35 0A 89** (hier als Datei zum Download; diese kann z.B. per Drag-and-Drop einfach auf terminal.exe geschoben werden)

Beispiel: #XCB75 ändert z.B. die Helligkeit auf 75%.
In Hex: **11 07 23 58 43 42 37 35 0A 8E** (hier als Datei zum Download)

2. Inhalt des Sendepuffers anfordern

Fallen Daten im Modul an, werden diese im Sendepuffer des Moduls gespeichert. Über die seriellen Schnittstellen können die Daten angefordert werden. Ob Daten verfügbar sind kann über den Pin 20 SBUF herausgefunden werden, oder aber die übergeordnete Steuerung fragt zyklisch die Daten ab (polling).

Modul empfängt	DC2 0x12	Länge (8 Bit) 0x01	'S' 0x53	bcc (8 Bit) 0x66
Modul sendet	ACK 0x06			
Modul sendet	DC1 0x11	Länge (8 Bit) 0xXX	Data..... 0x....	bcc (8 Bit) 0xXX

3. Letztes Datenpaket wiederholen

Ist ein Empfangenes Packet des Moduls fehlerhaft (falsche Länge oder Prüfsumme) kann es erneut angefordert werden:

Modul empfängt	DC2 0x12	Länge (8 Bit) 0x01	'R' 0x52	bcc (8 Bit) 0x65
Modul sendet	ACK 0x06			
Modul sendet	DC1 0x11	Länge (8 Bit) 0xXX	Data..... 0x....	bcc (8 Bit) 0xXX

4. Pufferinformationen anfordern

Mit diesem Befehl wird nachgefragt, ob Nutzdaten zur Abholung bereit stehen (= Pin20 SBUF) und wie voll der Empfangspuffer des Displays bereits ist.

Modul empfängt	DC2 0x12	Länge (8 Bit) 0x01	'I' 0x49		bcc 0x5
Modul sendet	ACK 0x06				
Modul sendet	DC2 0x12	Länge (8 Bit) 0x02	Sendepuffer Bytes ready (8 Bit) 0xXX	Empfangspuffer Bytes free (8 Bit) 0xXX	bcc 0x

5. Protokolleinstellungen

Hierüber lässt sich die maximale Paketgröße welche das Display senden darf begrenzen. Voreingestellt ist eine Paketgröße mit bis zu 255 Byte Nutzdaten. Weiterhin lässt sich der Time-out in 1/100s einstellen. Der Time-out spricht an, wenn einzelne Bytes verloren gegangen sind. Danach muss das gesamte Paket nochmals übertragen werden.

Modul empfängt Defaultwerte	DC2 0x12	Länge (8 Bit) 0x03	'D' 0x44	Paketgröße Sendepuffer (8 Bit) 0xFF	Time-out (8 Bit) in 1/100s 0xC8 (=2 Sekunden)	bcc (8 0x20
Modul sendet	ACK 0x06					

6. Protokollinformationen

Abfrage der Protokolleinstellungen (siehe 5.).

Modul empfängt	DC2 0x12	Länge (8 Bit) 0x01	'P' 0x50		
Modul sendet	ACK 0x06				
Modul sendet	DC2 0x12	Länge (8 Bit) 0x03	Maximale Paketgröße Sendepuffer (8 Bit) 0xFF	Paketgröße Sendepuffer (8 Bit) 0xXX	Tir 0x

7. RS485 Adresse selektieren / deselektieren

Mit diesem Befehl lässt sich ein Modul am RS485-Bus selektieren oder deselektieren. Per Default ist ein Modul mit der Adresse 7 immer aktiv.

Modul empfängt Defaultwerte	DC2 0x12	Länge (8 Bit) 0x03	'A' 0x41	'S' (=Selektieren) 'D' (=Deselektieren) 0x53 oder 0x44	RS485-Adresse 0xXX	bcc (8 0xXX
Modul sendet	ACK 0x06 ----	→ Selektieren → Deselektieren				

8. RS485 Verzögerung Enable Signal

Einige RS485 benötigen eine gewisse Zeit um das Enable-Signal zu verändern und z.B. vom schreibenden in den lesenden Modus umzuschalten. Um eine erfolgreiche Kommunikation mit diesen Geräten zu ermöglichen kann mit diesem Befehl die Umschaltung in den schreibenden Modus verzögert werden.

Modul empfängt Defaultwerte	DC2 0x12	Länge (8 Bit) 0x03	'T' 0x54	Verzögerung in 10 us 0x00 0x00	bcc (8 Bit) 0x69

Modul sendet	ACK 0x06

9. Schnittstelle exklusiv anfordern

Alle 4 seriellen Schnittstellen werden parallel und gleichwertig behandelt. Um zu gewährleisten, dass eine Abfolge von Protokollpaketen ohne Unterbrechung durchgeführt wird, können die anderen seriellen Schnittstellen deaktiviert werden und die Schnittelle exklusiv angefordert werden. Dies ist zum Beispiel bei einem Projektupdate über USB sinnvoll.

Modul empfängt	DC2 0x12	Länge (8 Bit) 0x02	'G' 0x47	0x00 = Freigabe 0x01 = Anfordern	bcc (8 Bit) 0xXX
Modul sendet	ACK 0x06				
Modul sendet	DC2 0x12	Länge (8 Bit) 0x01	Aktiv (8 Bit) 0x00 = alle 0x01 = RS232 0x02 = SPI 0x03 = IIC 0x04 = USB		bcc (8 Bit) 0xXX

10. Break-Kommando, Ausführung Unterbrechen / Beenden

Falls in einem Makro eine Dauerschleife programmiert wurde oder eine Unterbrechung des normalen Ablaufs gewünscht wird, kann mit diesem Befehl gezielt Unterbrochen und Beendet werden. Auch dieser Befehl eignet sich vor allem für Update-Vorgänge.

Modul empfängt Defaultwerte	DC2 0x12	Länge (8 Bit) 0x02	'C' 0x43	break 0x01 = Wait command 0x02 = aktuelles Makrofile 0x04 = Sendepuffer löschen 0x08 = Empfangsbuffer leeren 0x10 = Makrodefiniton (z.B. Portmakros) 0xFF = Alles Unterbrechen und Beenden	bcc (8 Bit) 0xXX
Modul sendet	ACK 0x06				

11. Hardware Reset

Das Modul wird mit diesem Protokollbefehl neu gestartet. Je nach Parameter wird nach dem Reset eine andere Startoption gewählt.

Modul empfängt Defaultwerte	DC2 0x12	Länge (8 Bit) 0x02	'B' 0x42	Option 0x00 = Normaler Neustart 0x01 = Neustart im Testmode 0x02 = Neustart ohne 'start.emc' 0x03 = Neustart ohne default Styles 0x04 = Bootmenü anzeigen (Projektauswahl) 0x05 = Reserved 0x06 = Mass Storage Mode (ab V1.2)	bcc (8 Bit) 0xXX
Modul sendet	ACK 0x06				

BCC-Berechnung

Für die Berechnung der Prüfsumme wird eine einfache 8-Bit Summenprüfung (Modulo 256) benötigt. Im Folgenden sehen Sie eine typische C-Implementierung.

```
//-------------------------------------------------------------------------
//Funktion: buffer2bcc()
//input:    ptr Datum, blocklänge
//output:   ---
//Beschr:   Byte bcc für einen Speicherbereich
//-------------------------------------------------------------------------
UBYTE buffer2bcc(UBYTE *dat, UBYTE anz)
{
  UBYTE bcc = 0;
  while(anz--)
    bcc += *dat++;
  return bcc;
}
```

BEFEHLSÜBERSICHT

Die EA uniTFTs-Serie verfügt über einen integrierten Befehlssatz, welcher grafische Primitive, Rechnungen, Hardwareansteuerungen und vieles mehr erlaubt.
Die Befehle können zur Laufzeit über die seriellen Schnittstellen übertragen werden oder in sogenannten Macrofiles auf das interne FLASH des Moduls abgelegt sein.
In den folgenden Tabellen sind alle Befehle beschrieben. Die Defaultwerte sind in Klammern hinter den jeweiligen Parametern angegeben. In SCHWARZ geschriebene Parameter sind anzugeben, GRAUE sind optional.

Alle Befehlsgruppen auf einen Blick

Terminalfenster #Y	Im Terminalfenster werden alle empfangenen Daten direkt angezeigt. Dieses Fenster ist nützlich um während der Entwicklungszeit schnell einfache Ausgaben zu erstellen oder Fehlermeldungen zu erhalten.
Textausgabe / Zeichenketten #S	Die Gruppe umfasst Befehle um einfache,sowie formatierte und sich selbst ändernde Zeichenketten darzustellen. Darüber hinaus gibt es die Möglichkeit mit Hilfe von EditBoxen (einzeilige Eingaben) und StringBoxen (mehrzeilige Ausgaben) Texte zu platzieren.
Bilder #P	Befehlsgruppe um Bilder darzustellen. Mit der Designsoftware uniTFT Designer können folgende Dateiformate/Grafikformate verwendet werden: png, bmp, jpg, jpeg, tga, gif, g16, svg, svgz. Der uniFTF Designer wandelt die Daten automatisch in das korrekte interne Format um. Wenn der uniTFT Designer nicht verwendet wird, können diese Dateien mit dem Tool EAconvert.exe (im Verzeichnis \Simulator_and_Tools) konvertiert werden (→ evg, epg, epa)
Touchfunktionen #T	Befehlsgruppe um Touchfunktionen zu ermöglichen. Es können einfache Taster und Schalter platziert werden, sowie Radiobuttons, Schieberegler, Bargraphen und Dreh-/Zeigerinstrumente.
Zeichnen / grafische Primitive #G	Befehlsgruppe um geometrische Formen und Linien darzustellen.
Bargraph / Instrumente #I	Befehlsgruppe um Bargraphen, Schieberegler und Dreh-/Zeigerinstrumente darzustellen.
Eingabeelement per Touch #E	Befehlsgruppe Toucheingabelemente wie Menüs, SpinBoxen oder ComboBoxen zu erstellen.
Keyboard / Tastatur #K	Befehlsgruppe um ein Keyboard für Werteingaben darzustellen. Im Normalfall wird das Keyboard mit einer EditBox verbunden.
Action / Animation #A	Befehlsgruppe um Objekte zu animieren, z.B. Erscheinen, Wegfliegen, Rotieren oder Ausblenden zu lassen.
Objektverwaltung #O	Befehlsgruppe um Objekte zu Verwalten, zu Verändern oder zu Gruppieren.

Styles #C	Befehlsgruppe um Formatvorlagen zu erstellen. Das Aussehen jedes Objekts basiert auf einer Style passend zur Objektart. Für jeden Style stehen maximal 100 zur Verfügung.
Makros #M	Einzelne oder mehrere Befehlsfolgen können als sogenannte Makros zusammengefasst und auf dem internen FLASH fest abgespeichert werden. Diese können dann mit den Befehlen gestartet werden.
Variablen / Register #V	Befehlsgruppe um interne Rechnungen und logische Operationen auszuführen. Mit Hilfe der Stringfiles kann eine Mehrsprachigkeit realisiert werden. Es sind 200 Register vorhanden. Stringregister können bis zu 200 Zeichen aufnehmen, bei Festkommaregistern wird mit signed 32 Bit, bei Fließkommaregistern wird mit 23 Bit Mantisse, 8 Bit Exponent, 1 Bit signed gerechnet.
I/O Port #H	Das Modul verfügt über 8 I/O Portleitungen, welche auf bis zu 136 erweitert werden können. Bei Änderungen der Porteingangspins können Makros gestartet werden, siehe #MHP, und #MHB.
Analog Input #H	Befehlsgruppe um den Analogeingang des Moduls zu parametrisieren und auszulesen. Das Modul hat vier 12-Bit Analogeingänge. Bei Änderungen des Analogeingangs kann ein Makro gestartet werden, siehe #MHA.
PWM Output #H	Befehlsgruppe um den PWM Output einzustellen
Serielle Master-Schnittstellen #H	Befehlsgruppe um die 3 seriellen Schnittstellen als Master zu verwenden z.B. zum Anschluss weiterer Peripherie an das Modul wie ein Temperaturfühler.
Sound #H	Befehlsgruppe um töne abzuspielen
Uhrzeit #W	Befehlsgruppe um mit der RTC zu arbeiten
Files auf der SD-Card #F	Befehlsgruppe um Dateizugriffe zu realisieren
Systembefehle #X	Einstellung der EA uniTFTs-Serie.
Antwort / Rückmeldung	Das Modul stellt nach Anfragen oder Touch-Ereignissen Informationen in seinen Sendepuffer. Erklärung der Rückantworten.

Befehlssyntax

Alle Befehle sind gleich aufgebaut:

Start	Befehlscode	Parameter	Abschluss
#	XXX	123, $52, %01101010, "Hello"; R0	[CR]LF
ASCII : 35 (0x23) UniCode: 23 (0x23 0x00)	3-stellige Buchstabenfolge	Angabe der Parameter	ASCII : [13] 10 ([0x0D] 0x0A) UniCode: [13] 19 ([0x0D 0x00] 0x CR ist optional

Wenn nicht anders angegeben werden alle Parameter als 16-Bit Werte übergeben.

Parameter

Zahlen

123	dezimale Übergabe als ASCII-Zeichen
$5A	hexadezimale Übergabe als ASCII Zeichen
% 1010001	binäre Übergabe als ASCII Zeichen
5-8	Übergabe eines Bereichs. Befehle die Eigenschaften mehrerer Objekte beeinflussen können als Objektbereich übergeben werden. In diesem Fall Objekt-ID 5,6,7,8
?x	Code von einem Zeichen (Unicode/ASCII)
R0 ... R499	Übergabe des Registerwertes
Q0 ... Q499	Indizierte Übergabe des Registerwertes (Pointer) → R (R0 ... R499)
(....)	Kalkulationsstring Ergebnis übernehmen
G len32 data....	Übergabe von binären Daten: len32 gibt die Datenlänge an (bereits als binärer 32-Bit Wert)
!index!	Werte des Indexes aus einem Stringfile übernehmen

Strings

"string" oder 'string'	normale Stringübergabe
"str" 32 "str"	Normaler String mit beliebigen Codes, welche mit in den Gesamtstring eingetragen werden
"str1"; "str2"	**Semikolon bildet den Stringabschluss**, wichtig bei der Übergabe zweier Strings oder wenn andere Parameter folgen.
S0 ... S499	Stringregister übernehmen
T0 ... T499	Stringregister aus Registernummer übernehmen S(R0 ... R499)
U"Hello"	Zeichen nach dem U als 16 Bit Unicode (bis zum nächsten # oder V auch CR + LF)
V"Hello"	Zeichen nach dem V als 8 Bit ASCII (bis zum nächsten # oder U auch CR + LF)
!index!	String des Indexes aus einem Stringfile übernehmen

Die einzelnen Parameter werden mit Leerzeichen (' '), Komma (','), Semikolon (';') oder Punkt ('.') getrennt. **Für die Trennung von Strings ist ein Semikolon zwingend erforderlich**.

Befehlsabschluss
Der Befehlsabschluss ist immer ein LF (0x0A). Ein vorangestelltes CR (0x0D) ist erlaubt und wird ignoriert.

Kommentare
In Makrofiles können Kommentarzeilen eingefügt werden. Ein Kommentar beginnt mit #- und gilt bis Zeilenende / Befehlsabschluss (LF).

Kalkulationen
Jeder numerische Parameter kann durch eine Kalkulation ersetzt werden. Die Kalkulation muss in Klammern () eingeschlossen als Parameter übergeben werden. Unter den Kalkulationsbefehle finden Sie eine Auflistung aller Operationen, darunter mathematische und logische, aber auch Modul-bezogene, wie zum Beispiel Uhrzeit oder Objekteigenschaften auslesen sowie Portfunktionen.

Terminalfenster #Y

Das Terminalfenster hilft die serielle Verbindung zu überprüfen. Dazu legt man den Pin 14 (DPROT) auf GND (Protokoll aus) und schaltet das Display ein. Alle über eine der seriellen Interfaces empfangenen Daten werden nun direkt dargestellt (ASCII Codes sowie CR/LF). FF (0x0C) löscht das Terminalfenster und setzt den Cursor nach links oben in die erste Zeile.

Das Terminalfenster bietet gleichzeitig eine einfache Möglichkeit Ausgaben zum Test während der Entwicklung und auch Fehlermeldungen zu erhalten.

Terminalfenster Einstellungen

Größen-/Positionseinstellungen (Terminal Define Window)	**#YDW**	x(0), y(0), Anker(7), Spalten(x-Displayauflösung/8), Zeilen(y-Displayauflösung/16)
Farbeinstellung (Terminal Define Color)	**#YDC**	Text-Farbe, Text-Deckkraft(100), Hintergrund-Farbe($000000), Hintergrund-Deckkraft(0)
Zeichenreihenfolge (Terminal Define Layer)	**#YDL**	Zeichenreihenfolge
Terminalfenster Ein-/Ausschalten (Terminal Define Output)	**#YDO**	Ausgabe, Sichtbar(=Ausgabe)
Cursor Ein-/Ausschalten (Terminal Cursor Blink)	**#YCB**	Cursor
Cursor Position setzen (Terminal Cursor Position)	**#YCP**	Spalte, Zeile(keine Änderung)
Cursor Position sichern (Terminal Cursor Save)	**#YCS**	
Cursor Position wiederherstellen (Terminal Cursor Restore)	**#YCR**	

Terminalfenster Ausgaben

Ausgabe Zeichenkette (Terminal Print Ascii)	**#YPA**	Zeichenkette
Ausgabe formatierte Zeichenkette (Terminal Print Formated)	**#YPF**	"Formatstring"; Wert1, Wert2, ..., WertN
Ausgabe Datum/Uhrzeit (Terminal Print Date)	**#YPD**	"Datumsformat"; date (aktuelle Zeit)
Ausgabe Modulkonfiguration (Terminal Print Info)	**#YPI**	
Ausgabe Versionsstring (Terminal Print Version)	**#YPV**	

Terminalfenster Einstellungen

In dieser Befehlsgruppe werden alle wichtigen Einstellungen für das Terminalfenster zusammengefasst.

Größen-/Positionseinstellung

#YDW	x(0), y(0), Anker(7), Spalten(x-Displayauflösung/8), Zeilen(y-Displayauflösung/16)

Die Abmessungen des Terminalfensters werden definiert. Die Breite ergibt sich aus der Angabe der Spalten und Zeilen und der Schriftgröße (8x16): Breite in Pixeln =8∗Spalten; Höhe in Pixeln =16∗Zeilen


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

Farbeinstellung

#YDC	Text-Farbe, Text-Deckkraft(100), Hintergrund-Farbe($000000), Hintergrund-Deckkraft(0)

Der Befehl stellt die Farbe und die Deckkraft der Schrift und des Hintergrunds ein. Die Farbe wird jeweils als 24 Bit RGB Wert übergeben (z.B. `$c80000`, `%11001000000000000000000`, `(RGB(200,0,0))`).


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

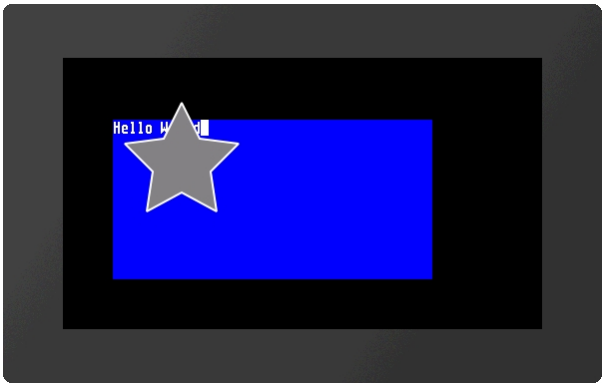
Zeichenreihenfolge

#YDL	Zeichenreihenfolge

Mit dem Befehl wird die **Zeichenreihenfolge** des Terminalfensters festgelegt:

Zeichenreihenfolge	
0	Terminal wird hinter allen Objekten dargestellt
1	Terminal wird im Vordergrund dargestellt

Standardmäßig ist das Terminal immer im Vordergrund.



```
...
#YDL 0
...
```

Terminalfenster Ein-/Ausschalten

#YDO	Ausgabe, Sichtbar (=Ausgabe)

Mit diesem Befehl kann zum einen die Terminalausgabe aktiviert bzw. deaktivert werden und zum anderen die Sichtbarkeit eingestellt werden. Wird nur ein Parameter übergeben, so gilt dieser für beide Werte.

Festlegung der **Ausgabe**:

Ausgabe	
0	Terminalausgabe ist deaktiviert
1	Terminalausgabe ist aktiviert

Festlegung der **Sichtbarkeit**:

Sichtbarkeit	
0	Terminal ist unsichtbar
1	Terminal ist sichtbar

`#YDO 0`	Ausgaben sind deaktiviert und das Terminal ist unsichtbar
`#YDO 1`	Ausgaben sind aktiviert und das Terminal ist sichtbar
`#YDO 0,1`	Ausgaben sind deaktiviert und das Terminal ist sichtbar
`#YDO 1,0`	Ausgaben sind aktiviert und das Terminal ist unsichtbar.

Cursor Ein-/Ausschalten

#YCB	Cursor

Mit dem Befehl wird die Sichtbarkeit des **Cursors** eingestellt:

Cursor	
0	Cursor ist unsichtbar
1	Cursor ist sichtbar

Cursor Position setzen

#YCP	Spalte, Zeile(keine Änderung)

Der Befehl setzt die Cursorposition innerhalb des Terminalfensters. Wird keine Zeile angegeben, so wird sie nicht geändert. Die Position beginnt bei (1,1).

```
...
#YCP 10,2
...
```

Cursor Position sichern

#YCS	

Die aktuelle Position des Cursors wird gesichert.

Cursor Position wiederherstellen

#YCR	

Der Cursor wird auf die zuletzt gesicherte Position gesetzt.

Terminalfenster Ausgaben

Diese Gruppe umfasst Befehle Zeichenketten und vordefinierte Ausgaben am Terminal anzuzeigen.

Ausgabe Zeichenkette

#YPA	Zeichenkette

Die Zeichen(ketten) werden im Terminalfenster ausgegeben. Es können sowohl ganze Zeichenketten (z.B. `"Test"`, `'Test'`) oder einzelne ASCI Zeichen (`$21`, `33`, `?!`) übergeben werden. Das Semikolon bildet den Stringabschluss.


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

Ausgabe formatierte Zeichenkette

#YPF	"Formatstring"; Wert1, Wert2, ..., WertN, Wert1, Wert2, ..., WertN

Die formatierte Zeichenkette wird auf dem Terminal ausgegeben. Wiederholt sich der Variablensatz, wird der **Formatstring** erneut verwendet. Im Unterkapitel Formatierte Zeichenkette ist der Aufbau genauer erläutert.


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

Ausgabe Datum/Uhrzeit

#YPD	"Datumsformat"; date (aktuelle Zeit)

Auf dem Terminal wird das Datum und die Uhrzeit ausgegeben. Die Darstellungsweise richtet sich nach dem **Datumsformat**. Der Aufbau ist im Unterkapitel Datumsformate genauer erläutert.


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

Ausgabe Modulkonfiguration

#YPI	

Im Terminal werden Modulparameter (u.a. Firmwareversion, Auflösung, oder Schnittstellenparameter) ausgegeben

Ausgabe Versionsstring

#YPV	

Im Terminal wird die Firmwareversion des Moduls ausgegeben.

Textausgabe / Zeichenketten #S

Die Gruppe umfasst Befehle um einfache,sowie formatierte und sich selbst ändernde Zeichenketten darzustellen. Darüber hinaus gibt es die Möglichkeit mit Hilfe von EditBoxen (einzeilige Eingaben) und StringBoxen (mehrzeilige Ausgaben) Texte zu platzieren.

Einfache Zeichenketten

Zeichenkette platzieren (String Static Place)	**#SSP**	Obj-ID, TextStyle-Nr., x, y, Anker, Text
Zeichenkette ändern (String Static Change)	**#SSC**	Obj-ID, Text

Formatierte Zeichenketten

Formatierte Zeichenkette platzieren (String Formated Place)	**#SFP**	Obj-ID, TextStyle-Nr., x, y, Anker, "Formatstring"; Wert1, Wert2, ..., WertN, Wert1 ...,WertN,...
Parameter aus formatierter Zeichenkette ändern (String Formated Change)	**#SFC**	Obj-ID, Wert1, Wert2, ..., WertN
Zeichenkette in formatierte Zeichenkette umwandeln (String Formated Format)	**#SFF**	Obj-ID, "Formatstring"; Wert1, Wert2, ..., WertN

Formatierte Zeichenketten mit automatischer Update Funktion

Formatierte Zeichenkette mit Auto Update platzieren (String Automatic Place)	**#SAP**	Obj-ID, TextStyle-Nr., x, y, Anker, "Formatstring"; (Kalkulation), Wert1 ...., WertN
Kalkulation aus formatierter Zeichenkette mit Auto Update ändern (String Automatic Change)	**#SAC**	Obj-ID, (Kalkulation)
Zeichenkette in formatierte Zeichenkette mit Auto Update umwandeln (String Automatic Format)	**#SAF**	Obj-ID, "Formatstring"; (Kalkulation), Wert1 ...., WertN

Zeichenkette mit Datum / Uhrzeit

Zeichenkette mit Datum/Uhrzeit platzieren (String Date Place)	**#SDP**	Obj-ID, TextStyle-Nr., x, y, Anker, "Datumsformat"; date (aktuelle Zeit)
Datum/Uhrzeit in Zeichenkette ändern (String Date Change)	**#SDC**	Obj-ID, date (aktuelle Zeit)
Zeichenkette in Zeichenkette mit Datum/Uhrzeit umwandeln (String Date Format)	**#SDF**	Obj-ID, "Datumsformat"; date (aktuelle Zeit)

EditBox

EditBox platzieren (String Edit Place)	**#SEP**	Obj-ID, DrawStyle-Nr., x, y, Anker, Breite, Höhe, Radius, TextStyle-Nr., RandX(0), RandY(0)
Defaultstring für EditBox festlegen (String Edit Default)	**#SED**	Obj-ID, "Standardtext"; "Standardtext (Obj-ID+1)"; "Standardtext (Obj-ID+2)";....
Strings/Codes an EditBox senden (String Edit Codes)	**#SEC**	Obj-ID, "String"; "String (Obj-ID+1)"; "String (Obj-ID+2)";....

EditBox mit Keyboard verbinden (String Edit Keyboard)	**#SEK**	Keyboard-ID, Obj-ID, Obj-ID+1, ...
EditBox aktivieren/deaktivieren (String Edit Activate)	**#SEA**	Obj-ID(0), Keyboard-ID(0)
Zulässinge Eingaben für EditBox festlegen (String Edit Range)	**#SER**	Obj-ID, Codes
Eingabemaske für EditBox definieren (String Edit Mask)	**#SEM**	Obj-ID, "Eingabemaske"; Platzhalter
Passwortmodus für EditBox definieren (String Edit Wildcard)	**#SEW**	Obj-ID, Ersatzzeichen

StringBox

StringBox platzieren (String Box Place)	**#SBP**	Obj-ID, x, y, Anker, Breite, Höhe, Radius, ScrollbarBreite(Texthöhe)
Style für StringBox definieren (String Box Styles)	**#SBS**	Obj-ID, DrawStyle-Nr. Hintergrund, DrawStyle-Nr. Scrollbar, TextStyle-Nr., RandX(0), RandY(0), AutoWrap(1)
Absatz in der StringBox hinzufügen (String Box Add)	**#SBA**	Obj-ID, Absatz, "Text"; "Text (Zeile+1)"; "Text (Zeile+2)";....
Absatz in der StringBox entfernen (String Box Delete)	**#SBD**	Obj-ID, Absatz, Zeile1. ...
Hinzufügen einer Textdatei in der StringBox (String Box File)	**#SBF**	Obj-ID, Absatz, <Textdatei>
In StringBox zu Zeile springen (String Box Offset)	**#SBO**	Obj-ID, Zeile, Zeit (0), Aktionskurve-Nr (0)

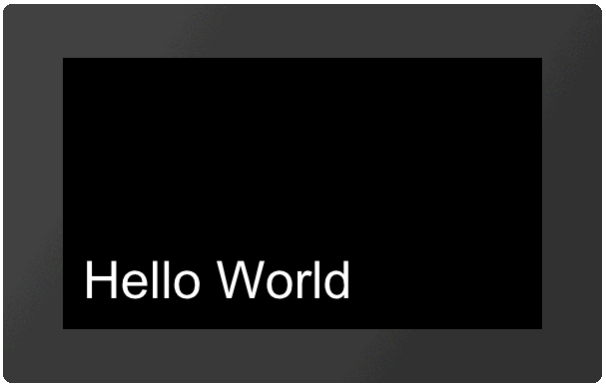
Einfache Zeichenkette

Diese Gruppe umfasst Befehle zur Platzierung und Änderung einfacher Zeichenketten.

Zeichenkette platzieren

#SSP	Obj-ID, TextStyle-Nr., x, y, Anker, Text

Eine Zeichenkette wird mit dem gegebenen **Anker** an die Position **x**, **y** platziert. Mit dem TextStyle wird das Aussehen der Zeichenkette bestimmt (**TextStyle-Nr.**). Im Unterkapitel TextStyle ist dies genauer erläutert.

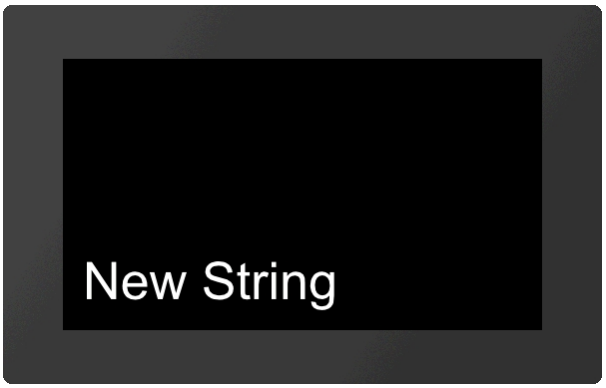



```
...
#SSP 1,1,20,20,7,"Hello World";
...
```

Zeichenkette ändern

#SSC	Obj-ID, Text

Der Befehl ändert eine vorhandene Zeichenkette. Andere Objekteigenschaften (Position, Style, usw.) bleiben dabei unverändert.




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

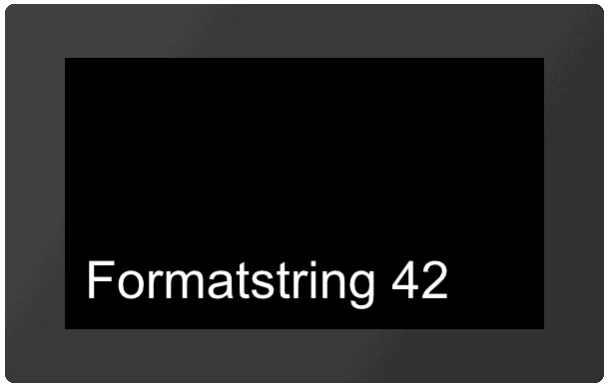
Formatierte Zeichenketten

Diese Gruppe enthält Befehle zur Platzierung und Änderung formatierter Zeichenketten.

Formatierte Zeichenkette platzieren

#SFP	Obj-ID, TextStyle-Nr., x, y, Anker, "Formatstring"; Wert1, Wert2, ..., WertN, Wert1 ...,WertN,...

Eine formatierte Zeichenkette wird mit dem gegebenen **Anker** an die Position **x**, **y** platziert. Mit dem TextStyle wird das Aussehen der Zeichenkette bestimmt (**TextStyle-Nr.**). Im Unterkapitel TextStyle ist dies genauer erläutert. Wiederholt sich der Variablensatz, wird der **Formatstring** erneut verwendet. Im Unterkapitel Formatierte Zeichenkette ist der Aufbau näher beschrieben.

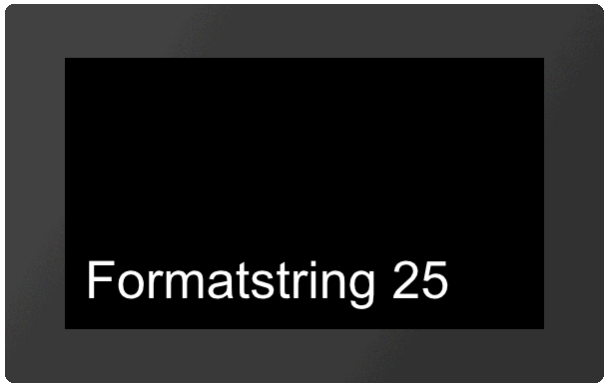



```
...
#SFP 1,1,20,20,7,"Formatstring %d"; 42
...
```

Parameter aus formatierter Zeichenkette ändern

#SFC	Obj-ID, Wert1, Wert2, ..., WertN

Dieser Befehl ändert die Parameter einer formatierten Zeichenkette. Die Objekteigenschaften (Position, Style, usw.) bleiben dabei unverändert.

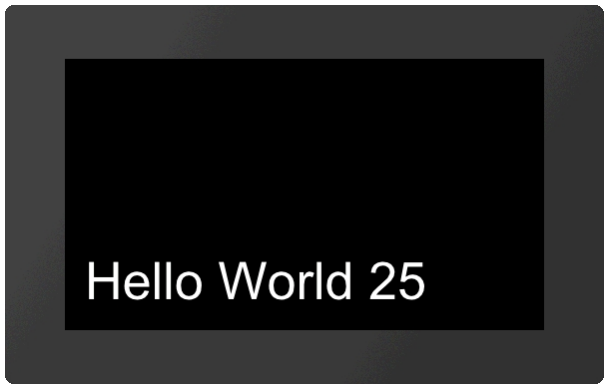



```
...
#SFC 1,25
...
```

Zeichenkette in formatierte Zeichenkette umwandeln

#SFF	Obj-ID, "Formatstring"; Wert1, Wert2, ..., WertN

Eine vorhandene Zeichenkette wird in eine formatierte Zeichenkette geändert. Andere Objekteigenschaften (Position, Style, usw.) bleiben dabei unverändert. Im Kapitel Formatierte Zeichenkette ist der Aufbau genauer erläutert.




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

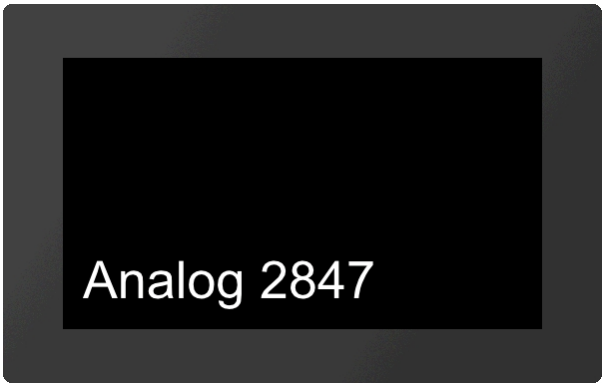
Formatierte Zeichenketten mit automatischer Update Funktion

In dieser Gruppe sind Befehle zur Platzierung von formatierten Zeichenketten, deren Wert sich automatisch ändert, zusammengefasst.

Formatierte Zeichenkette mit Auto Update platzieren

#SAP	Obj-ID, TextStyle-Nr., x, y, Anker, "Formatstring"; (Kalkulation), Wert1 ...., WertN

Es wird eine formatierte Zeichenkette mit dem gegebenen **Anker** an die Position **x**, **y** platziert. Mit dem TextStyle wird das Aussehen der Zeichenkette bestimmt (**TextStyle-Nr.**). Im Unterkapitel Formatierte Zeichenkette ist der Aufbau des Strings näher beschrieben. Die Ausgabe wird erneuert, sobald sich die **Kalkulation** ändert. Sind die weiteren Parameter (Wert1, ... WertN) ebenfalls Kalkulationen wird auch ihr Wert neu berechnet (nur wenn sich der Wert der ersten **Kalkulation** ändert).

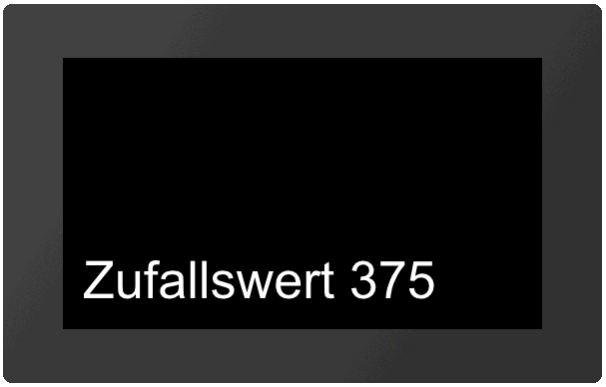



```
...
#SAP 1,1,20,20,7,"Analog %d";(analog(0))
...
```

Kalkulation aus formatierter Zeichenkette mit Auto Update ändern

#SAC	Obj-ID, (Kalkulation)

Der Befehl ändert die Kalkulation einer formatierten Zeichenkette mit Auto Update. Die neue Kalkulation bestimmt nur den Zeitpunkt der erneuten Ausgabe der Zeichenkette, ohne die angezeigten Werte/Kalkulation zu beeinflussen.

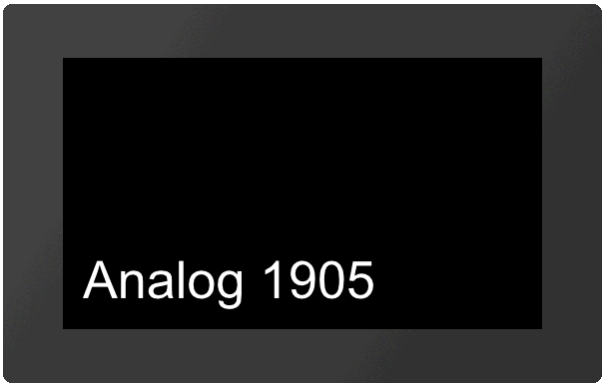



```
...
#SAP 1,1,20,20,7,"Zufallswert %d";
(rand())  /**Ausgabe eines Zufallswertes
#SAC 1,
(time())                                /**Änderu
ng des Zufallswertes nur alle Sekunde
...
```

Zeichenkette in formatierte Zeichenkette mit Auto Update umwandeln

#SAF	Obj-ID, "Formatstring"; (Kalkulation), Wert1 ...., WertN

Eine vorhandene Zeichenkette wird in eine formatierte Zeichenkette mit Auto Update Funktion geändert. Andere Objekteigenschaften (Position, Style, usw.) bleiben dabei unverändert. Im Unterkapitel Formatierte Zeichenkette ist der Aufbau genauer erläutert. Die Zeichenkette erneuert die Ausgabe, sobald sich die **Kalkulation** ändert. Sind die weiteren Parameter (Wert1, ... WertN) ebenfalls Kalkulationen wird auch ihr Wert neu berechnet (nur wenn sich der Wert der ersten **Kalkulation** ändert).




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

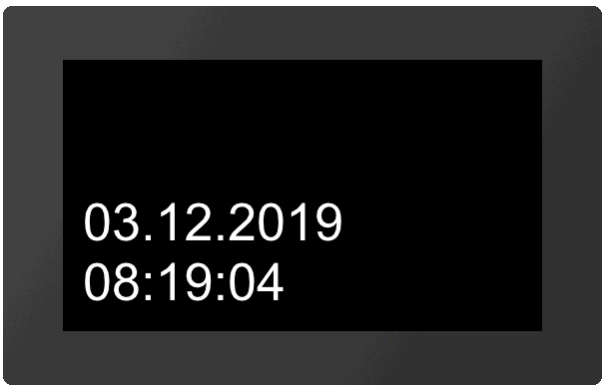
Zeichenkette mit Datum / Uhrzeit

In dieser Gruppe sind Befehle zur Platzierung von Zeichenketten mit Ausgabe des Datums und/oder Uhrzeit zusammengefasst.

Zeichenkette mit Datum/Uhrzeit platzieren

#SDP	Obj-ID, TextStyle-Nr., x, y, Anker, "Datumsformat"; date (aktuelle Zeit)

Es wird eine Zeichenkette mit Datum und Uhrzeit und dem gegebenen **Anker** an die Position **x**, **y** platziert. Die Darstellungsweise richtet sich nach dem **Datumsformat**. Der Aufbau ist im Unterkapitel Datumsformate näher beschrieben. Wird die aktuelle Uhrzeit ausgegeben passt sich die Ausgabe der aktuellen Uhrzeit automatisch an. Mit dem TextStyle wird das Aussehen der Zeichenkette bestimmt (**TextStyle-Nr.**). Im Unterkapitel TextStyle ist dies genauer erläutert.

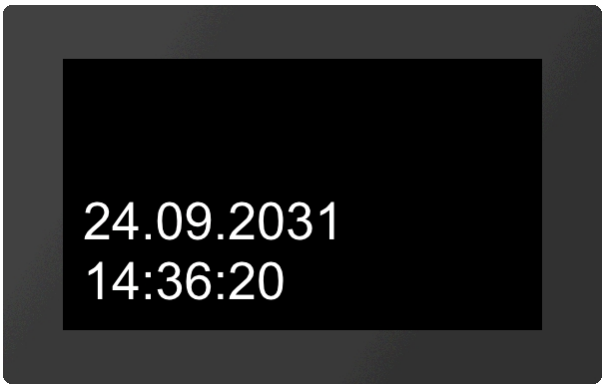



```
...
#SDP 1,1,20,20,7,"%D.%M.%Y|%h:%m:%s";
...
```

Datum/Uhrzeit in Zeichenkette ändern

#SDC	Obj-ID, date (aktuelle Zeit)

Der angezeigte Zeit des Datumsformat wird geändert. Andere Objekteigenschaften (Position, Style, usw.) bleiben dabei unverändert. Der Aufbau ist im Unterkapitel Datumsformate näher beschrieben.

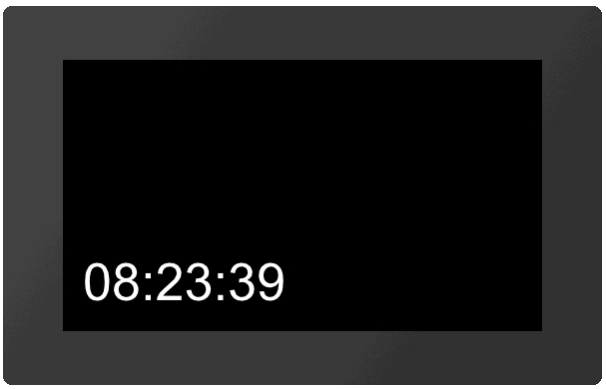



```
...
#SDC 1,(datetime(14,36,20,24,09,2031))
...
```

Zeichenkette in Zeichenkette mit Datum/Uhrzeit umwandeln

#SDF	Obj-ID, "Datumsformat"; date (aktuelle Zeit)

Eine vorhandene Zeichenkette wird in eine Zeichenkette mit Datum/Uhrzeit geändert. Andere Objekteigenschaften (Position, Style, usw.) bleiben dabei unverändert. Der Aufbau ist im Unterkapitel Datumsformate näher beschrieben.




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

EditBox

EditBoxen werden für die Eingabe von Zeichen verwendet. Die Eingabe erfolgt dabei in der Regel mit Hilfe eines Keyboards. Die Definition einer Tastatur ist im Unterkapitel Keyboard/Tastatur näher erläutert. Darüber hinaus kann die Eingabe auch per Befehl erfolgen (siehe #SEC). Damit Eingaben über die Tastatur in der EditBox landen, muss die Box mit dem Keyboard verbunden sein (siehe #SEK). Zudem muss die EditBox aktiv sein. Dies kann entweder per Befehl (#SEA) oder per Touch (#TID) erfolgen. Im Folgenden Beispiel wird eine EditBox platziert, mit einem Keyboard verbunden und per Touch aktiviert. Die Definition der Tastatur ist in diesem Beispiel nicht enthalten.


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

EditBox platzieren

#SEP	Obj-ID, DrawStyle-Nr., x, y, Anker, Breite, Höhe, Radius, TextStyle-Nr., RandX(0), RandY(0)

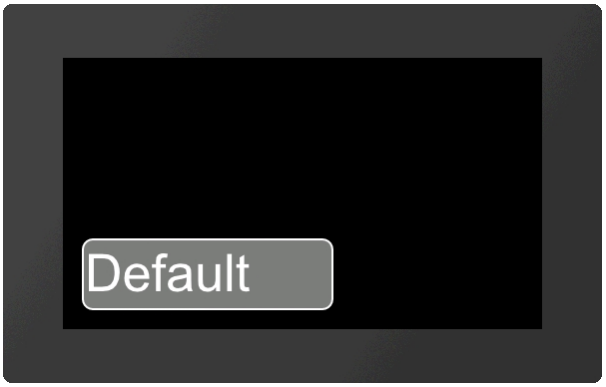
Eine EditBox wird mit dem gegebenen **Anker** an die Position **x**, **y** mit definierter **Höhe** und **Breite** platziert. Sie wird in der Regel zusammen mit einem Keyboard für die Eingabe von Zeichen verwendet. Der DrawStyle definiert das Aussehen des Hintergrundes der EditBox (**DrawStyle-Nr.**). Der Aufbau ist im Unterkapitel DrawStyle näher beschrieben. Der Parameter **Radius** gibt die Eckenabrundung an. Mit dem TextStyle wird das Aussehen der Zeichenkette bestimmt (**TextStyle-Nr.**). Im Unterkapitel TextStyle ist dies genauer erläutert. Mit den beiden optionalen Parametern (**RandX** und **RandY**) kann der Abstand des Textes zum Rand der Box angegeben werden.

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

Defaultstring für EditBoxfestlegen

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

Ein Standardtext wird für die EditBox festgelegt. Weitere Strings geben den Defaultstring für weitere EditBoxen mit den Objekt-IDs **Obj-ID+1**, ..., **Obj-ID+n** an.

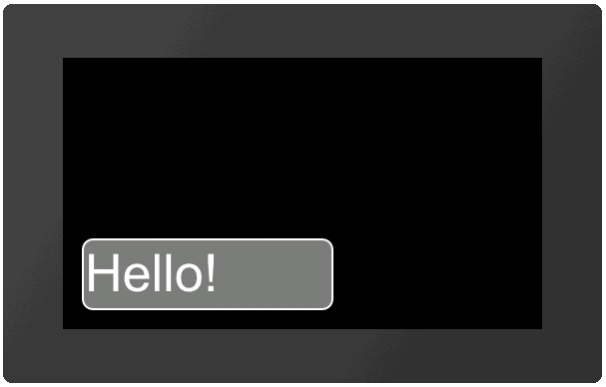



```
...
#SED 1,"Default";
...
```

Strings/Codes an EditBox senden

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

Mit dem Befehl können Strings und Codes an die EditBox gesendet werden. Weitere String werden an die EditBoxen mit den Objekt-IDs **Obj-ID+1**, ..., **Obj-ID+n** gesendet.




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

EditBox mit Keyboard verbinden

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

Dieser Befehl verbindet ein Keyboard (**Keyboard-ID**) mit einer oder mehreren EditBoxen (**Obj-ID**)

EditBox aktivieren/deaktivieren

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

Der Befehl aktiviert bzw. deaktiviert EditBoxen.

Aktivieren:

Obj-ID	Objekt ID der EditBox
Keyboard-ID	nicht notwendig

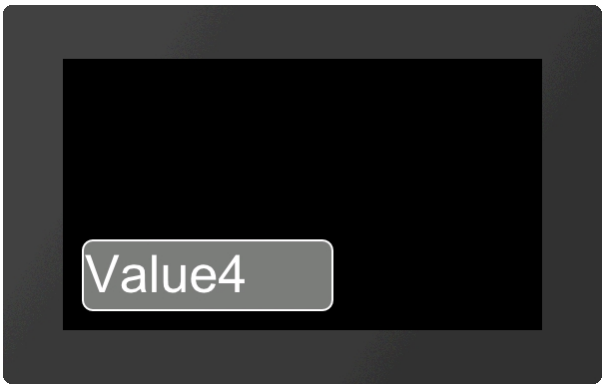
Deaktivieren:

Obj-ID	0	
Keyboard-ID	0	alle EditBoxen werden deaktiviert
	Keyboard-ID	alle EditBoxen, die dem Keyboard zugeordnet sind, werden deaktiviert

Zulässige Eingaben für EditBox festlegen (ab V1.2)

#SER	Obj-ID, Codes

Der Befehl legt zulässige Eingaben fest, die in der EditBox angezeigt werden. Gültige Zeichen (Codes) werden durch Komma getrennt oder als Bereichstring angegeben (z.B. "0-9A-Za-z", welcher alle Ziffern und das lateinische Alphabet erlaubt).




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

Eingabemaske für EditBox definieren (ab V1.2)

#SEM	Obj-ID, "Eingabemaske"; Platzhalter

Für die EditBox wird eine Eingabemaske definiert. Der Parameter Platzhalter definiert den sichtbaren Charaktercode (z.B. '_'). Folgende Masken sind möglich:

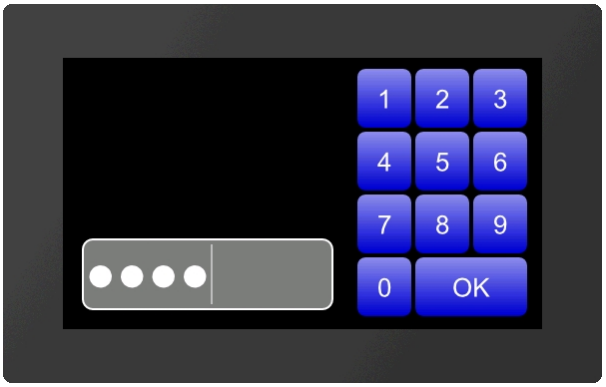
Typ	Maske	Beispiel
Integer	%MaximalwertI oder % Von;BisI	`"%42I"`; oder `"%10;25I"`;
Float	%MaximalwertF oder % Von;BisF	`"%23.4I"`; oder `"% 0.5;7.9I"`;
ASCII	%StellenanzahlA	`"%4A"` (maximal 4 Zeichen aus dem ASCII Bereich)
Unicode	%StellenanzahlU	`"%4U"` (maximal 4 Zeichen aus dem Unicode Bereich)
Bereich	%StellenanzahlR	`"%4R"` (maximal 4 Zeichen aus dem Bereich #SER)


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

Passwortmodus für EditBox definieren (ab V1.2)

#SEW	Obj-ID, Wildcardcorde

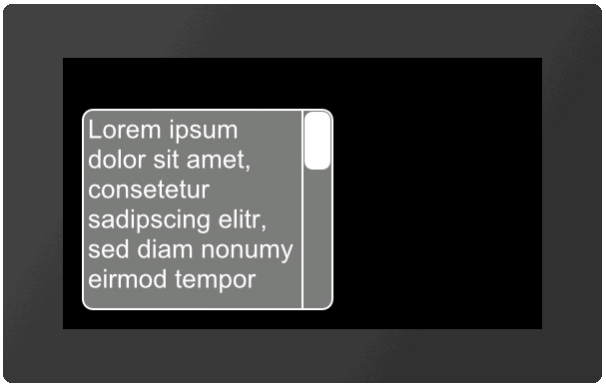
Anstelle der eingegebenen Zeichen wird das **Wildcardcode** angezeigt.




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

StringBox

In StringBoxen können größere Mengen an Text angezeigt werden. Es kann jederzeit zusätzlicher Text hinzugefügt aber auch gelöscht werden. Jeder neu hinzugefügte Text (#SBA, #SBF) wird als neuer Absatz eingefügt. Ist die AutoWrap (siehe #SBS) Funktion deaktiviert, so ist die Absatznummer gleich der Zeilennummer. Ansonsten können sich beide unterscheiden. Es gibt jedoch Kalkulationen um sie ineinander um zurechnen. Im nachfolgenden Beispiel wird eine StringBox erzeugt und ein Absatz hinzugefügt




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

StringBox platzieren (ab V1.3)

#SBP	Obj-ID, x, y, Anker, Breite, Höhe, Radius, ScrollbarBreite(Texthöhe)

Eine StringBox wird mit dem gegebenen **Anker** an die Position **x**, **y** mit definierter **Höhe** und **Breite** platziert. Optional kann noch die Breite der Scrollbar angegeben werden (**ScrollbarBreite**). Wird kein Wert angegeben, wird die Texthöhe als Breite verwendet. Damit die StringBox sichtbar ist, muss zwingend ein Style zugewiesen werden (siehe #SBS).

Style für StringBox definieren (ab V1.3)

#SBS	Obj-ID, DrawStyle-Nr. Hintergrund, DrawStyle-Nr. Scrollbar, TextStyle-Nr., RandX(0), RandY(0), AutoWrap(1)

Mit dem Befehl wird das Aussehen der StringBox festgelegt. Es werden zwei DrawStyles benötigt. Zum einen wird der Hintergrund der EditBox und zum anderen der Balken des Schiebereglers (**Scrollbar**) definiert.Der Aufbau ist im Unterkapitel DrawStyle näher beschrieben. Mit dem TextStyle wird das Aussehen der Zeichenkette bestimmt (**TextStyle-Nr.**). Im Unterkapitel TextStyle ist dies genauer erläutert. Mit den beiden optionalen Parametern (**RandX** und **RandY**) kann der Abstand des Textes zum Rand der Box angegeben werden. **AutoWrap** bestimmt den Zeilenumbruch:

AutoWrap	
0	Text wird am Ende der Zeile abgeschnitten
1	Automatischer Zeilenumbruch aktiv

Absatz in der StringBox hinzufügen (ab V1.3)

#SBA	Obj-ID, Absatz, "Text"; "Text (Zeile+1)"; "Text (Zeile+2)";....

Mit dem Befehl können weitere Zeilen zu der StringBox hinzugefügt werden. Der Parameter **Absatz** gibt die Position in der Box vor. Die erste Zeile hat die Nummer 1. Wird als Absatz 0 ausgewählt, so wird der Text am Ende hinzugefügt.


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

Absatz in der StringBox entfernen (ab V1.3)

#SBD	Obj-ID, Absatz, Zeile1. ...

Aus der StringBox werden einzelne oder mehrere Absätze entfernt. Wird 0 als Absatz übergeben, so werden alle Strings aus der StringBox entfernt und die Box ist leer. Zudem können auch Bereiche angegeben werden, z.B. 1-5.

Hinzufügen einer Textdatei in die StringBox (ab V1.3)

#SBF	Obj-ID, Absatz, <Textdatei>

Eine StringBox kann auch komplette Textdateien anzeigen. Dabei gibt der Parameter **<Textdatei>** den Pfad zur Datei an. Der Parameter **Absatz** gibt die Position in der Box vor. Die erste Zeile hat die Nummer 1. Wird als Absatz 0 ausgewählt, so wird der Text am Ende hinzugefügt.


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

In StringBox zu Zeile springen (ab V1.3)

#SBO	Obj-ID, Zeile, Zeit (0), Aktionskurve-Nr (0)

Der Inhalt der StringBox springt zur angegeben **Zeile**. Mit den optionalen Parametern kann der Sprung animiert werden. Der Parameter **Zeit** wird in 1/100s angegeben. Ist der Wert positiv, wird die Zeitdauer für den gesamten Scrollbereich verwendet. Die Geschwindigkeit ist somit konstant. Ein negativer Wert bestimmt dagegen die Zeit bis die neue Zeile erreicht ist. Somit ist die Geschwindigkeit abhängig von der zu scrollenden Zeilenanzahl. Die **Aktionskurve-Nr** bestimmt den zeitlichen Ablauf. Im Unterkapitel Aktionskurven und Aktionspfade ist dies näher erläutert.

Formatierte Zeichenkette

Formatierte Strings werden in Zeichenkettenausgaben verwendet. Das Format ist an die C-Funktion "printf" angelehnt. Die Funktion besteht aus einem Format-String und den konkret dazugehörigen Argumenten.Für die verschiedenen Datentypen werden folgende Platzhalter im Format-String verwendet:

Typ	Platzhalter	Beispiel
Festkommawert Dezimal	%d	42
Oktalwert	%o	645
Hexwert	%x, %X	7a, 7A
Float	%f,	299.57
Wissenschaftliche Notation	%e, %E	2.9957e+2, 2.9957E+2
Kürzeste Notation: Float oder Wissenschaftlich	%g, %G	299.57
Character	%c	a

Jeder Platzhalter kann genauer spezifiziert werden: Flags, Feldbreite und Genauigkeit - in dieser Reihenfolge:

Flag	Erklärung
-	Wert innerhalb Feldbreite linksbündig ausrichten. Rechtsbündig ist die Standardausgabe
+	Ausgabe von '+' bei positiven, '-' bei negativen Werten
(space)	Ausgabe von ' ' (space) bei positiven, '-' bei negativen Werten
#	Bei Hex- und Oktalausgabe wird das 0x, 0X bzw. 0 bei Werten ≠ 0. Bei Float und wissenschaftlicher Notation wird immer ein '.' ausgegeben - auch wenn nur noch 0 folgt. Default wird das Komma nur ausgegeben wenn Werte folgen
0	Innerhalb der Feldbreite wird Linksbündig nicht benötigter Platz mit 0 aufgefüllt.

Feldbreite	Erklärung
(number)	Mindestbreite des Feldes für die Ausgabe
*	Feldbreite wird aus der Argumentenliste übernommen. Wobei die Feldbreite direkt vor dem eigentlich Argument in der Liste steht.

Genauigkeit	Erklärung
.number	Für Integer: Minimum Anzahl an Stellen (default =1) Float: Minimum Anzahl nach dem Komma (default =6)
.*	Anzahl der Stellen wird aus der Argumentenliste übernommen. Wobei die Anzahl der Stellen direkt vor dem eigentlich Argument in der Liste steht.

Bilder #P

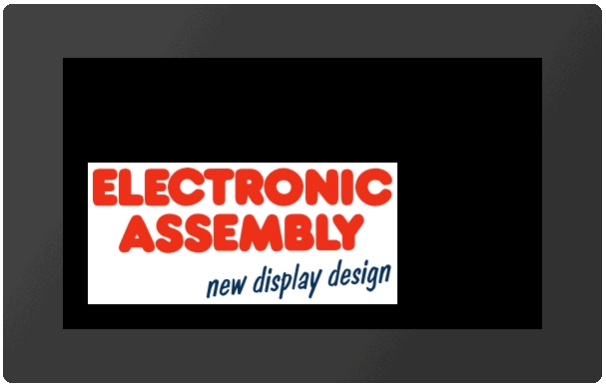
Befehlsgruppe um Bilder darzustellen. Mit der Designsoftware uniTFTDesigner können folgende Dateiformate/Grafikformate verwendet werden: png, bmp, jpg, jpeg, tga, gif, g16, svg, svgz. Der uniFTF Designer wandelt die Daten automatisch in das korrekte interne Format um. Wenn der uniTFTDesigner nicht verwendet wird, können diese Dateien mit dem Tool EAconvert.exe (im Verzeichnis \Simulator_and_Tools) konvertiert werden (→ evg, epg, epa)

Bild platzieren (Picture Place)	**#PPP**	Obj-ID, <Name>, y, y, Anker(1), Breite(0), Höhe (0), Winkel (0)
Bildanimation ändern (Picture change)	**#PPA**	Obj-ID, Animationstyp(0), Zeit, Bild-Nr.
Bild über die Schnittstelle laden und platzieren (Picture Interface Place)	**#PIP**	Obj-ID, Binäre Daten; x(0), y(y-Auflösung -1),Anker(1), Breite (0), Höhe(0), Winkel (0)

Bild platzieren

#PPP	Obj-ID, <Name>, y, y, Anker(1), Breite(0), Höhe (0), Winkel (0)

Mit dem Befehl wird das Bild (**<Name>**) mit dem gegebenen **Anker** an die Position **x**, **y** platziert. Wird **Breite**=0 und **Höhe**=0 übergeben wird die Originalgröße des Bildes übernommen. Ist nur einer der beiden Parameter 0 wird das Bild proportional auf den jeweilig anderen skaliert. Der weitere optionale Parameter **Winkel** gibt die Rotation des Bildes an. Wird eine Animation platziert, so wird diese zyklisch ausgeführt.




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

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

Bildanimation ändern

#PPA	Obj-ID, Animationstyp(0), Zeit, Bild-Nr.

Der Befehl ändert eine vorhandene Bildanimation. Die beiden Parameter **Zeit** und **Bild-Nr.** werden nur beachtet, wenn der **Animationstyp** 7 ist. Die Animation läuft dann in der vorgegeben Zeit (1/100s) bis zur **Bild-Nr** ab. Die Zeit zwischen den Bildern wird dabei neu berechnet. Folgende Animationstypen können ausgewählt werden:

Animationstyp	
1	Zyklisch
2	Zyklisch rückwärts
3	Ping Pong

4	Ping Pong rückwärts
5	Einmalig
6	Einmalig rückwärts
7	Goto

Bild über die Schnittstelle laden und platzieren

#PIP	Obj-ID, Binäre Daten; x(0), y(y-Auflösung -1),Anker(1), Breite (0), Höhe(0), Winkel (0)

Mit dem Befehl wird ein Bild angezeigt. Die Daten werden hierfür im *.epg bzw. *.evg Format binär über die Schnittstelle übertragen und analog zum Befehl #PPP platziert. (ab V1.3)

Touchfunktionen #T

Befehlsgruppe um Touchfunktionen zu ermöglichen. Es können einfache Taster und Schalter platziert werden, sowie Radiobuttons, Schieberegler, Bargraphen und Dreh-/Zeigerinstrumente.

Taster und Schalter definieren

Rechteckigen Taster platzieren (Touch Button Rectangle)	**#TBR**	Obj-ID, ButtonStyle-Nr., "Text normal"; "Text down"; x, y, Anker (5), Breite(ButtonStyle Breite), Höhe(ButtonStyle Höhe)
Rechteckigen Schalter platzieren (Touch Switch Rectangle)	**#TSR**	Obj-ID, ButtonStyle-Nr., "Text normal"; "Text down"; x, y, Anker (5), Breite(ButtonStyle Breite), Höhe(ButtonStyle Höhe)
Ellipsenförmigen Taster platzieren (Touch Button Ellipse)	**#TBE**	Obj-ID, ButtonStyle-Nr., "Text normal"; "Text down"; x, y, Anker (5), Breite(ButtonStyle Breite), Höhe(ButtonStyle Höhe)
Ellipsenförmigen Schalter platzieren (Touch Switch Ellipse)	**#TSE**	Obj-ID, ButtonStyle-Nr., "Text normal"; "Text down"; x, y, Anker (5), Breite(ButtonStyle Breite), Höhe(ButtonStyle Höhe)
Bild Taster platzieren (Touch Button Picture)	**#TBP**	Obj-ID, ButtonStyle-Nr., "Text normal"; "Text down"; x, y, Anker (5), Breite(ButtonStyle Breite), Höhe(ButtonStyle Höhe)
Bild Schalter platzieren (Touch Switch Picture)	**#TSP**	Obj-ID, ButtonStyle-Nr., "Text normal"; "Text down"; x, y, Anker (5), Breite(ButtonStyle Breite), Höhe(ButtonStyle Höhe)
Beschriftungsfreien Icon Taster platzieren (Touch Button Icon)	**#TBI**	Obj-ID, x, y, Anker, <Buttonname normal>, Breite normal (0), 'Buttonname down' (keine Änderung); Breite down (0), "SoundString"
Beschriftungsfreien Icon Schalter platzieren (Touch Switch Icon)	**#TSI**	Obj-ID, x, y, Anker, <Buttonname normal>, Breite normal (0), 'Buttonname down' (keine Änderung); Breite down (0), "SoundString"
Objekt in Taster umwandeln (Touch Button Object)	**#TBO**	Obj-ID, ButtonStyle-Nr., "Text normal"; "Text down";
Objekt in Schalter umwandeln (Touch Switch Object)	**#TSO**	Obj-ID, ButtonStyle-Nr., "Text normal"; "Text down";

Einstellungen von Tastern und Schaltern

Beschriftung von Taster/Schalter ändern (Touch Change Label)	**#TCL**	Obj-ID, "Text normal"; "Text down";
Zustand von Taster/Schalter ändern (Touch Change State)	**#TCS**	Zustand, Obj-ID1, ..., Obj-IDn
Zustand von Taster/Schalter abfragen (Touch Query State)	**#TQS**	Obj-ID1, ..., Obj-IDn
Taster/Schalter aktivieren/ deaktivieren (Touch Change Enable)	**#TCE**	Aktiv, Obj-ID1, ..., Obj-IDn
Rückmeldungen von Touchereignissen festlegen (Touch Change Response)	**#TCR**	Event, Filter, Obj-ID1, ..., Obj-IDn

Radiogroup

Taster/Schalter zu Radiogroup hinzufügen (Touch Radiougroup Add)	**#TRA**	Group-ID, Obj-ID1, ..., Obj-IDn
Zustand der Radiogroup abfragen	**#TQR**	Group-ID1, ..., Group-IDn

(Touch Query Radiougroup)		

Spezielle Touchfunktion

Interne Verarbeitung von Touchfunktionen (Touch Id Define)	#TID	Maske, Obj-ID1, ..., Obj-IDn
Freien Touchbereich platzieren (Touch Area Free)	#TAF	Obj-ID, x, y, Anker, Breite, Höhe
Einstellen von Gestenzeiten (Touch Configure Gesture)	#TCG	DoubleClick Zeit (30), LongClick Zeit (100)

Taster und Schalter definieren

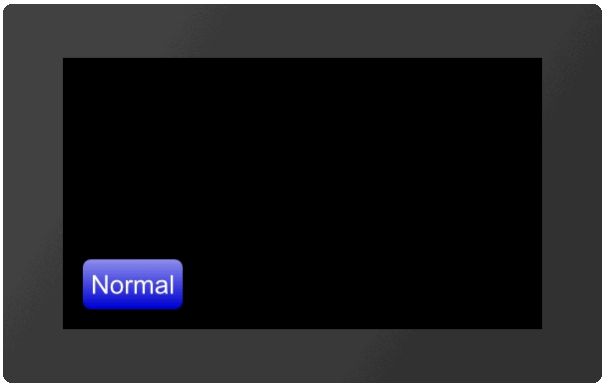
Taster und Schalter können auf verschiedene Events (Down, Up, Drag, DoubleClick, LongClick) reagieren. Es gibt zwei Möglichkeiten die Zustandsänderungen von Tastern und Schaltern auszuwerten:

- **Änderungen werden in den Sendepuffer gestellt:**
  Mit dem Befehl #TCR kann angegeben werden, welche Rückmeldungen in den Sendepuffer gestellt werden sollen (kein DoubleClick und LongClick)
- **Bei Änderungen wird ein Makro ausgeführt:**
  Mit den Befehlen #MDT und #MDG können Makros mit dem Taster/Schalter verbunden werden. Bei der jeweiligen Zustandsänderung wird das dazugehörige Makro aufgerufen.

Rechteckigen Taster/Schalter platzieren

#TBR #TSR	Obj-ID, ButtonStyle-Nr., "Text normal"; "Text down"; x, y, Anker (5), Breite(ButtonStyle Breite), Höhe(ButtonStyle Höhe)

Mit dem Befehl wird ein rechteckiger Taster/Schalter mit dem gegebenen **Anker** an die Position **x**, **y** platziert. Der Parameter **"Text normal"** gibt die Ausgabe im ungedrückten, **"Text down"** im gedrückten Zustand an. Mit dem ButtonStyle wird das Aussehen des Tasters/Schalters bestimmt (**ButtonStyle-Nr.**). Im Unterkapitel ButtonStyle ist dies genauer erläutert. Die **Breite** und **Höhe** des Tasters/Schalters wird aus dem ButtonStyle übernommen, kann aber optional überschrieben werden.



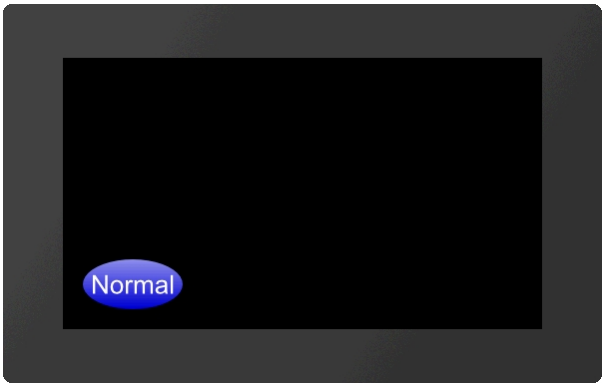

```
...
#TBR 1,1,"Normal";"Pressed";20,20,7
...
```

Ellipsenförmigen Taster/Schalter platzieren

#TBE	Obj-ID, ButtonStyle-Nr., "Text normal"; "Text down"; x, y, Anker (5), Breite(ButtonStyle Breite), Höhe(ButtonStyle Höhe)

#TSE	

Mit dem Befehl wird ein ellipsenförmiger Taster/Schalter mit dem gegebenen **Anker** an die Position **x**, **y** platziert. Der Parameter **"Text normal"** gibt die Ausgabe im ungedrückten, **"Text down"** im gedrückten Zustand an. Mit dem ButtonStyle wird das Aussehen des Tasters/Schalters bestimmt (**ButtonStyle-Nr.**). Im Unterkapitel ButtonStyle ist dies genauer erläutert. Die **Breite** (∅ X) und **Höhe** (∅ Y) des Tasters/Schalters wird aus dem ButtonStyle übernommen, kann aber optional überschrieben werden.

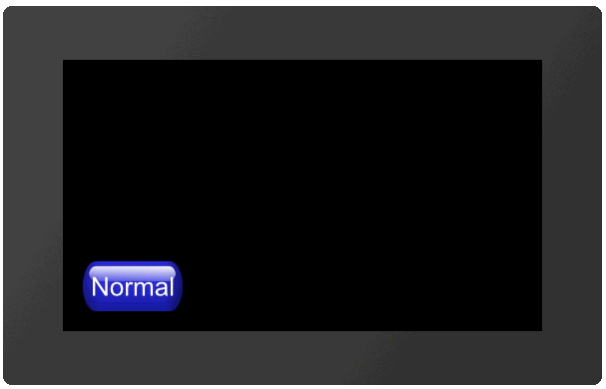



```
...
#TBE 1,1,"Normal";"Pressed";20,20,7
...
```

Bild Taster/Schalter platzieren

#TBP #TSP	Obj-ID, ButtonStyle-Nr., "Text normal"; "Text down"; x, y, Anker (5), Breite(ButtonStyle Breite), Höhe(ButtonStyle Höhe)

Mit dem Befehl wird ein Taster/Schalter als Bild mit dem gegebenen **Anker** an die Position **x**, **y** platziert. Der Parameter **"Text normal"** gibt die Ausgabe im ungedrückten, **"Text down"** im gedrückten Zustand an. Mit dem ButtonStyle wird das Aussehen des Tasters/Schalters bestimmt (**ButtonStyle-Nr.**). Im Unterkapitel ButtonStyle ist dies genauer erläutert. Die **Breite** und **Höhe** des Tasters/Schalters wird aus dem ButtonStyle übernommen, kann aber optional überschrieben werden.




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

Beschriftungsfreien Icon Taster/Schalter platzieren

#TBI #TSI	Obj-ID, x, y, Anker, <Buttonname normal>, Breite normal (0), 'Buttonname down' (keine Änderung); Breite down (0), "SoundString"

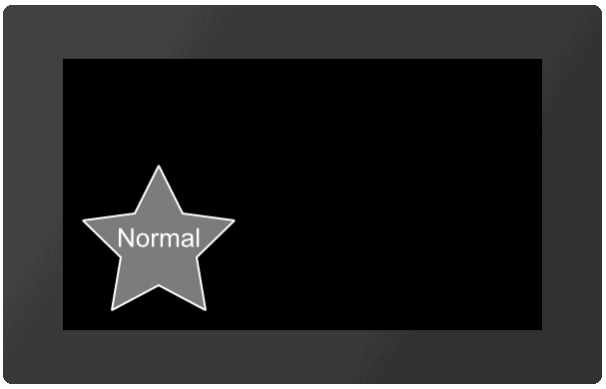
Mit dem Befehl wird ein Taster/Schalter als Icon mit dem gegebenen **Anker** an die Position **x**, **y** platziert. Ein ButtonStyle ist hierfür nicht notwendig. Die beiden Parameter **'Buttonname normal'** und **'Buttonname down'** geben die anzuzeigenden Bilder an. Wird keine **Breite** (in Pixel) oder null angegeben, so wird die Originalgröße des Bildes verwendet. Die Höhe wird intern berechnet (proportional). Der letzte Parameter **"SoundString"** gibt die Töne an, welche bei Touchbetätigung abgespielt werden.

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

Objekt in Taster/Schalter umwandeln

#TBO #TSO	Obj-ID, ButtonStyle-Nr., "Text normal"; "Text down";

Ein beliebiges bestehendes Objekt wird in einen Taster/Schalter gewandelt. Aus dem ButtonStyle werden Zusätzliche Informationen wie z.B. Soundname entnommen. Handelt es sich bei dem zu wandelten Objekt um eine grafische Primitive (z.B. Polygon) mit dem selben DrawSytle wie im ButtonStyle, wird für den gedrückten Zustand automatisch der DraswStyle des ButtonStyles übernommen.




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

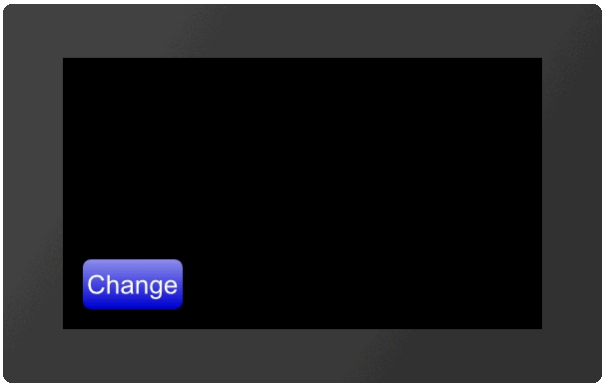
Einstellungen von Tastern und Schaltern

In dieser Gruppe sind Befehle für die Einstellung von Tastern und Schaltern zusammengefasst.

Beschriftung von Taster/Schalter ändern

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

Der Befehl ändert die Beschriftung von Touchobjekten. Wird für den gedrückten Zustand (**"Text down"**) kein Text angegeben, so wird auch hierfür **"Text normal"** verwendet.



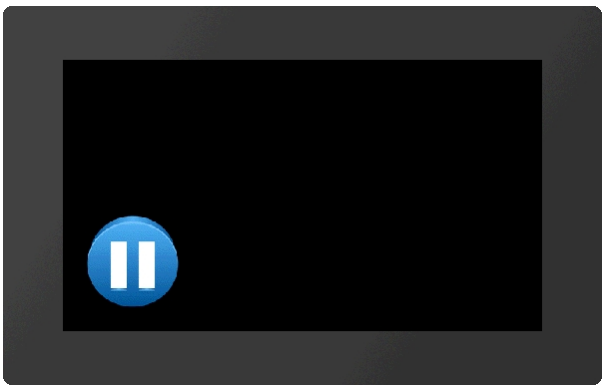

```
...
#TBR 1,...
#TCL 1,"Change";
...
```

Zustand von Taster/Schalter ändern

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

Mit dem Befehl wird der Zustand der Touchobjekte (**Obj-ID1, ..., Obj-IDn**) geändert:

Zustand	
1	ungedrückt
2	gedrückt



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

Zustand von Taster/Schalter abfragen

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

Der Zustand der Touchobjekte (**Obj-ID1, ..., Obj-IDn**) wird in den Sendepuffer gestellt. Die Rückmeldung ist folgendermaßen aufgebaut:

#	T	Q	S	Obj-ID	Zustand	...
$1B	$54	$51	$53	16-Bit Wert	16-Bit Wert	

```
1Bh 54h 51h 53 h 01h 00h 01h 00h
```

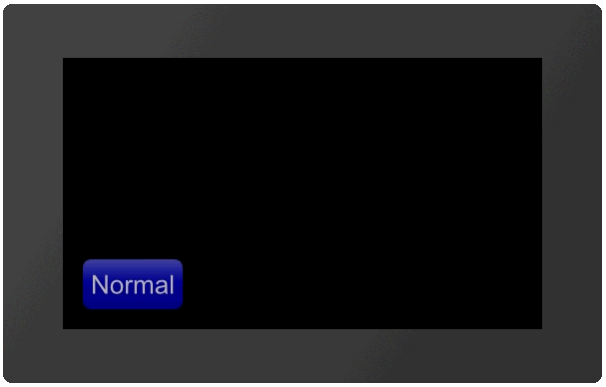
```
...
#TBR 1,...
#TQS 1
...
```

Taster/Schalter aktivieren/ deaktivieren

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

Mit dem Befehl werden Touchobjekte (**Obj-ID1, ..., Obj-IDn**) aktiviert bzw. deaktiviert (**Aktiv**):

Aktiv	
0	nicht aktiv
1	aktiv




```
...
#TBR 1,...
#TCE 0,1
...
```

Rückmeldungen von Touchereignissen festlegen

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

Jedem Touchobjekt kann zugeordnet werden ob und welche Rückmeldungen in den Sendepuffer gelangen. Es werden drei Events unterschieden: Up, down und drag. Für jedes dieser Events kann einzeln eingestellt werden, ob es eine Rückmeldung auslöst oder nicht. Dies kann mit dem Parameter **Event** eingestellt werden. Dieser ist gemäß nachfolgender Tabelle bitcodiert:

	Event							
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Up		●		●		●		●
Down			●	●			●	●

Drag					●	●	●	●

Möchte man demnach z.B nur für Up und Down Events Rückmeldungen erhalten, so stellt man den Parameter Event auf 3. Zudem kann die Rückmeldung auch davon abhängig gemacht werden, ob für das Event ein Makro definiert ist (**Filter**):

Filter	
0	nur senden wenn kein Makro definiert ist
1	immer senden

Bei der Definition von Tastern wird automatisch der gedrückte Zustand gesendet wenn kein Makro definiert ist (= `#TCR 2,0,Obj-ID`). Bei Schaltern beide Zustände (= `#TCR 3,0,Obj-ID`). Bei Bargraphen und Instrumenten das Loslassen (= `#TCR 1,0,Obj-ID`). Eine EditBox sendet den Inhalt beim Deaktivieren (=`#TCR 1,0,Obj-ID`). Sollen keine Ereignisse gesendet werden kann dies mit `#TCR 0,0,Obj-ID` eingestellt werden.

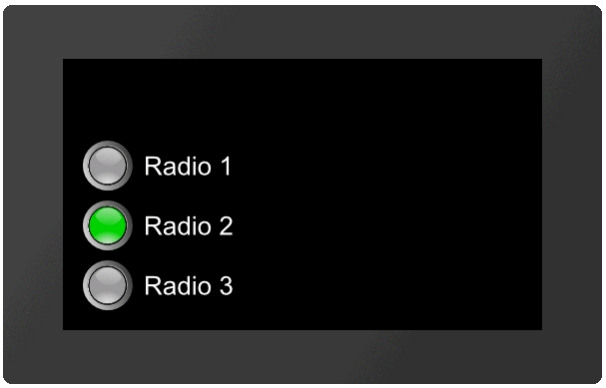
Radiogroup

Diese Gruppe umfasst Befehle zur Erstellung und Verwaltung von Radiogroups.

Taster/Schalter zu Radiogroup hinzufügen

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

Einer bestehenden oder neuen Radiogruppe (**Group-ID**) werden eine oder mehrere Schalter (**Obj-ID, ..., Obj-IDn**) hinzugefügt.




```
...
#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
...
```

Zustand der Radiogroup abfragen

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

Der aktive Schalter der Radiogruppe (**Group-ID**) wird in den Sendepuffer gestellt. Die Rückmeldung ist folgendermaßen aufgebaut:

ESC	T	Q	R	Obj-ID	Group-ID	...
$1B	$54	$51	$52	16-Bit Wert	16-Bit Wert	

```
1Bh 54h 51h 52h 03h 00h 04h 00h
```

```
...
#TRA 4,...
#TQR 4
...
```

Spezielle Touchfunktion

Diese Gruppe enthält weitere spezielle Touchfunktionen.

Interne Verarbeitung von Touchfunktionen

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

Jedem Objekt (Obj-ID) kann eine spezielle Touchaktion zugewiesen, bzw. Toucheingabe ermöglicht werden. Die einzelnen Bits von **Maske** können mit Bitveroderung zusammengefasst werden, sodass mehrere Touchaktion gleichzeitig möglich sind:

Maske	
1	Interne Verarbeitung (z.B. Bargraph/ Instrument/ EditBox)
2	Objekt frei bewegen
4	Objekt proportional vergrößern/verkleinern
8	Um den aktiven Anker drehen
16	Größenänderung mit zwei Fingern
32	Rotieren mit zwei Fingern
128	Objekt bleibt unverändert/ Touchmakros werden ausgeführt

Freien Touchbereich platzieren (ab V1.4)

#TAF	Obj-ID, x, y, Anker, Breite, Höhe

Mit dem Befehl wird ein freier Touchbereich mit gegebenen **Anker**, **Breite** und **Höhe** an die Position **x**, **y,** platziert.

Einstellen von Gestenzeiten (ab V1.4)

#TCG	DoubleClick Zeit, LongClick Zeit

Mit dem Befehl wird die Zeitschwelle von Gesten eingestellt. Die **DoubleClick Zeit** gibt in 1/100s an, wie viel Zeit maximal zwischen zwei Down Events verstreichen darf, sodass noch ein gültiger Doppelklick erkannt wird. Mit dem Parameter **LongClick Zeit** wird festgelegt, welche Zeitspanne (in 1/100s) mindestens verstreichen muss, damit ein LongClick detektiert wird.
Der gültige Wertebereich für Doppelcklick ist 20 (=200 ms) bis 100 (=1 sec.), beim LongClick ist er bei 30 (=300 ms)

bis 1000 (=10 sec.).

Zeichnen / grafische Primitive #G

Befehlsgruppe um geometrische Formen und Linien darzustellen.

Rechteck platzieren (Graphic Rounded Rectangle)	**#GRR**	Obj-ID, DrawStyle-Nr, x, y, Anker, Breite, Höhe(=Breite), Radius (0), Rahmendicke(0), Winkel(0)
n-Eck platzieren (Graphic Geometric Polygon)	**#GGP**	Obj-ID, DrawStyle-Nr, x, y, Anker, Radius, Ecken, Rahmendicke(0), Winkel(0)
Stern platzieren (Graphic Geometric Star)	**#GGS**	Obj-ID, DrawStyle-Nr, x, y, Anker, Radius1, Radius2, Spitzen, Rahmendicke(0), Winkel(0)
Kreis/Ellipse platzieren (Graphic Ellipse Total)	**#GET**	Obj-ID, DrawStyle-Nr, x, y, Anker, RadiusX, RadiusY(=RadiusX), Rahmendicke(0), Winkel(0)
Kreissektor/Kuchenstück platzieren Graphic Ellipse Pie)	**#GEP**	Obj-ID, DrawStyle-Nr, x, y, Anker, RadiusX, RadiusY, StartWinkel, EndWinkel, Winkel(0)
Kreissegment platzieren (Graphic Ellipse Segment)	**#GES**	Obj-ID, DrawStyle-Nr, x, y, Anker, RadiusX, RadiusY, StartWinkel, EndWinkel, Winkel(0)
Kreisbogen platzieren (Graphic Ellipse Arc)	**#GEA**	Obj-ID, DrawStyle-Nr, x, y, Anker, RadiusX, RadiusY, StartWinkel, EndWinkel, Border(0), Winkel(0)
Polylinie platzieren (Graphic Polygon Line)	**#GPL**	Obj-ID, DrawStyle-Nr, x1, y1, x2, y2, ... xn, yn
Polygon platzieren (Graphic Polygon Fill)	**#GPF**	Obj-ID, DrawStyle-Nr, x1, y1, x2, y2, ... x4, y4
Punkte zu Polyline hinzufügen (Graphic Polyline Add)	**#GPA**	Obj-ID, x1, y1, x2, y2, ... xn, yn

Geometrische Figuren

Rechteck platzieren

#GRR	Obj-ID, DrawStyle-Nr, x, y, Anker, Breite, Höhe(=Breite), Radius (0), Rahmendicke(0), Winkel(0)

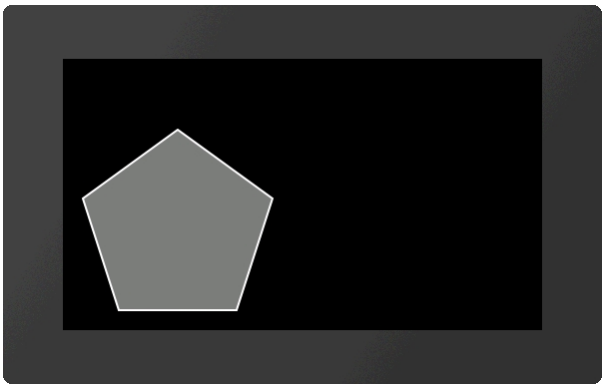
Eine Rechteck wird mit dem **Anker** und der **Breite** an die Position **x**, **y** platziert. Mit dem DrawStyle wird das Aussehen des Rechtecks bestimmt (**DrawStyle-Nr.**). Im Unterkapitel DrawStyle ist dies genauer erläutert. Ist keine **Höhe** angegeben, wird sie auf die Breite gesetzt (Quadrat). Optional kann ein Radius angegeben werden. Dieser rundet die Ecken ab. Zudem ist es möglich eine **Rahmendicke** zu bestimmen. Auch eine Rotation um den Anker (**Winkel**) kann eingestellt werden.

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

n-Eck platzieren

#GGP	Obj-ID, DrawStyle-Nr, x, y, Anker, Radius, Ecken, Rahmendicke(0), Winkel(0)

Ein regelmäßiges Vieleck wird mit dem **Anker** und der gegebenen Anzahl an **Ecken** an die Position **x**, **y** platziert. Mit dem DrawStyle wird das Aussehen des n-Ecks bestimmt (**DrawStyle-Nr.**). Im Unterkapitel DrawStyle ist dies genauer erläutert. Der **Radius** gibt die Größe der Figur vor. Zudem ist es möglich eine **Rahmendicke** zu bestimmen. Auch eine Rotation um den Anker (**Winkel**) kann eingestellt werden. Bei **Anker** =0 wird der Konstruktionspunkt verwendet.

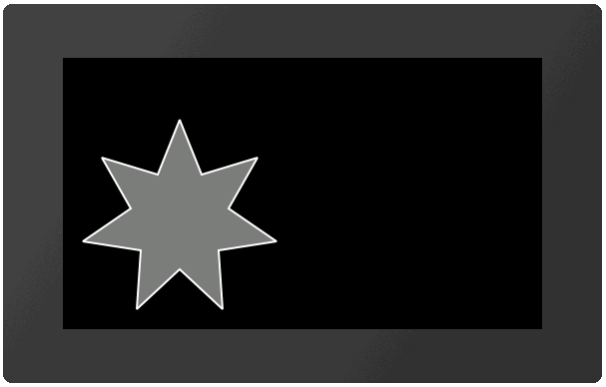


```
...
#GGP 1,1,20,20,7,100,5
...
```

Stern platzieren

#GGS	Obj-ID, DrawStyle-Nr, x, y, Anker, Radius1, Radius2, Spitzen, Rahmendicke(0), Winkel(0)

Ein Stern wird mit dem **Anker** an die Position **x**, **y** platziert. Mit dem DrawStyle wird das Aussehen des Sterns bestimmt (**DrawStyle-Nr.**). Im Unterkapitel DrawStyle ist dies genauer erläutert. Die erste Spitze wird oberhalb des Mittelpunktes auf **Radius1** gesetzt. Dann erfolgt die Verbindung zu **Radius2** dann zurück zu Radius1 usw. bis die Anzahl **Spitzen** erreicht ist. Zudem ist es möglich eine **Rahmendicke** zu bestimmen. Auch eine Rotation um den Anker (**Winkel**) kann eingestellt werden. Bei **Anker** =0 wird der Konstruktionspunkt verwendet.

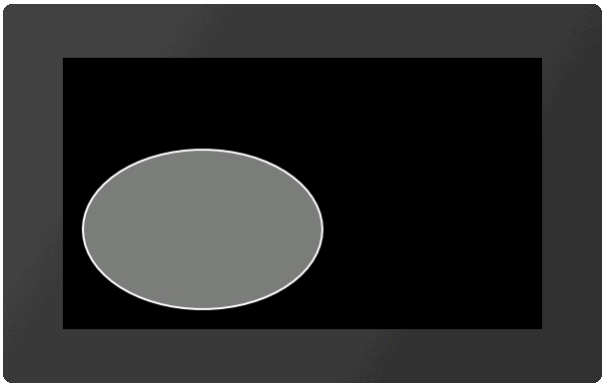


```
...
#GGS 1,1,20,20,7,100,50,7
...
```

Kreis/Ellipse platzieren

#GET	Obj-ID, DrawStyle-Nr, x, y, Anker, RadiusX, RadiusY(=RadiusX), Rahmendicke(0), Winkel(0)

Eine Ellipse wird mit dem **Anker** und dem **RadiusX** an die Position **x**, **y** platziert. Mit dem DrawStyle wird das Aussehen des Kreises bestimmt (**DrawStyle-Nr.**). Im Unterkapitel DrawStyle ist dies genauer erläutert. Ist kein **RadiusY** angegeben, wird er auf RadiusX gesetzt (Kreis). Zudem ist es möglich eine **Rahmendicke** zu bestimmen. Auch eine Rotation um den Anker (**Winkel**) kann eingestellt werden.

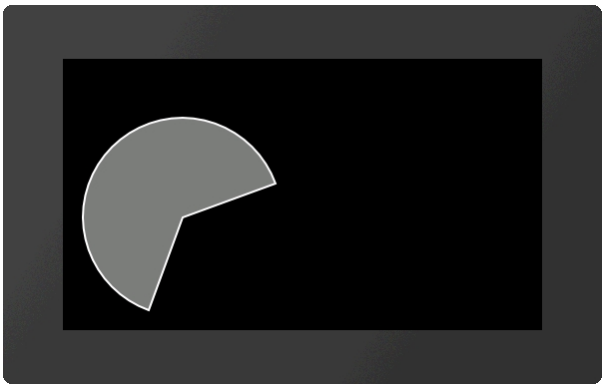


```
...
#GET 1,1,20,20,7,120,80
...
```

Kreissektor/Kuchenstück platzieren

#GEP	Obj-ID, DrawStyle-Nr, x, y, Anker, RadiusX, RadiusY, StartWinkel, EndWinkel, Winkel(0)

Ein Kreissektor/Kuchenstück wird mit dem **Anker**, dem **RadiusX** und dem **RadiusY** an die Position **x**, **y** platziert. **Start/EndWinkel** geben die Größe des Stücks an. Mit dem DrawStyle wird das Aussehen des Kresisektors bestimmt (**DrawStyle-Nr.**). Im Unterkapitel DrawStyle ist dies genauer erläutert. Ist kein **RadiusY** angegeben, wird er auf RadiusX gesetzt (Kreis). Auch eine Rotation um den Anker (**Winkel**) kann eingestellt werden.



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

Kreissegment platzieren

#GES	Obj-ID, DrawStyle-Nr, x, y, Anker, RadiusX, RadiusY, StartWinkel, EndWinkel, Winkel(0)

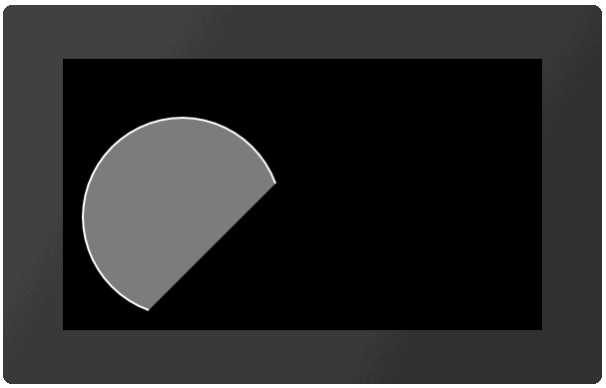
Ein Kreissegment wird mit dem **Anker**, dem **RadiusX** und dem **RadiusY** an die Position **x**, **y** platziert. **Start/EndWinkel** geben die Größe des Stücks an. Mit dem DrawStyle wird das Aussehen des Kreissegments bestimmt (**DrawStyle-Nr.**). Im Unterkapitel DrawStyle ist dies genauer erläutert. Ist kein **RadiusY** angegeben, wird er auf RadiusX gesetzt (Kreis). Auch eine Rotation um den Anker (**Winkel**) kann eingestellt werden.

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

Kreisbogen platzieren

#GEA	Obj-ID, DrawStyle-Nr, x, y, Anker, RadiusX, RadiusY, StartWinkel, EndWinkel, Border(0), Winkel(0)

Ein Kreisbogen wird mit dem **Anker**, dem **RadiusX** und dem **RadiusY** an die Position **x**, **y** platziert. **Start/EndWinkel** geben die Größe des Stücks an. Mit dem DrawStyle wird das Aussehen des Kreisbogens bestimmt (**DrawStyle-Nr.**). Im Unterkapitel DrawStyle ist dies genauer erläutert. Ist kein **RadiusY** angegeben, wird er auf RadiusX gesetzt (Kreis). Zudem ist es möglich eine **Rahmendicke** zu bestimmen. Auch eine Rotation um den Anker (**Winkel**) kann eingestellt werden.



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

Polilinie platzieren

#GPL	Obj-ID, DrawStyle-Nr, x1, y1, x2, y2, ... xn, yn

Eine Polyline wird mit den Koordinaten **[x1, y1],[x2, y2],...,[xn, yn**] gezeichnet. Mit dem DrawStyle wird das Aussehen der Polyline bestimmt (**DrawStyle-Nr.**). Im Unterkapitel DrawStyle ist dies genauer erläutert.

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

Polygon platzieren

#GPF	Obj-ID, DrawStyle-Nr, x1, y1, x2, y2, ... x4, y4

Ein gefülltes Polygon wird mit den Koordinaten **[x1, y1],[x2, y2],...,[x4, y4**] gezeichnet. Mit dem DrawStyle wird das Aussehen des Polygons bestimmt (**DrawStyle-Nr.**). Im Unterkapitel DrawStyle ist dies genauer erläutert. Vom letzten gegeben Punkt wird automatisch eine Verbindung zum ersten Punkt gezeichnet und so das Polygon geschlossen. Es sind nur 2-4 Punkte erlaubt, also Linien, Dreieck und Viereck.

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

Punkte zu Polyline hinzufügen

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

Der Befehl fügt am Ende einer Polyline Koordinaten **[x1, y1],[x2, y2],...,[xn, yn**] hinzu. Beim Polygon wird die Figur automatisch geschlossen.

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

Bargraphen / Instrumente #I

Befehlsgruppe um Bargraphen, Schieberegler und Dreh-/Zeigerinstrumente darzustellen.

Bargraph

Rechteckigen Bargraph platzieren (Instrument Bar Rectangle)	**#IBR**	Obj-ID, DrawStyle-Füllung, DrawStyle-Hintergrund, x, y, Anker, Breite, Höhe, Radius(0), Startwert(0), Endwert(100), Richtung (1), Winkel(0)
Dreieckigen Bargraph platzieren (Instrument Bar Triangle)	**#IBT**	Obj-ID, DrawStyle-Füllung, DrawStyle-Hintergrund, x, y, Anker, Breite, Höhe, Spitze(0), Startwert(0), Endwert(100), Richtung(1), Winkel(0)
Gebogenen Bargraph platzieren (Instrument Bar Arc)	**#IBA**	Obj-ID, DrawStyle-Füllung, DrawStyle-Hintergrund, x, y, Anker, Radius, Dicke, Startwinkel, Endwinkel, Startwert(0), Endwert(100), Richtung (1)

Schieberegler

Schieberegler definieren (Instrument Group Slider)	**#IGS**	Group-ID, Path-ID, Regler-ID, Tangential(0), Startwert(0), Endwert(100)

Dreh-/Zeigerinstrumente

Zeigerinstrument aus Objekten definieren (Instrument Group Meter)	**#IGM**	Group-ID, Indicator-ID, StartWinkel, DeltaWinkel, Startwert(0), Endwert(100)
Instrument aus eDIP Serie platzieren (Instrument Picture Place)	**#IPP**	Object-ID, 'InstrumentName'; x, y, Anker, Breite(0), Höhe(0), Startwert(0), Endwert(100), Winkel(0)

Einstellungen Bargraph / Instrumente

Bargraph/Instrument Wert setzen (Instrument Value Set)	**#IVS**	Obj-ID, Wert, Zeit(0), Aktionskurve-Nr.(0)
Start/Endwert eines Bargraph/Instrument ändern (Instrument Value New)	**#IVN**	Obj-ID, Startwert, Endwert(keine Änderung)
Bargraph/Instrument Wert auf Kalkulationswert mit AutoUpdate setzen (Instrument Value Autochange)	**#IVA**	Obj-ID, Kalkulation, Zeit(0), Aktionskurve-Nr.(0)
Bargraph/Instrument Kalkulation für AutoUpdate ändern (Instrument Value Calulation)	**#IVC**	Obj-ID, Kalkulation

Bargraph

Bargraphen können sowohl zur Anzeige wie auch zur Eingabe von Werten verwendet werden. Nach der Definition (#IBR, #IBT, #IBA) ist der Bargraph weder per Touch bedienbar noch zeigt er einen vordefinierten Wert an. Um ihn für Toucheingaben zu aktivieren benötigt man den Befehl #TID. Mit dem Befehl #IVS können Werte eingestellt werden. Die Funktion #IVA wird benötigt, wenn sich der Bargraph bei Änderung eines Kalkulationswertes (z.B. Analogeingang, Registerwert, ..) automatisch ändern soll. Im Folgenden Beispiel wird eine rechteckiger Bargraph platziert, auf den Wert 30 vorbelegt und für die Touchbedienung aktiviert.

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

Rechteckigen Bargraph platzieren

#IBR	Obj-ID, DrawStyle-Füllung, DrawStyle-Hintergrund, x, y, Anker, Breite, Höhe, Radius(0), Startwert(0), Endwert(100), Richtung (1), Winkel(0)

Eine rechteckiger Bargraph wird mit dem **Anker** der **Breite** und der **Höhe** an die Position **x**, **y** platziert. Aus dem **DrawStyle-Füllung** wird die Füllfarbe übernommen und das Band gezeichnet. Der **DrawStyle-Hintergund** gibt die Hintergrund- und Rahmenfarbe vor Im Unterkapitel DrawStyle ist der Aufbau genauer erläutert. Optional kann ein Radius angegeben werden. Dieser rundet die Ecken ab. Der **Startwert** und **Endwert** bestimmen die beiden Grenzen des Bargraphen. Die Laufrichtung wird durch **Richtung** bestimmt:

Richtung	
0	Von rechts nach links
1	Von links nach rechts

Auch eine Rotation um den Anker (**Winkel**) kann eingestellt werden.

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

Dreieckigen Bargraph platzieren

#IBT	Obj-ID, DrawStyle-Füllung, DrawStyle-Hintergrund, x, y, Anker, Breite, Höhe, Spitze(0), Startwert(0), Endwert(100), Richtung(1), Winkel(0)

Eine dreieckiger Bargraph wird mit dem **Anker** der **Breite** und der **Höhe** an die Position **x**, **y** platziert. Aus dem **DrawStyle-Füllung** wird die Füllfarbe übernommen und das Band gezeichnet. Der **DrawStyle-Hintergund** gibt die

Hintergrund- und Rahmenfarbe vor Im Unterkapitel DrawStyle ist der Aufbau genauer erläutert. Die Spitze befindet sich Links. Der optionale Parameter **Spitze** gibt die Lage der Spitze an:

Spitze	
0	Unten
1	Oben
2	Mitte

Der **Startwert** und **Endwert** bestimmen die beiden Grenzen des Bargraphen. Die Laufrichtung wird durch **Richtung** bestimmt:

Richtung	
0	Zur Spitze
1	Von der Spitze Weg

Auch eine Rotation um den Anker (**Winkel**) kann eingestellt werden.

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

Gebogenen Bargraph platzieren

#IBA	Obj-ID, DrawStyle-Füllung, DrawStyle-Hintergrund, x, y, Anker, Radius, Dicke, Startwinkel, Endwinkel, Startwert(0), Endwert(100), Richtung (1)

Ein gebogener Bargraph wird mit dem **Anker** und gegebener **Dicke** an die Position **x**, **y** platziert. Die Größe wird durch die Parameter **Radius**, **Startwinkel** und **Endwinkel** festgelegt. Aus dem **DrawStyle-Füllung** wird die Füllfarbe übernommen und das Band gezeichnet. Der **DrawStyle-Hintergund** gibt die Hintergrund- und Rahmenfarbe vor Im Unterkapitel DrawStyle ist der Aufbau genauer erläutert. Der **Startwert** und **Endwert** bestimmen die beiden Grenzen des Bargraphen. Die Laufrichtung wird durch **Richtung** bestimmt:

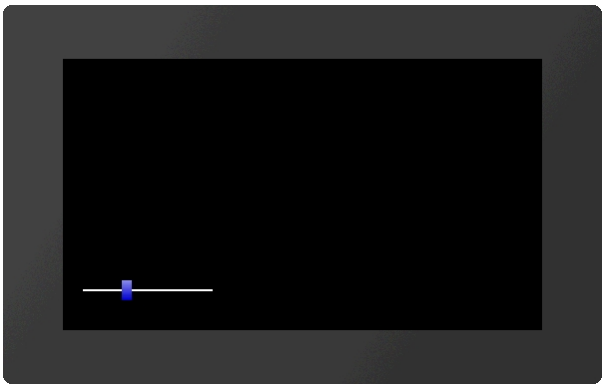
Richtung	
0	Gegen den Uhrzeigersinn
1	Im Uhrzeigersinn

Auch eine Rotation um den Anker (**Winkel**) kann eingestellt werden.

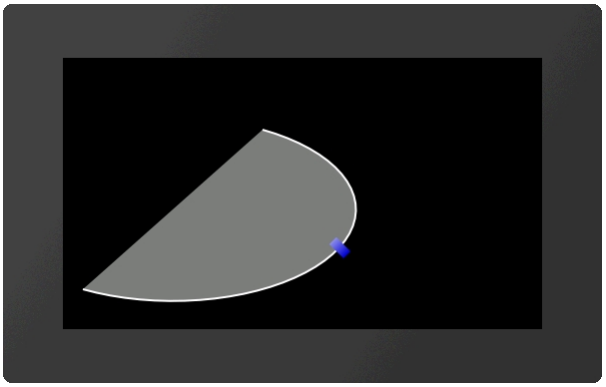
```
...
#IBA 1,2,1,20,20,7,100,40,0,180
...
```

Schieberegler

Ein Schieberegler besteht aus einem Pfad #GPL, #GPP) und einem Regler (z.B. #GRR, #PPP, ...). Beide Objekte müssen vorab definiert werden und zu einer Gruppe zusammengefasst werden (#OGA). Die Startposition des Reglers (Wert 0) fällt mit dem Konstruktionspunkt des Pfades zusammen. Dennoch ist es sinnvoll den Regler schon an die richtige Stelle zu positionieren, da sich die Gruppenbegrenzung (Bounding Box) nicht automatisch anpasst.
Um den Schieberegler für Toucheingaben zu aktivieren benötigt man den Befehl #TID. Mit dem Befehl #IVS können Werte eingestellt werden.



```
...
#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
...
```

Schieberegler definieren

#IGS	Group-ID, Path-ID, Regler-ID, Tangential(0), Startwert(0), Endwert (100)

Der Befehl wandelt eine bestehende Gruppe **Group-ID** in einen Schieberegler um. Die Gruppe muss mindestens zwei bestehende Objekte beinhalten:

- Einen Pfad (#GPL, #GPP) **Path-ID**
- Einen Regler (z.B. #GRR, #PPP, ...) **Regler-ID**

Der Regler wandert den Pfad entlang:

Tangential	
0	Regler dreht sich nicht mit
1	Regler dreht sich tangential zum Pfad

Der **Startwert** und **Endwert** bestimmen die beiden Grenzen des Schiebereglers.

Dreh-/Zeigerinstrumente

Ein Dreh-/Zeigersinstrument besteht aus einem Hintergrund (Skala) und einem Zeiger. Beide Objekte müssen vorab definiert werden und zu einer Gruppe zusammengefasst werden (#OGA). Bei der Konstruktion des Zeigers ist darauf zu achten, dass er in einer 90° Lage positioniert werden muss (nach oben). Um das Instrument für Toucheingaben zu aktivieren benötigt man den Befehl #TID. Mit dem Befehl #IVS können Werte eingestellt werden.


```
...
#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
#TID 1,3
...
```

Alternativ können auch Instrumente aus der eDIP Serie direkt platziert werden (siehe #IPP)

Zeigerinstrument aus Objekten definieren

#IGM	Group-ID, Indicator-ID, StartWinkel, DeltaWinkel, Startwert(0), Endwert(100)

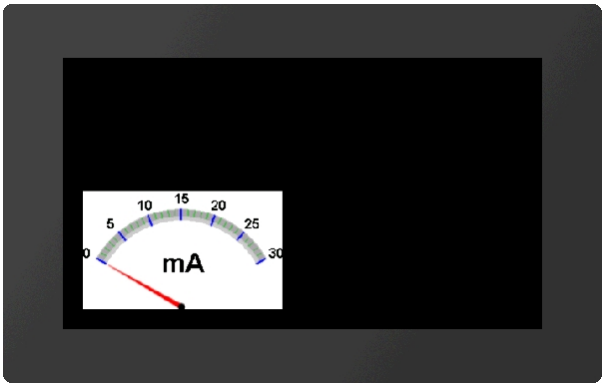
Der Befehl wandelt eine bestehende Gruppe **Group-ID** in ein Zeigerinstrument um. Der Parameter **Indicator-ID** bestimmt den Zeiger. Der Zeiger muss so positioniert sein, dass er zu Begin auf den Wert 0 zeigt. Der Parameter **StartWinkel** gibt den Startwinkel der Skala an. Der **DeltaWinkel** bestimmt die Drehrichtung (positiv: gegen den Uhrzeigersinn; negativ: im Uhrzeigersinn) und den Gesamtdrehwinkel (vom StartWinkel aus). Optional kann noch der **Start**- und **Endwert** (Ein- und Ausgabewerte) angegeben werden.

Instrument aus eDIP Serie platzieren

#IPP	Object-ID, 'InstrumentName'; x, y, Anker, Breite(0), Höhe(0), Startwert(0), Endwert(100), Winkel(0)

Dieser Befehl ist aus Kompatibilitätsgründen vorhanden. Fertige Instrumente aus der eDIP-Serie, welche mit den LCDTools erstellt wurden können direkt angezeigt werden. Das Instrument muss vorab mit Hilfe der EAconvert.exe in

das *.epi Format gewandelt werden. Die Größe wird über die Parameter **Breite** und **Höhe** festgelegt. Wird **Breite** =0 und **Höhe** =0 übergeben wird die Originalgröße des Instruments übernommen. Ist nur einer der beiden Parameter 0 wird das Instrument proportional auf den jeweilig anderen skaliert. Optional kann noch der **Start**- und **Endwert** (Ein- und Ausgabewerte), sowie die Rotation um den Anker (**Winkel**) angegeben werden.




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

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

Einstellungen Bargraph / Instrumente

In dieser Gruppe sind Befehle für die Einstellung von Bargraphen und Instrumente zusammengefasst.

Bargraph/Instrument Wert setzen

#IVS	Obj-ID, Wert, Zeit(0), Aktionskurve-Nr.(0)

Mit dem Befehl wird ein Bargraph bzw. Instrument auf einen neuen **Wert** gesetzt. Der Parameter **Zeit** wird in 1/100s angegeben. Ist der Wert positiv, wird die Zeitdauer für den Gesamtausschlag verwendet. Die Geschwindigkeit ist somit konstant. Ein negativer Wert bestimmt dagegen die Zeit bis der neue Wert erreicht ist. Somit ist die Geschwindigkeit abhängig von der Entfernung. Die **Aktionskurve-Nr** bestimmt den zeitlichen Ablauf. Im Unterkapitel Aktionskurven und Aktionspfade ist dies näher erläutert.

Start/Endwert eines Bargraph/Instrument ändern

#IVN	Obj-ID, Startwert, Endwert(keine Änderung)

Der Befehl weist dem Bargraph bzw. Instrument einen neuen **Start-** und/oder **Endwert** zu.

Bargraph/Instrument Wert auf Kalkulationswert mit AutoUpdate setzen

#IVA	Obj-ID, Kalkulation, Zeit(0), Aktionskurve-Nr.(0)

Der Wert eines Bargraphs oder Instruments wird mit Hilfe der Kalkulation (z.B. Analogeingang, Register, Rechnung) automatisch berechnet und immer geändert, wenn sich sein Wert erneuert. Der Parameter **Zeit** wird in 1/100s angegeben. Ist der Wert positiv, wird die Zeitdauer für den Gesamtausschlag verwendet. Die Geschwindigkeit ist somit konstant. Ein negativer Wert bestimmt dagegen die Zeit bis der neue Wert erreicht ist. Somit ist die Geschwindigkeit abhängig von der Entfernung. Die **Aktionskurve-Nr** bestimmt den zeitlichen Ablauf. Im Unterkapitel Aktionskurven und Aktionspfade ist dies näher erläutert.

Bargraph/Instrument Kalkulation für AutoUpdate ändern

#IVC	Obj-ID, Kalkulation

Der Befehl ändert die **Kalkulation** für die AutoUpdate Funktion. Nun wird der Wert des Bargraphs/ Instruments immer dann erneuert, wenn sich der neue Kalkulationswert ändert. Für die Berechnung der Anzeige wird allerdings auf die alte

Kalkulation (#IVA) zurückgegriffen.

Keyboard / Tastatur #K

Befehlsgruppe um ein Keyboard für Tastureingaben darzustellen. Das Modul muss mit einem Touch ausgerüstet sein (Bestellnummern: EA uniTFTxxx-ATC oder EA uniTFTxxx-ATP). Im Normalfall wird das Keyboard mit einer EditBox verbunden.

Layout des Keyboards definieren (Keyboard Define Buttons)	**#KDB**	Obj-ID, Layout-Nr., "ButtonStringLine1"; ...; "ButtonStringLineN"
Definition alternativer Tastenbeschriftung/Styles (Keyboard Define Styles)	**#KDS**	Obj-ID, Code, ButtonStyle, "Label"; Code1, ButtonStyle1, "Label1" ...; CodeN, ButtonStyleN, "LabelN";
Keyboard platzieren und anzeigen (Keyboard Place)	**#KKP**	Obj-ID, ButtonStyleNormal, ButtonStyleSpecial, x, y, Anker, Tastenabstand, Gesamtbreite, Gesamthöhe(0), Erscheinen(0)


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

Layout des Keyboards definieren

#KDB	Obj-ID, Layout-Nr., "ButtonStringLine1"; ...; "ButtonStringLineN"

Ein Keyboard kann bis zu 4 unterschiedliche Layouts (**Layout-Nr.**) haben. Jedem Layout können Tasten(Codes) zugeordnet werden. Mehrere Zeilen werden durch Stringende ';' gekennzeichnet
Tasten können als String (z.B. "ASDF") oder als ASCII/ Unicode (z.B. $41 $53 $44 $56) übergeben werden. Für spezielle Tasten stehen folgende Keycodes zur Verfügung:

Code	Code im String	Beschreibung
1	\1	Anzeigen von Keyboard Layout-Nr. 1
2	\2	Anzeigen von Keyboard Layout-Nr. 2
3	\3	Anzeigen von Keyboard Layout-Nr. 3
4	\4	Anzeigen von Keyboard Layout-Nr. 4
5	\5	SHIFT (Verwende für einen Tastendruck das Keyboard Layout-Nr. 2, dann automatisch wieder Keyboard Layout-Nr. 1)
6	\6	CAPSLOG (Umschalten zwischen Keyboard Layout-Nr. 1, dann automatisch wieder Keyboard Layout-Nr. 2)

7	\7	DELETE
8	\8	BACKSPACE
9	\9	Reserviert für zukünftige Verwendung
10	\A	Reserviert für zukünftige Verwendung
11	\B	INSERT/ OVERWRITE
12	\C	CLEAR
13	\D	SEND
14	\E	Cursor left
15	\F	Cursor right
16	\G	Reserviert für zukünftige Verwendung
17	\H	Reserviert für zukünftige Verwendung
18	\I	POS1
19	\J	END
20	\K	Reserviert für zukünftige Verwendung
21	\L	CANCEL
22	\M	Cursor On/Off
23	\N	Platzhalter (Dieser Keycode wird nicht gezeichnet)
24 - 31	\O - \V	8 Funktionstasten

Definition alternativer Tastenbeschriftung/Styles

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

Einem bestimmten Keycode (**Code**) wird eine spezielle Tastenbeschriftung (**"Label"**) und **ButtonStyle** zuweisen. Diese Einstellung überschreibt die Styledefinition des Befehls #KKP. Die ButtonStyles werden nicht komplett übernommen, so wird die Größenangabe aus dem ButtonStyle ignoriert, der Radius wird einmalig übernommen. Ändert sich der Radius im ButtonStyle im Nachhinein wird dieser Werte nicht im Keyboard übernommen, eine Farbänderung oder TextStyleänderung hingegen schon.

Keyboard platzieren und anzeigen

#KKP	Obj-ID, ButtonStyleNormal, ButtonStyleSpecial, x, y, Anker, Tastenabstand, Gesamtbreite, Gesamthöhe(0), Erscheinen(0)

Das mit den Befehlen #KDB und #KDS definierte Keyboard wird an die Stelle x, y mit dem gegebenen Anker platziert. Die Breite einer Taste errechnet sich automatisch aus der **Gesamtbreite** bzw. der **Gesamthöhe** und den Abständen zwischen den Tasten (**Tastenabstand**). Ist die **Gesamthöhe**, oder **Gesamtbreite** =0 wird diese Länge automatisch aus der resultierenden Tastengröße berechnet. Die Größenwerte sind die gewünschten Maximalwerte. Die Tasten werden gleichmäßig aufgeteilt. Der **ButtonStyleNormal** definiert den Style für Buchstaben und Ziffern, **ButtonStyleSpecial** gilt für Sondertasten. Der letzte Parameter (**Erscheinen**) gibt an ob das Keyboard sofort angezeigt wird oder gemäß einer definierten Animation (#AOA / #AOR) erscheint:

Erscheinen	
0	Keyboard wird sofort angezeigt
>=1 (Time)	Keyboard erscheint gemäß Animation in der definierten Zeit (in 1/100s)

Eingabeelement per Touch #E

Befehlsgruppe Toucheingabelemente wie Menüs, SpinBoxen oder ComboBoxen zu erstellen. Die Funktionen sind ab der Firmware V1.2 verfügbar.

Menü

Style für Menü definieren (Input Menu Styles)	**#EMS**	Obj-ID, Ausklapprichtung, TextStyle-Nr., DrawStyle-Nr. Hintergrund, DrawStyle-Nr. Auswahl, DrawStyle-Nr. Icon, "NoteString"
Einträge für Menü definieren (Input Menu Define)	**#EMD**	Obj-ID, ItemNummer, "Eintrag"
Einem Menüeintrag ein Icon zuordnen (Input Menu Icons)	**#EMI**	Obj-ID, ItemNummer, <Iconname>, ItemNummer2, <Iconname2>, ... ItemNummerN, <IconnameN>
Enable/Disable Menüeintrag (Input Menu Enable)	**#EME**	Obj-ID, Enable, ItemNummer, ItemNummer2, ... ItemNummerN
Check/Uncheck Menüeintrag (Input Menu Check)	**#EMC**	Obj-ID, Check, ItemNummer, ItemNummer2, ... ItemNummerN
Menü platzieren und anzeigen (Input Menu Place)	**#EMP**	Obj-ID, x, y, Anker, Radius(0), RandX(0), RandY(0), Zeit(0)
Menüeintrag selektieren (Input Menu choOse)	**#EMO**	Obj-ID, ItemNummer

ComboBox

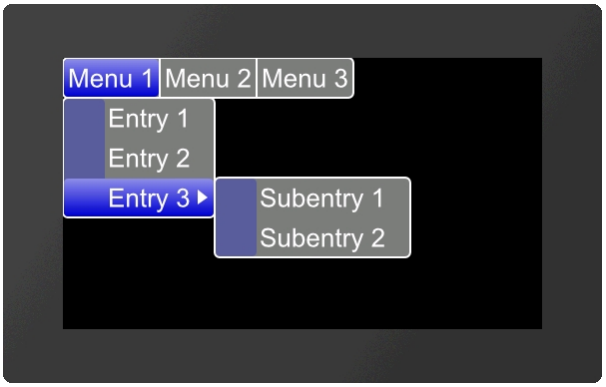
Styles für ComboBox definieren (Input Comobox Styles)	**#ECS**	Obj-ID, Ausklapprichtung, TextStyle-Nr., DrawStyle-Nr. Hintergund, DrawStyle-Nr. Auswahl, DrawStyle-Nr. Scrollbar, "NoteString"
Einträge für ComboBox definieren (Input Comobox Define)	**#ECD**	Obj-ID, "Einträge" Obj-ID, "Formatstring"; Startwert, Endwert
Einem ComboBox-Eintrag ein Icon zuordnen (Input Comobox Icons)	**#ECI**	Obj-ID, ItemNummer, <Iconname>, ItemNummer2, <Iconname2>, ... ItemNummerN, <IconnameN>
Enable/ Disable ComboBox-Eintrag (Input Comobox Enable)	**#ECE**	Obj-ID, Enable, ItemNummer, ItemNummer2, ... ItemNummerN
ComboBox platzieren und anzeigen (Input Comobox Place)	**#ECP**	Obj-ID, x, y, Anker, Radius(0), Breite(0), SichtbareEinträge(0), RandX(0), RandY(0), Zeit(0)
ComboBox-Eintrag selektieren (Input Comobox choOse)	**#ECO**	Obj-ID, ItemNummer

SpinBox

Styles für SpinBox definieren (Input Spinbox Styles)	**#ESS**	Obj-ID, Rollverhalten, TextStyle-Nr., DrawStyle-Nr. Hintergund, DrawStyle-Nr. Auswahl, "NoteString"
Einträge für SpinBox definieren (Input Spinbox Define)	**#ESD**	Obj-ID, Box-Nr, "Einträge" Obj-ID, Box-Nr, "Formatstring"; Startwert, Endwert
Einem SpinBox-Eintrag ein Icon zuweisen (Input Spinbox Icons)	**#ESI**	Obj-ID, ItemNummer, <Iconname>, ItemNummer2, <Iconname2>, ... ItemNummerN, <IconnameN>
Enable/Disable SpinBox-Eintrag (Input Spinbox Enable)	**#ESE**	Obj-ID, Enable, ItemNummer, ItemNummer2, ... ItemNummerN

SpinBox platzieren und anzeigen (Input Spinbox Place)	**#ESP**	Obj-ID, x, y, Anker, Radius, Breite, SichtbareEinträge, RandX(0), Abstand(0)
SpinBox-Eintrag selektieren (Input Spinbox choOse)	**#ESO**	Obj-ID, ItemNummer

Menü



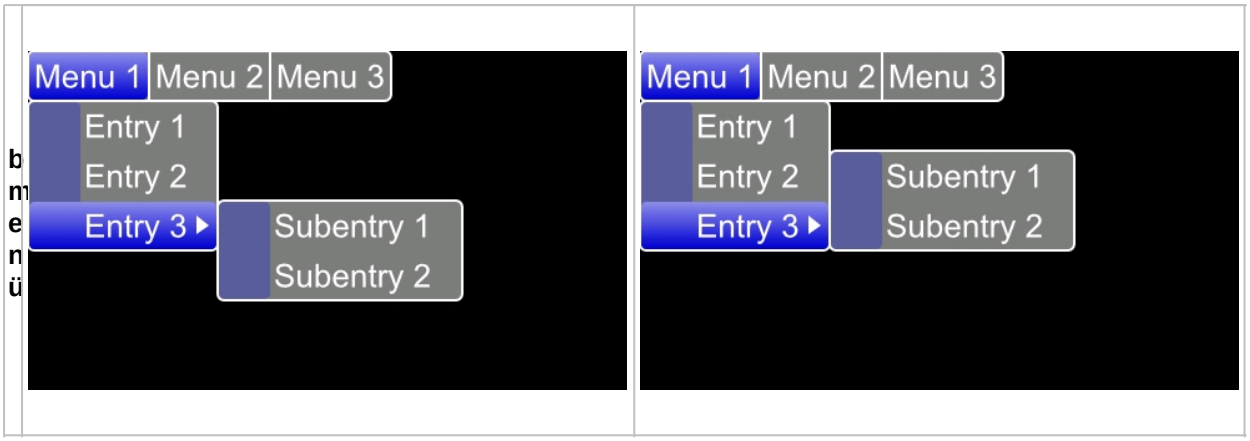

```
...
#EMS 1,0,4,1,5,17
#EMD 1,0,"Menu 1||Menu 2||Menu 3";
#EMD 1,1,"Entry 1|Entry 2|Entry 3";
#EMD 1,769,"Subentry 1|Subentry 2";
#EMP 1,0,271,1,5,5,5,10
...
```

Style für Menü definieren

#EMS	Obj-ID, Ausklapprichtung, TextStyle-Nr., DrawStyle-Nr. Hintergrund, DrawStyle-Nr. Auswahl, DrawStyle-Nr. Icon, "NoteString"

Mit dem Befehl wird das Aussehen des Menüs festgelegt. Es werden drei DrawStyles benötigt. Der Hintergrund des Menüs (**DrawStyle-Nr. Hintergrund**), das Aussehen des selektierten Eintrags (**DrawStyle-Nr. Auswahl**) und der Hintergrund des Icons (**DrawStyle-Nr. Icon**) werden definiert.Der Aufbau ist im Unterkapitel DrawStyle näher beschrieben. Mit dem TextStyle wird das Aussehen der Zeichenkette bestimmt (**TextStyle-Nr.**). Im Unterkapitel TextStyle ist dies genauer erläutert. Auch das **Ausklapprichtung** des Menüs wird hier definiert:

	Ausklapprichtung							
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Richtung	unten/rechts	oben/rechts	unten/links	oben/links	unten/rechts	oben/rechts	unten/links	oben/links
RichtungSu	normal				platzsparend			



Zuletzt kann optional noch der Parameter **"NoteString"** übergeben werden. Dieser definiert die abzuspielenden Noten.

Einträge für Menü definieren

#EMD	Obj-ID, ItemNummer, "Eintrag"

Der Befehl fügt dem Elternobjekt Untermenüs hinzu. Das Hauptmenü hat die **ItemNummer** 0, die Hauptmenüeintrage $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.

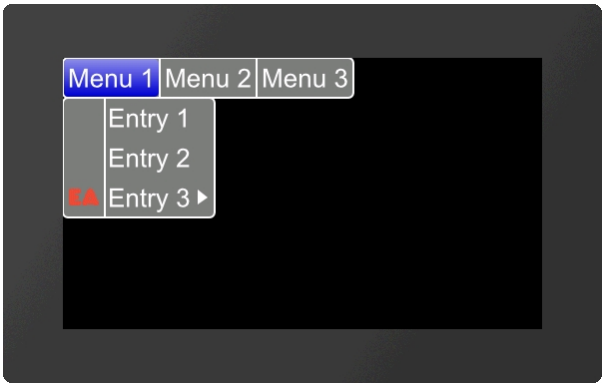
ItemNummer			
Hauptmenü 1 $01	Hauptmenü 2 $02	Hauptmenü 3 $03	...
Menü 1 $01 01			
Menü 2 $02 01			
Menü 3 $03 01	Submenü 1 $01 03 01		
...	Submenü 2 $02 03 01		
	...		

Die Einzelnen Einträge werden als String (**"Eintrag"**) mit einem Pipe '|' getrennt übergeben. Ein doppelter Pipe '||' fügt einen Trennstrich / Seperator hinzu.

Einem Menüeintrag ein Icon zuordnen

#EMI	Obj-ID, ItemNummer, <Iconname>, ItemNummer2, <Iconname2>, ... ItemNummerN, <IconnameN>

Jedem Eintrag (**ItemNummer**) kann ein Icon zugeordnet werden **<Iconname>**. Damit ein Icon zugewiesen werden kann, muss der Eintrag bereits vorhanden sein.




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

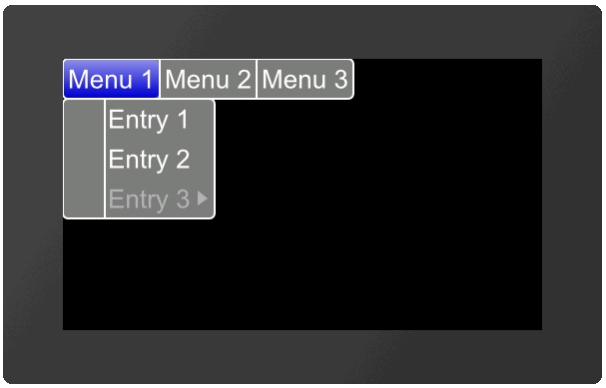
```
...
#EMI 1,$0301,"EA";
...
```

Enable/Disable Menüeintrag

#EME	Obj-ID, Enable, ItemNummer, ItemNummer2, ... ItemNummerN

Der Befehl aktiviert/deaktiviert einen Eintrag (**ItemNummer**). Ist ein Eintrag deaktiviert kann er nicht per Touch ausgewählt werden. Per default sind alle Einträge aktiv.

Enable	
0	Disable
1	Enable
2	Toggle




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

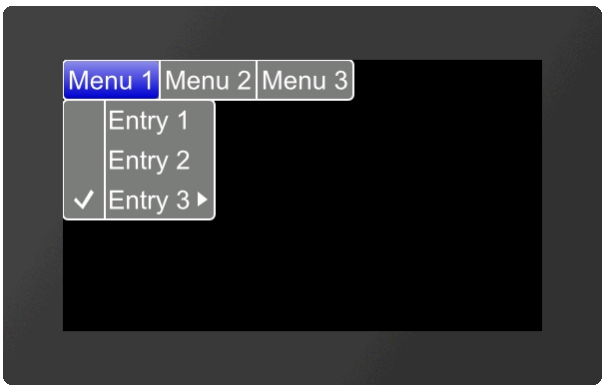
Check/Uncheck Menüeintrag

#EMC	Obj-ID, Check, ItemNummer, ItemNummer2, ... ItemNummerN

Der Befehl selektiert/deselektiert einen Eintrag (**ItemNummer**). Visuell wird ein Haken angezeigt. Per default ist kein Eintrag ausgewählt.

Check	
0	Deselektieren
1	Selektieren

2	Toggle




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

Menü platzieren und anzeigen

#EMP	Obj-ID, x, y, Anker, Radius(0), RandX(0), RandY(0), Zeit(0)

Das mit den Befehlen #EMS und #EMD definierte Menu wird an die Stelle x, y mit dem gegebenen Anker platziert. Der Parameter **Radius** gibt die Eckenabrundung. Mit den beiden optionalen Parametern (**RandX** und **RandY**) kann der Abstand des Textes zum Rand des Menüs angegeben werden. Mit dem Parameter **Zeit** kann das Öffnen/Schließen des Menüs zeitlich in 1/100s animiert werden.

Menüeintrag selektieren

#EMO	Obj-ID, ItemNummer

Der Befehl selektiert einen Eintrag (**ItemNummer**).

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
...
```

Styles für ComboBox definieren

#ECS	Obj-ID, Ausklapprichtung, TextStyle-Nr., DrawStyle-Nr. Hintergund, DrawStyle-Nr. Auswahl, DrawStyle-Nr. Scrollbar, "NoteString"

Mit dem Befehl wird das Aussehen der ComboBox festgelegt. Es werden drei DrawStyles benötigt. Der Hintergrund der ComboBox (**DrawStyle-Nr. Hintergrund**), das Aussehen des selektierten Eintrags (**DrawStyle-Nr. Auswahl**) und

das Aussehend der Scrollbar (**DrawStyle-Nr. Scrollbar**) werden definiert.Der Aufbau ist im Unterkapitel DrawStyle näher beschrieben. Mit dem TextStyle wird das Aussehen der Zeichenkette bestimmt (**TextStyle-Nr.**). Im Unterkapitel TextStyle ist dies genauer erläutert. Auch das **Ausklapprichtung** der ComboBox wird hier definiert:

Ausklapprichtung	
0	Nach unten
1	Nach oben

Zuletzt kann optional noch der Parameter **"NoteString"** übergeben werden. Dieser definiert die abzuspielenden Noten.

Einträge für ComboBox definieren

#ECD	Obj-ID, "Einträge"
	Obj-ID, "Formatstring"; Startwert, Endwert

Es gibt zwei Möglichkeiten die Einträge der ComboBox zu übergeben:

1. Die einzelnen Einträge werden als String (**"Eintrag"**) mit einem Pipe '|' getrennt übergeben (z.B. `"Eintrag1|Eintrag2|Eintrag3";`)
2. Die einzelnen Einträge werden als Formatsstring mit Start- und Endwert übergeben (z.B. `"Eintrag %d";1,3`)

Einem ComboBox-Eintrag ein Icon zuordnen

#ECI	Obj-ID, ItemNummer, <Iconname>, ItemNummer2, <Iconname2>, ... ItemNummerN, <IconnameN>

Jedem Eintrag (**ItemNummer**) kann ein Icon zugeordnet werden **<Iconname>**. Damit ein Icon zugewiesen werden kann, muss der Eintrag bereits vorhanden sein.


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

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

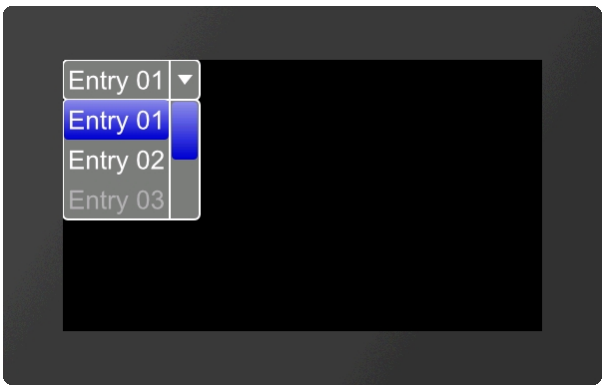
Enable/ Disable ComboBox-Eintrag

#ECE	Obj-ID, Enable, ItemNummer, ItemNummer2, ... ItemNummerN

Der Befehl aktiviert/deaktiviert einen Eintrag (**ItemNummer**). Ist ein Eintrag deaktiviert kann er nicht per Touch ausgewählt werden. Per default sind alle Einträge aktiv.

Enable	
0	Disable
1	Enable

2	Toggle




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

ComboBox platzieren und anzeigen

#ECP	Obj-ID, x, y, Anker, Radius(0), Breite(0), SichtbareEinträge(0), RandX(0), RandY(0), Zeit(0)

Die mit den Befehlen #ECS und #ECD definierte ComboBox wird an die Stelle x, y mit dem gegebenen Anker platziert. Der Parameter **Radius** gibt die Eckenabrundung. **Breite** gibt in Pixeln die Breite der Box an. Bei **Breite** =0 wird die Breite der Box anhand des breitesten Eintrages automatisch ermittelt.Der Parameter **SichtbareEinträge** definiert die Anzahl der sichtbaren Einträge (SichtbareEinträge =0: alle Einträge sichtbar). Mit den beiden optionalen Parametern (**RandX** und **RandY**) kann der Abstand des Textes zum Rand des Menüs angegeben werden. Mit dem Parameter **Zeit** kann das Öffnen/Schließen der ComboBox zeitlich in 1/100s animiert werden.

ComboBox-Eintrag selektieren

#ECO	Obj-ID, ItemNummer

Der Befehl selektiert einen Eintrag (**ItemNummer**).

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
...
```

Styles für SpinBox definieren

#ESS	Obj-ID, Rollverhalten, TextStyle-Nr., DrawStyle-Nr. Hintergund, DrawStyle-Nr. Auswahl, "NoteString"

Mit dem Befehl wird das Aussehen der SpinBox festgelegt. Es werden zwei DrawStyles benötigt. Der Hintergrund der

SpinBox (**DrawStyle-Nr. Hintergrund**) und das Aussehen des selektierten Eintrags (**DrawStyle-Nr. Auswahl**) werden definiert. Der Aufbau ist im Unterkapitel DrawStyle näher beschrieben. Mit dem TextStyle wird das Aussehen der Zeichenkette bestimmt (**TextStyle-Nr.**). Im Unterkapitel TextStyle ist dies genauer erläutert. Auch das **Rollverhalten** Position des Selektionsrahmens der SpinBox wird hier definiert:

	Rollverhalten			
	0	**1**	**2**	**3**
Rollverhalten	Endlos	Mit Anschlag	Endlos	Mit Anschlag
Selektionsrahmen	Hinter dem Text		Vor dem Text	

Zuletzt kann optional noch der Parameter **"NoteString"** übergeben werden. Dieser definiert die abzuspielenden Noten.

Einträge für SpinBox definieren

#ESD	Obj-ID, Box-Nr, "Einträge"
	Obj-ID, Box-Nr, "Formatstring"; Startwert, Endwert

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:

1. Die einzelnen Einträge werden als String (**"Eintrag"**) mit einem Pipe '|' getrennt übergeben (z.B. `"Eintrag1|Eintrag2|Eintrag3";`)
2. Die einzelnen Einträge werden als Formatsstring mit Start- und Endwert übergeben (z.B. `"Eintrag %d";1,3`)

Einem SpinBox-Eintrag ein Icon zuweisen

#ESI	Obj-ID, ItemNummer, <Iconname>, ItemNummer2, <Iconname2>, ... ItemNummerN, <IconnameN>

Jedem Eintrag kann ein Icon zugeordnet werden **<Iconname>**. Damit ein Icon zugewiesen werden kann, muss der Eintrag bereits vorhanden sein.
ItemNummer setzt sich folgendermaßen zusammen:

	ItemNummer			
	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-Eintrag

#ESE	Obj-ID, Enable, ItemNummer, ItemNummer2, ... ItemNummerN

Der Befehl aktiviert/deaktiviert einen Eintrag (**ItemNummer**). Ist ein Eintrag deaktiviert kann er nicht per Touch ausgewählt werden. Per default sind alle Einträge aktiv.

Enable	
0	Disable
1	Enable
2	Toggle


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

SpinBox platzieren und anzeigen

#ESP	Obj-ID, x, y, Anker, Radius, Breite, SichtbareEinträge, RandX(0), Abstand(0)

Die mit den Befehlen #ESS und #ESD definierte SpinBox wird an die Stelle x, y mit dem gegebenen Anker platziert. Der Parameter **Radius** gibt die Eckenabrundung. **Breite** gibt in Pixeln die Breite der Box an. Bei **Breite** =0 wird die Breite der Box anhand des breitesten Eintrages automatisch ermittelt. Der Parameter **SichtbareEinträge** definiert die anzuzeigenden Einträge oberhalb der Auswahlbox. **RandX** gibt den Abstand des Eintrages zum Boxrand und dem Icon an. **Abstand** definiert den Abstand der Boxen innerhalb der SpinBox-Gruppe.

SpinBox-Eintrag selektieren

#ESO	Obj-ID, ItemNummer

Der Befehl selektiert einen Eintrag (**ItemNummer**).

Action / Animation #A

Befehlsgruppe um Objekte zu animieren, z.B. Erscheinen, Wegfliegen, Rotieren oder Ausblenden zu lassen.

Animation definieren und einstellen

Animationsdefinition beginnen/beenden (Action Define Condition)	**#ADC**	Start
Animation absolut definieren (Action Object Absolut)	**#AOA**	Obj-ID, Action1, ..., ActionN
Animation relativ definieren (Action Object Relative)	**#AOR**	Obj-ID, Action1, ..., ActionN
Animationstyp und -zeit einstellen (Action Object Type)	**#AOT**	Obj-ID, Typ, Gesamtzeit(100), Start (0), Ende(0)
Animation stoppen (Action Object Stop)	**#AOS**	Obj-ID, Stop(0), Abbruchverhalten(0)
Animation löschen (Action Object Delete)	**#AOD**	Obj-ID, ..., Obj-IDn

Aktionspfade und Aktionskurven definieren

Aktinskurve definieren (Action Curve Define)	**#ACD**	Kurvennummer, x1, y1, x2, y2

Animation definieren und einstellen

Animationsdefinition beginnen/beenden

#ADC	Start

Ist die Animationsdefinition innerhalb eines Makros ist dieser Befehl nicht notwendig, da ein Makro immer komplett abgearbeitet wird, bevor der Bildschirminhalt neu gezeichnet wird.

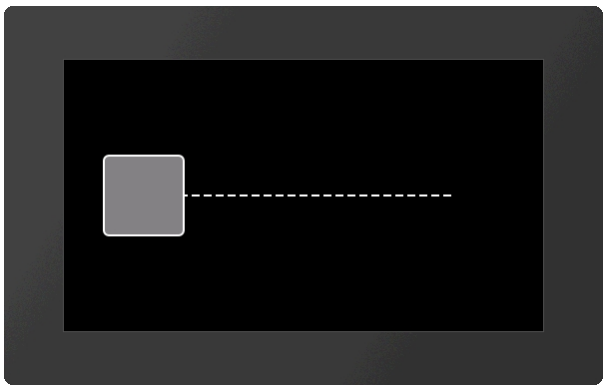
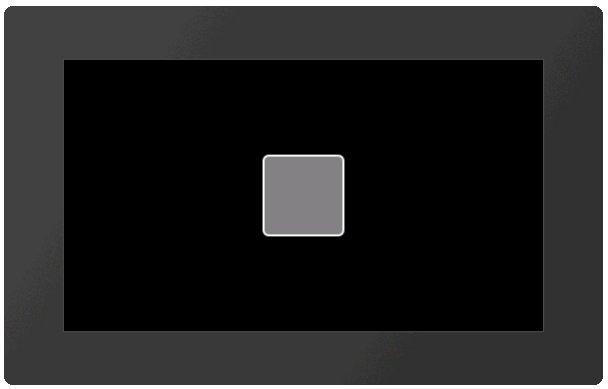
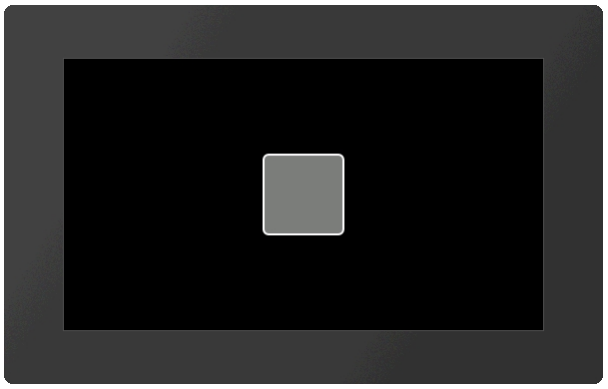
Start	
0	Animationsdefinition beenden → alle neuen Animationen starten
1	Animationsdefinition beginnen → keine Animation wird ausgeführt

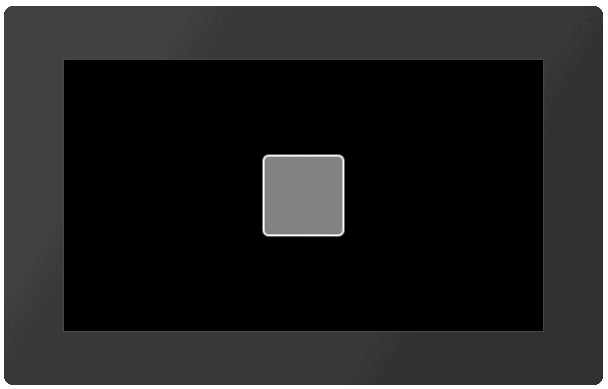
Animation absolut/relativ definieren

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

Der Befehl definiert eine Animation für ein Objekt. Der Parameter **Animation** gibt die Animation vor. Je nach Befehl werden **absolute #AOA** oder **relative #AOR** Werte übergeben.

Animation

Position ändern	100+ 1..10	x, y		... #AOA 1,104 ,400,136 ...
	Das Objekt bewegt sich in einer Geraden zum angegeben Punkt (**x**, **y**). Der zeitliche Verlauf wird durch die Aktionskurve (1..10) vorgegeben.			
Skalierung ändern	200+ 1..10	SkalierungX, SkalierungY		... #AOA 1,204 ,200,200 ...
	Die Objektgröße wird linear prozentual auf die Originalgröße verändert. Der zeitliche Verlauf wird durch die Aktionskurve (1..10) vorgegeben.			
Deckkraft ändern	500+ 1..10	Transparenz		... #AOA 2,504,0 ...
	Das Objekt ändert die Deckkraft (**Transparenz**). Der zeitliche Verlauf wird durch die Aktionskurve (1..10) vorgegeben.			

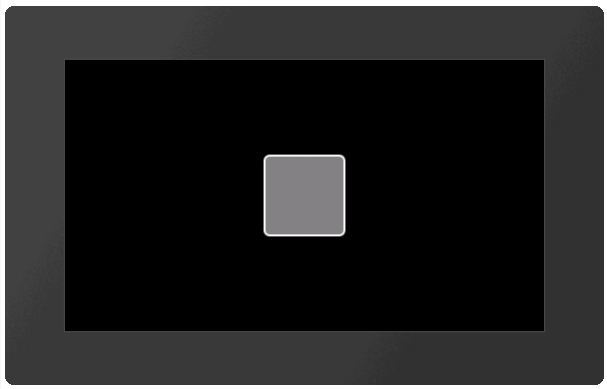
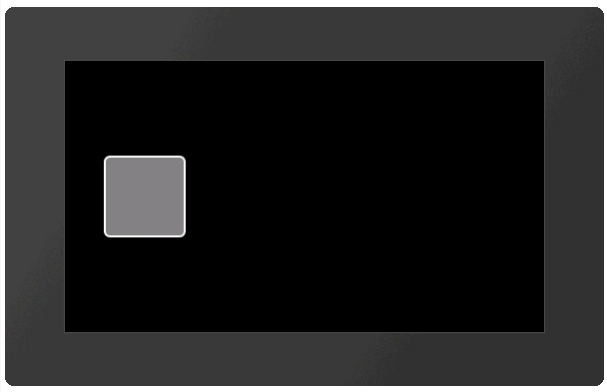
	600+ 1..10	R, G, B		... #AOA 2,604,- 27,-1,18 ...
Farbkanal ändern	Ändern der Farbkanäle **R**ot, **G**rün und **B**lau. Der Farbkanal der Zielfarbe wird relativ zu den übergebenen Parametern bestimmt. Die Parameter (R, G, B) werden als Prozentwerte im Bereich von -100 bis 100 übergeben. Beispiel: Angenommen die Ausgangsfarbe soll von RGB(50,0,0) auf RGB(200,0,0) geändert werden. Die Zielfarbe hat sich nur im Rotanteil geändert. Die Differenz des Rotanteils beträgt 150. Dieser muss noch in die Prozentdarstellung umgerechnet werden: $\frac{150}{255} 100 = 117{,}65$ `#AOA 1,604,118,0,0` Der zeitliche Verlauf wird durch die Aktionskurve (1..10) vorgegeben.			

Animationstyp und -zeit einstellen

#AOT	Obj-ID, Typ, Gesamtzeit(100), Start (0), Ende(0)

Die **Gesamtzeit** (in 1/100s) der Animation beinhaltet die Verzögerung beim **Start** (in 1/100s) und am **End**e (in 1/100s). Der **Typ** gibt den Animationstyp vor:

Typ		
1	Erscheinen	

2	Verschwinden	Objekt wird gelöscht	
3		Objekt wird unsichtbar	
4	Change (einmalig)		
5	Zyklisch		
6	PingPong		

Die Animation wird nach dem Befehl automatisch gestartet. Wurde jedoch davor der Befehl #ADC **mit dem Parameter 1** aufgerufen, wird die Animation erst nach dem Befehl `#ADC 0` ausgeführt.

Animation stoppen

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

Die Animation wird angehalten. Der Befehl kann nur bei periodischen Animationen angewendet werden (Zyklisch/Ping Pong). Der Parameter **Stop** gibt den Zeitpunkt an:

Stop	
0	Sofort
1	Am Anfang der Animation
2	Am Ende der Animation

Das **Abbruchverhalten** gibt an, was mit dem Objekt geschehen soll:

Abbruchverhalten	
0	Objekt behält aktuellen Zustand
1	Sprung (nur bei Stop =1 oder =2)
2	Objekt löschen
3	Objekt unsichtbar schalten

Animation löschen

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

Der Befehl löscht eine oder mehrere Animationen. Wird die Objekt-ID 0 übergeben, werden alle Animationen gelöscht.

Aktionskurven definieren

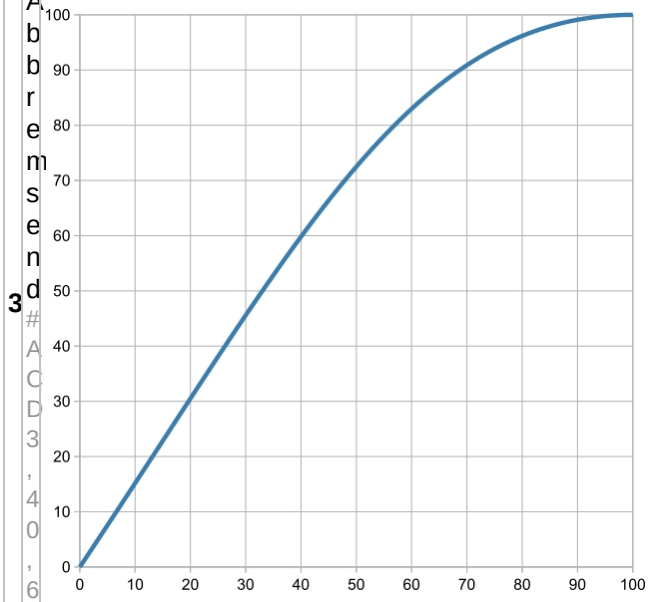
#ACD	Kurvennummer, x1, y1, x2, y2

Der Befehl erstellt eine eigene Aktionskurve, die den zeitlichen Verlauf der Animation vorgibt. Es sind 10 Kurven vordefiniert, die überschreiben werden können. **Kurvennummer** (1-10) gibt die Aktionskurve an, die überschrieben wird. Die Aktionskurve ist eine kubische Bézierkurve. Mit den Parametern **x1**, **y1**, **x2** und **y2** werden die Kontrollpunkte der Kurve vorgegeben. Der Wertebereich der Parameter ist für **X** 0...100, für Y -200...300.

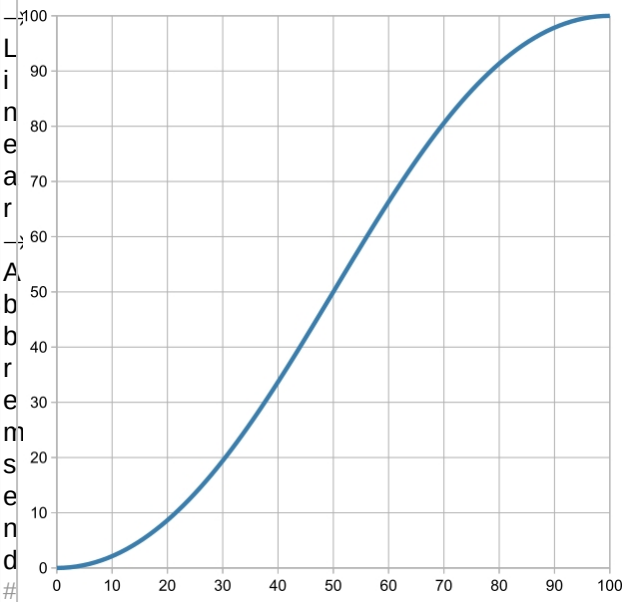
Vordefinierte Aktionskurven

Die Aktionskurven geben den zeitlichen Verlauf der Animation vor. Es sind 10 Kurven vordefiniert, die verändert werden können (#ACD). Bei den Kurven handelt es sich um kubische Bézierkurven mit zwei Stützpunkten:

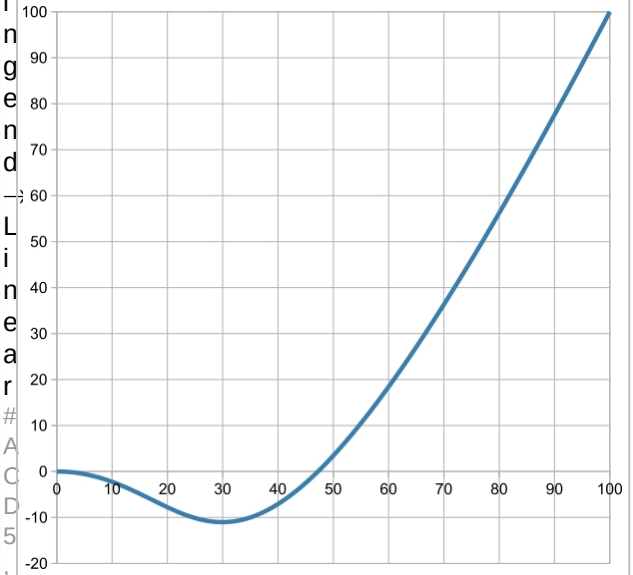
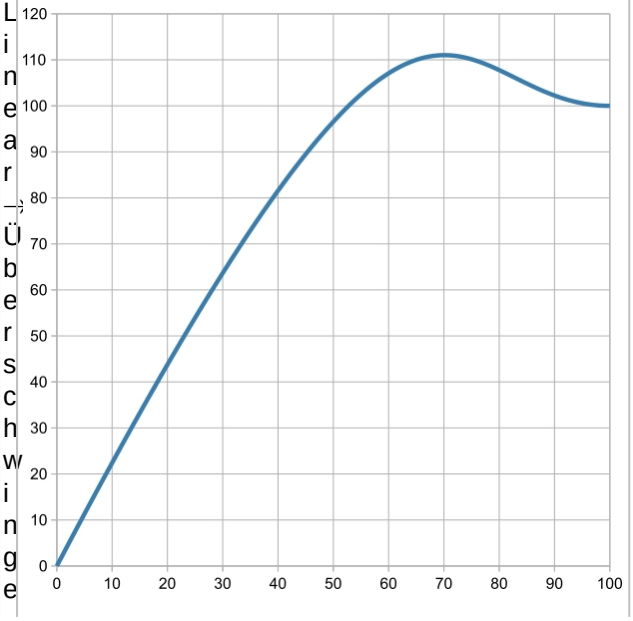
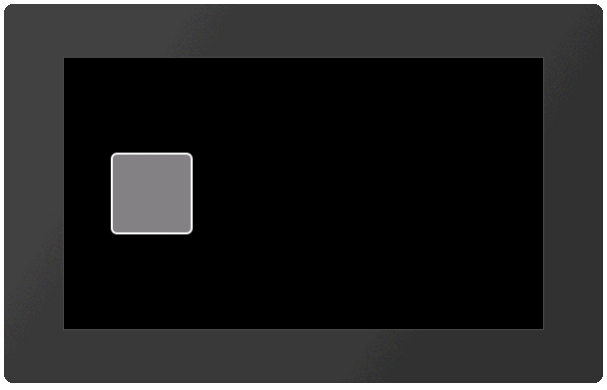
Aktionskurve	
1 Linear #ACD1,10,10,90,90	
2 Beschleunigend, Linear #ACD2,40,0,6	

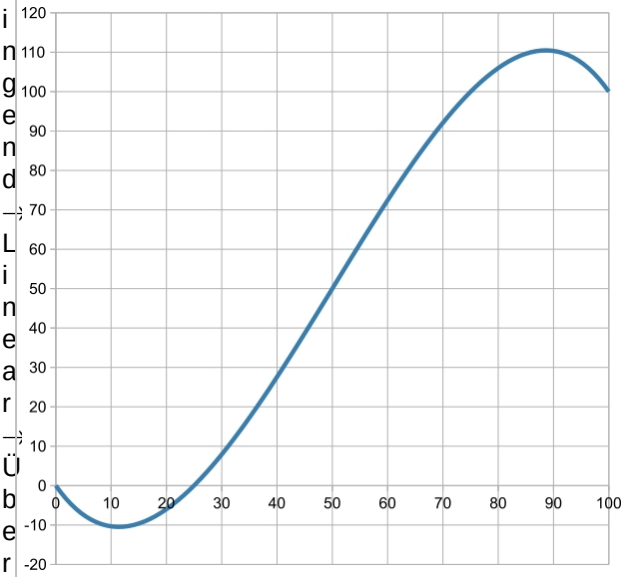
0,40		
3	 Linear → Abbremsend #ACD3,40,60,60,100 100 90 80 70 60 50 40 30 20 10 0 0 10 20 30 40 50 60 70 80 90 100	

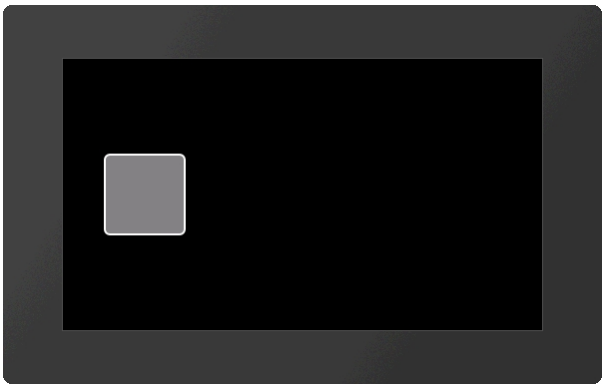
Beschleunigend_Linear_Abbremsend#ACD4,40,060,100

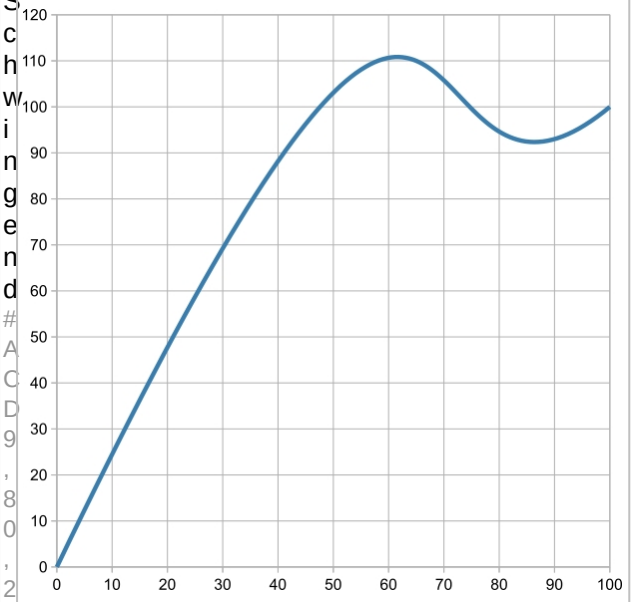
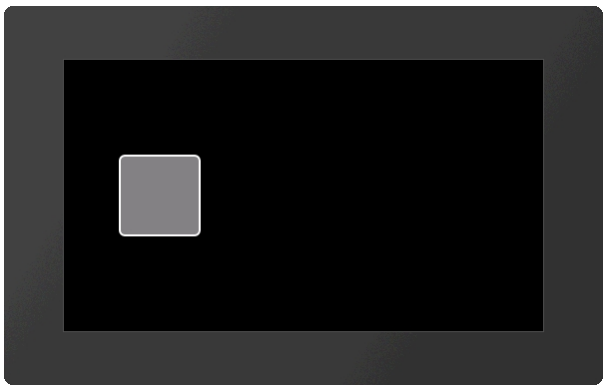
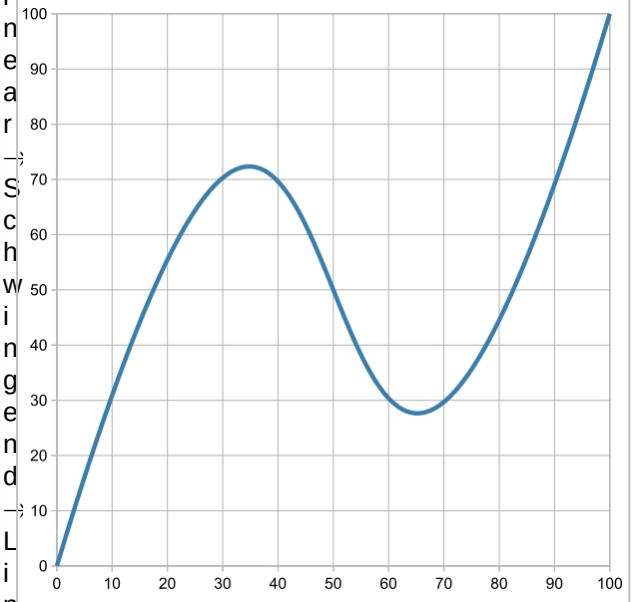
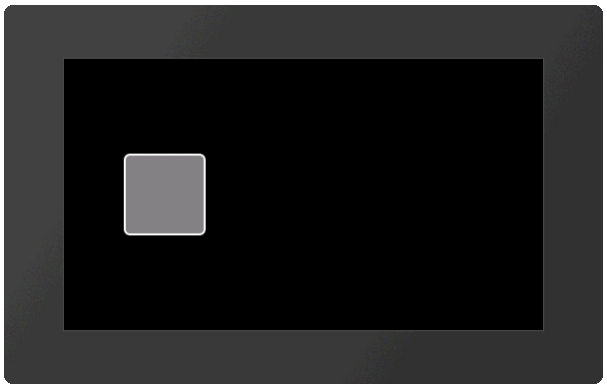


4

5	Unterschwingend_Linear#ACD5,30,0300,-60		
6	Linear_Überschwinge		

nd#ACD6,70,160,70,100		
7	Unterschwingen_Linear_Überschwingen 	

d#ACD7,30,-60,70,160		
8	Schwingend_Linear#ACD8,40,40,20,-100	

9	Linear→Schwingend#ACD9,80,200,60,60 	
10	Linear→Schwingend→Linie 	

ar#ACD10,60,20040,-100	

Objektverwaltung #O

Befehlsgruppe um Objekte zu Verwalten, zu Verändern oder zu Gruppieren.

Objektmanipulation

Objekt löschen (Object Delete Id)	**#ODI**	Obj-ID, ..., Obj-IDn
Objekt vor löschen schützen (Object Delete Protection)	**#ODP**	Löschschutz, Obj-ID, ..., Obj-IDn
Sichtbarkeit von Objekt ändern (Object Visible Id)	**#OVI**	Sichtbar, Obj-ID, ..., Obj-IDn
Position absolut ändern (Object Position Absolut)	**#OPA**	x, y, Obj-ID, ..., Obj-IDn
Position relativ ändern (Object Position Relative)	**#OPR**	x, y, Obj-ID, ..., Obj-IDn
Größe absolut ändern (Object Scale Absolut)	**#OSA**	Breite, Höhe, Obj-ID, ..., Obj-IDn
Größe relativ ändern (Object Scale Relative)	**#OSR**	Breite, Höhe, Obj-ID, ..., Obj-IDn
Objekt absolut rotieren (Object Rotation Absolut)	**#ORA**	Winkel, Obj-ID, ..., Obj-IDn
Objekt relativ rotieren (Object Rotation Relative)	**#ORR**	Winkel, Obj-ID, ..., Obj-IDn
Deckkraft absolut ändern (Object Opacity Absolut)	**#OOA**	Transparenz, Obj-ID, ..., Obj-IDn
Deckkraft relativ ändern (Object Opacity Relative)	**#OOR**	Transparenz, Obj-ID, ..., Obj-IDn
Objektfarbe ändern (Object Change Color)	**#OCC**	R,G,B, Obj-ID, ..., Obj-IDn
Objektstyle ändern (Object Change Style)	**#OCS**	Style-Nr, Obj-ID, ..., Obj-IDn
Rahmen/Hintergrund festlegen (Object Frame Place)	**#OFP**	DrawStyleNr, addX, addY, Obj-ID, ..., Obj-IDn
Anker setzen (Object Anchor Active)	**#OAA**	Anker, Obj-ID, ..., Obj-IDn
Freien Anker absolut setzen (Object Anchor Screen)	**#OAS**	x, y, Obj-ID, ..., Obj-IDn
Freien Anker relativ setzen (Object Anchor Object)	**#OAO**	x, y, Obj-ID, ..., Obj-IDn
Zeichenreihenfolge (Layer) absolut ändern (Object Layer Absolut)	**#OLA**	Zeichenreihenfolge, Obj-ID, ..., Obj-IDn
Zeichenreihenfolge (Layer) relativ ändern (Object Layer Realtive)	**#OLR**	Zeichenreihenfolge, Obj-ID, ..., Obj-IDn
Benutzerwert (Integer) setzen (Object User Integer)	**#OUI**	Obj-ID, Wert, ..., Wert n (Obj-ID n)
Benutzerwert (Float) setzen	**#OUF**	Obj-ID, Wert, ..., Wert n (Obj-ID n)

(Object User Float)		

Gruppe

Objekt zur Gruppe hinzufügen (Object Group Add)	**#OGA**	Obj-ID Gruppe, Obj-ID, ..., Obj-IDn

Hintergrundebene

Objekt auf die Hintergrundebene verschieben (Object to BackGround)	**#OBG**	RGB, Obj-ID, ..., Obj-IDn
Bild in die Hintergrundebene laden (Object Background Picture)	**#OBP**	<Name>, x(0), y(0), Anker(7),<Gradient>, Zeit, Richtung, 'Endmakro'

Objektmanipulation

Objekt löschen

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

Der Befehl löscht einzelne bzw. mehrere Objekte. Wird die **Obj-ID** = 0 übergeben, werden alle Objekte, bei **Obj-ID** = -1 alle Objekte und der Hintergrund gelöscht (ab V1.2).

Objekt vor löschen schützen

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

Objekte mit **Löschschutz** = 1 können vom Befehl #ODI nicht gelöscht werden und bleiben bestehen. Sie werden ebenso wenig auf die Hintergrundebene verschoben (ab V1.2).

Sichtbarkeit von Objekt ändern

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

Mit dem Befehl kann die Sichtbarkeit von Objekten eingestellt werden. Wird die **Obj-ID** = 0 übergeben, wird der Befehl auf alle Objekte angewendet:

Sichtbar	
0	Unsichtbar
1	Sichtbar

Siehe auch objV(id) (ab V1.4)

Position absolut/relativ ändern

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

Der Befehl verschiebt Objekte (absolut oder relativ) an die neue Position. Wird die **Obj-ID** = 0 übergeben, werden alle Objekte verschoben.

Siehe auch objX(id), objY(id)

Größe absolut/relativ ändern

#OSA #OSR	Breite, Höhe, Obj-ID, ..., Obj-IDn

Die **Breite** bzw. **Höhe** eines Objektes prozentual ändern. **Obj-ID** =0 Größenänderung bei allen Objekten.

Siehe auch objW(id), objH(id), objSW(id), objSH(id)

Objekt absolut/relativ rotieren

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

Das Objekt (**Obj-ID**) wird um den **Winkel** gedreht. **Obj-ID** =0 Drehung aller Objekten. Der Winkel kann nur in 90° Schritten gedreht werden.

Siehe auch objR(id)

Deckkraft absolut/ relativ ändern

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

Sichtbarkeit (**Transparent**) von 0 (komplett durchsichtig) bis 100 (komplett sichtbar) einstellen. **Obj-ID** =0 auf alle Objekte anwenden.

Siehe auch objO(id)

Objektfarbe ändern

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

Ändern der Farbkanäle **R**ot, **G**rün und **B**lau. Der Farbkanal der Zielfarbe wird relativ zu den übergebenen Parametern bestimmt. Die Parameter (R, G, B) werden als Prozentwerte im Bereich von -100 bis 100 übergeben.
Beispiel:
Angenommen die Ausgangsfarbe soll von RGB(50,0,0) auf RGB(200,0,0) geändert werden.
Die Zielfarbe hat sich nur im Rotanteil geändert. Die Differenz des Rotanteils beträgt 150. Dieser muss noch in die Prozentdarstellung umgerechnet werden:

$$\frac{150}{255} 100 = 117{,}65$$

```
#OCC 118,0,0,...
```

Die Änderungen der Farbkanäle beziehen sich immer auf die Ausgangsfarbe (auch bei Mehrfachanwendung). **Obj-ID** =0 auf alle Objekte anwenden.

Objektstyle ändern

#OCS	Style-Nr, Obj-ID, ..., Obj-IDn

Einem Objekt (**Obj-ID** =0 allen) wird ein neuer **Style** zugewiesen. Der **Style** richtet sich nach dem Objekt. Einer Zeichenkette wird z.B. automatisch ein TextStyle zugewiesen. Der Befehl kann nur auf einfache graphische Objekte angewendet werden (z.B. nicht auf Button, SpinBox, ...). Auch monochromen Bildern kann einmalig ein Style mit diesem Befehl zugeordnet werden.

Siehe auch objC(id)

Rahmen/Hintergrund festlegen

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

Einem Objekt (**Obj-ID** =0 allen) wird ein Hintergrund zugeordnet. Die Farben werden über den **DrawStyle** bestimmt. Die beiden Parameter **addX** und **addY** ändern am Linken/Rechten bzw. Oberen/Unteren Rand die Größe (in Pixeln) des Hintergrunds im Vergleich zum Objekt.

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

Anker setzen

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

Einem Objekt (**Obj-ID** =0 allen) wir ein neuer **Anker** zugewiesen. Der aktive Anker wird z.B. zum Rotieren des Objektes verwendet.

Siehe auch objA(id)

Freien Anker absolut/relativ setzen

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

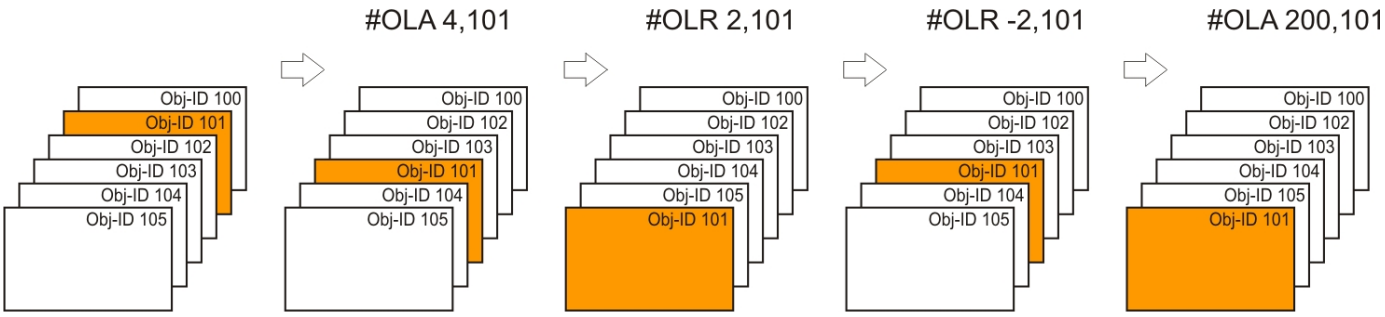
Den Anker 0 eines Objekts (**Obj-ID**) setzen. Der Befehl markiert gleichzeitig Anker 0 als aktiv.

Zeichenreihenfolge (Layer) absolut/ relativ ändern

#OLA	Zeichenreihenfolge, Obj-ID, ..., Obj-IDn

#OLR	

Der Befehl ändert die Zeichenreihenfolge eines oder mehrere Objekte. Das Objekt (**Obj-ID)** mit dem höchsten **Zeichenreihenfolge** wird als letztes gezeichnet.
Die Belegung der "Layer" beginnt mit dem Wert #1. Neue platzierte Objekte werden in einem jeweils höheren "Layer" gezeichnet. Diese können früher gezeichnete Objekte verdecken.



Eine Gruppe wird gemeinsam verschoben. Es können aber auch Objekte innerhalb einer Gruppe neu sortiert werden.

Benutzerwert (Integer) setzen

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

Jedem Objekt kann ein Integer-Wert zugeordnet werden. Der **Value** kann auch eine Kalkulation sein.

Siehe auch objUI(id)

Benutzerwert (Float) setzen

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

Jedem Objekt kann ein Float-Wert zugeordnet werden. Der **Value** kann auch eine Kalkulation sein.

Siehe auch objUF(id)

Gruppe

Objekt zur Gruppe hinzufügen

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

Eine Gruppe (**Obj-ID Gruppe**) erstellen bzw. einer bestehenden Gruppe Objekte hinzufügen.

Hintergrundebene

Objekt auf die Hintergrundebene verschieben

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

Existierende Objekte werden auf den Hintergrund verschoben. Die Hintergrundfarbe wird durch den Parameter **RGB** vorgegeben. Nach dem PowerOn-Reset ist die Hintergrundfarbe Schwarz (**RGB**=0). Wird RGB =-1 übergeben bleibt die bisher eingestellte Farbe unverändert.

Bild in die Hintergrundebene laden

#OBP	<Name>, x(0), y(0), Anker(7),<Gradient>, Zeit, Richtung, 'Endmakro'

Der Befehl platziert ein Bild direkt aus dem FLASH auf den Hintergrund. Transformationen (wie Skalieren) sind nicht möglich. Wenn Transformationen notwendig sind, muss ein Bildobjekt erstellt werden (#PPP) und die Transformationen angewendet werden bevor das Objekt mit dem Befehl #OBG auf den Hintergrund verschoben wird. Der Parameter **<Gradient>** gibt ein Graustufenbild an, welches zur Überblendung verwendet wird. Das Überblenden wird durch die Grauwerte und der **Zeit** in 1/100 s bestimmt. Der Überblendeffekt kann vorwärts oder rückwärts (**Richtung**) dargestellt werden. Nach dem Überblenden wird das Makro '**Endmakro'** aufgerufen.

Richtung	
0	Vorwärts
1	Rückwärts

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

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

Styles #C

Befehlsgruppe um Formatvorlagen zu erstellen. Das Aussehen jedes Objekts basiert auf einer Style passend zur Objektart. Für jeden Style stehen maximal 100 zur Verfügung.

DrawStyle

Füllung löschen (Style Fill Delete)	**#CFD**	DrawStyle-Nr.
Füllung mit Vollfarbe definieren (Style Fill Color)	**#CFC**	DrawStyle-Nr., RGB, Transparenz(100)
Füllung mit linearem Farbverlauf definieren (Style Fill Linear)	**#CFL**	DrawStyle-Nr, Farbverlauf-Nr, Winkel(0)
Füllung mit radialem Farbverlauf definieren (Style Fill Radial)	**#CFR**	DrawStyle-Nr, Farbverlauf-Nr, FokusX (5000), FokusY (0)
Füllung mit konischem Farbverlauf definieren (Style Fill Conial)	**#CFK**	DrawStyle-Nr, Farbverlauf-Nr, FokusX(5000), FokusY(0), Richtung(1)
Füllung mit Muster/Pattern definieren (Style Fill Pattern)	**#CFP**	DrawStyle-Nr, <PatternName>, 0, 0, 0, FokusX(5000), FokusY(0)
Winkel von linearem Farbverlauf ändern (Style Fill Angle)	**#CFA**	DrawStyle-Nr, Winkel
Farbverlauf ändern (Style Fill Garient)	**#CFG**	DrawStyle-Nr, Farbverlauf-Nr.
Fokus vom Farbverlauf ändern (Style Fill Focus)	**#CFF**	DrawStyle-Nr, FokusX, FokusY, PatternAnker (keine Änderung)
Muster/Pattern ändern (Style Fill pattern Name)	**#CFN**	DrawStyle-Nr, <PatternName>
Linie löschen (Style Line Delete)	**#CLD**	DrawStyle-Nr
Linienfarbe und-dicke definieren (Style Line Style)	**#CLS**	DrawStyle-Nr, RGB, Transparenz(100), Dicke(1), Verbindung(0),
Linienfarbe ändern (Style Line Color)	**#CLC**	DrawStyle-Nr, RGB, Transparenz (keine Änderung)
Linienbreite ändern (Style Line Width)	**#CLW**	DrawStyle-Nr, Dicke
Linienende/Verbindungsart ändern (Style Line End)	**#CLE**	DrawStyle-Nr, Verbindung

TextStyle

TextStyle definieren (Style Text Font)	**#CTF**	TextStyle-Nr, <FontName>, 0, Ausrichtung(0), DrawStyle-Nr.(0), 0, Zeilenabstand (0), Zeichenabstand (0)
Font ändern (Style Text Name)	**#CTN**	TextStyle-Nr, <FontName>
Ausrichtung ändern (Style Text Align)	**#CTA**	TextStyle-Nr, Ausrichtung

DrawStyle ändern (Style Text drawstyle)	#CTC	TextStyle, DrawStyle-Nr
Abstände ändern (Style Text Gap)	#CTG	TextStyle, Zeilenabstand, Zeichenabstand (keine Änderung)
Leerzeichenbreite ändern (Style Text space Width)	#CTW	TextStyle-Nr, LeerzeichenCode, Leerzeichenbreite(100)

ButtonStyle

ButtonStyle mit Bild definieren (Style Button Picture)	#CBP	ButtonStyle-Nr, <ButtonNameNormal>, <ButtonNameDown> (=Normal), Breite(0), Höhe(0), scale/pixel(0)
ButtonStyle definieren (Style Button Drawstyle)	#CBD	ButtonStyle-Nr, DrawStyle-Normal, DrawStyle-Down (=Normal), Breite (0), Höhe (0), Radius(0)
Text definieren (Style Button Textstyle)	#CBT	ButtonStyle-Nr, TextStyleNormal, TextStyleDown (=Normal), OffsetX(0), OffsetY(0)
DownEvent definieren (Style Button Offset)	#CBO	ButtonStyle-Nr, OffsetX(0), OffsetY (=OffsetX), Größe(100), Winkel(0)
Sound für DownEvent definieren (Style Button Sound)	#CBS	ButtonStyle-Nr, "SoundString"
Disabled ButtonStyle definieren (Style Button Greyout)	#CBG	R (-30), G (=R), B (=R), Transparenz(0)

Farbverlauf

Farbverlauf definieren (Style Color Ramp)	#CCR	Farbverlauf-Nr, Offset1, RGB1, Transparenz1, ... Offset10, RGB10, Transparenz10
Farbverlauf animieren (Style Animate Colorramp)	#CAC	Farbverlauf-Nr, Typ (1), Zeit (100)

DrawStyle

Füllung löschen

#CFD	DrawStyle-Nr.

Dieser Befehl löscht die Füllung des DrawStyles (**DrawStyle-Nr.**).

Füllung mit Vollfarbe definieren

#CFC	DrawStyle-Nr., RGB, Transparenz(100)

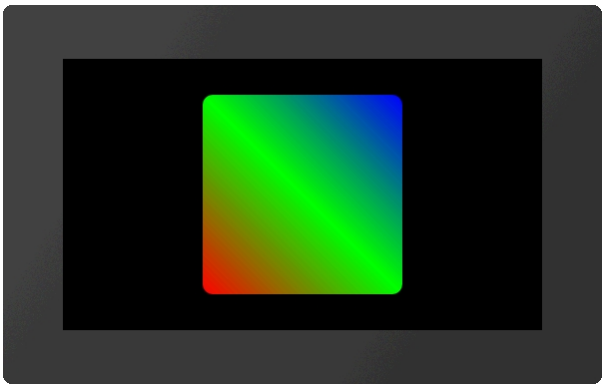
Dem DrawStyle (**DrawStyle-Nr.**) wird eine voll-farbige (**RGB**) Füllung zugewiesen. Die Deckkraft wird prozentual über **Transparenz** angegeben.

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

Füllung mit linearem Farbverlauf definieren

#CFL	DrawStyle-Nr, Farbverlauf-Nr, Winkel(0)

Dem DrawStyle **(DrawStyle-Nr.**) wird ein linearer Farbverlauf (**Farbverlauf-Nr**) zugewiesen. Der Verlauf muss vorab mit dem Befehl #CCR definiert werden. Optional kann noch die Ausrichtung angegeben werden (**Winkel** in Grad). Als Ausnahme kann bei diesem Befehl der Winkel in Grad-schritten angegeben werden.

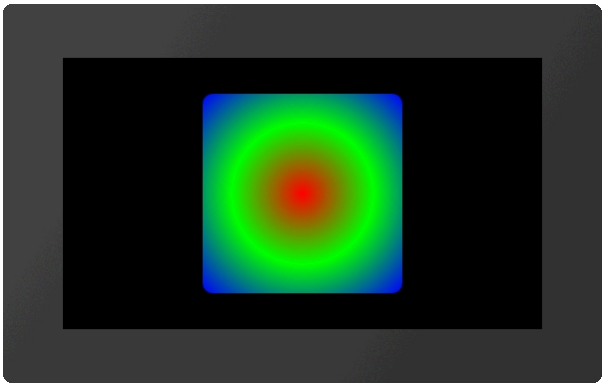


```
...
#CCR
5,0,$FF0000,100,50,$00FF00,100,100,$0000FF,100
#CFL 15,5,45
...
```

Füllung mit radialem Farbverlauf definieren

#CFR	DrawStyle-Nr, Farbverlauf-Nr, FokusX (5000), FokusY (0)

Dem DrawStyle (**DrawStyle-Nr**.) wird ein radialer Farbverlauf (**Farbverlauf-Nr**) zugewiesen. Der Verlauf muss vorab mit dem Befehl #CCR definiert werden. Der **Fokus** bestimmt prozentual den Startpunkt des Verlaufs. Bei **FokusX** =5000 wird mit **FokusY** der Anker angegeben der als Startpunkt des Verlaufs verwendet werden soll.



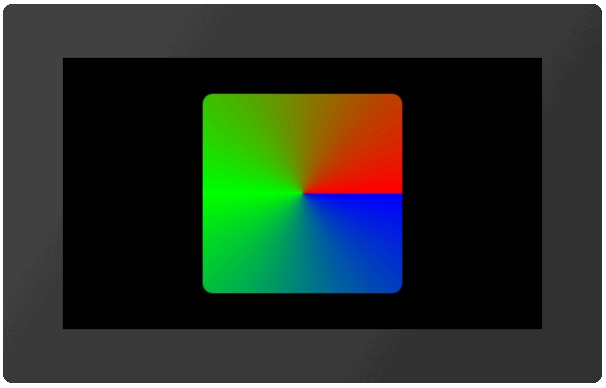
```
...
#CCR
5,0,$FF0000,100,50,$00FF00,100,100,$0000FF,100
#CFR 15,5,5000,5
...
```

Füllung mit konischem Farbverlauf definieren

#CFK	DrawStyle-Nr, Farbverlauf-Nr, FokusX(5000), FokusY(0), Richtung(1)

Dem DrawStyle (**DrawStyle-Nr**.) wird ein konischer Farbverlauf (**Farbverlauf-Nr**) zugewiesen. Der Verlauf muss vorab mit dem Befehl #CCR definiert werden. Der **Fokus** bestimmt prozentual den Startpunkt des Verlaufs. Bei **FokusX** =5000 wird mit **FokusY** der Anker angegeben der als Startpunkt des Verlaufs verwendet werden soll. Der optionale Parameter **Richtung** gibt den Drehsinn vor:

Richtung	
0	Gegen den Uhrzeigersinn
1	Im Uhrzeigersinn

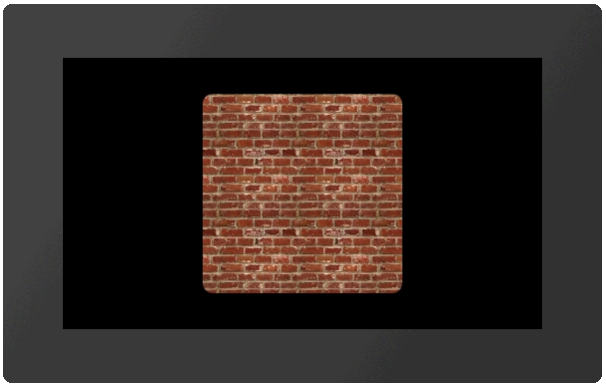


```
...
#CCR
5,0,$FF0000,100,50,$00FF00,100,100,$0000FF,100
#CFK 15,5,5000,5,0
...
```

Füllung mit Muster/Pattern definieren

#CFP	DrawStyle-Nr, <PatternName>, 0, 0, 0, FokusX(5000), FokusY(0)

Ein Muster (<**PatternName>**) wird als Füllung für den DrawStyle (**DrawStyle-Nr**.) verwendet. Der **Fokus** bestimmt prozentual den Startpunkt des Pattern. Bei **FokusX** =5000 wird mit **FokusY** der Anker angegeben der als Startpunkt des Patterns verwendet werden soll. Das Pattern wird direkt auf den Fokuspunkt gesetzt.



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

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

Winkel von linearem Farbverlauf ändern

#CFA	DrawStyle-Nr, Winkel

Der **Winkel** eines linearen Farbverlaufes wird verändert. Wird nur Übernommen bei einem linearen Farbverlauf und neu zeichnen des Objekts.

Farbverlauf ändern

#CFG	DrawStyle-Nr, Farbverlauf-Nr.

Dem DrawStyle wird ein neuer Farbverlauf zugewiesen (**Farbverlauf-Nr.**)

Fokus vom Farbverlauf ändern

#CFF	DrawStyle-Nr, FokusX, FokusY, PatternAnker (keine Änderung)

Der **Fokus** bestimmt prozentual den Startpunkt des Verlaufs oder des Patterns. Bei **FokusX** =5000 wird mit **FokusY** der Anker angegeben der als Startpunkt des Verlaufs verwendet werden soll. Der letzte Parameter (PatternAnker) ist nur bei Muster/Pattern notwendig: Das Pattern wird direkt mit dem **PatternAnker** auf den Fokuspunkt gesetzt.

Muster/Pattern ändern

#CFN	DrawStyle-Nr, <PatternName>

Der Füllung wird ein neues Muster (**<PatternName>**) zuweisen.

Linie löschen

#CLD	DrawStyle-Nr

Dieser Befehl löscht die Linie des DrawStyles (**DrawStyle-Nr.**).

Linienfarbe und-dicke definieren

#CLS	DrawStyle-Nr, RGB, Transparenz(100), Dicke(1), Verbindung(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:

Verbindung	
0	Harte Enden
1	Abgerundete Enden

Linienfarbe ändern

#CLC	DrawStyle-Nr, RGB, Transparenz (keine Änderung)

Der Linie eine neue Farbe (**RGB**) zuordnen.

Linienbreite ändern

#CLW	DrawStyle-Nr, Dicke

Die Dicke der Linie verändern.

Linienende/Verbindungsart ändern

#CLE	DrawStyle-Nr, Verbindung

Art des Linienendes bzw, Verbindung ändern:

Verbindung	
0	Harte Enden
1	Abgerundete Enden

TextStyle

TextStyle definieren

#CTF	TextStyle-Nr, <FontName>, 0, Ausrichtung(0), DrawStyle-Nr.(0), 0, Zeilenabstand (0), Zeichenabstand (0)

Definition eines TextStyle mit Font (**<FontName>**) und **Ausrichtung**.

Ausrichtung	
0	Linksbündig
1	Zentriert
2	Rechtsbündig

Der **DrawStyle** gibt die Farbe vor. Wir empfehlen aus Performancegründen eine einfache Füllung ohne Außenlinie. Die übrigen beiden Parameter geben den **Zeilenabstand** und zusätzlichen **Zeichenabstand** an.

Font ändern

#CTN	TextStyle-Nr, <FontName>

Der Befehl ändert den Font (**<FontName>**) des TextStyles.

Ausrichtung ändern

#CTA	TextStyle-Nr, Ausrichtung

Der Befehl ändert die **Ausrichtung** des Textes.

Ausrichtung	
0	Linksbündig
1	Zentriert
2	Rechtsbündig

DrawStyle ändern

#CTC	TextStyle, DrawStyle-Nr

Farbe mit Hilfe des DrawStyles (**DrawStyle-Nr.**) ändern.

Abstände ändern

#CTG	TextStyle, Zeilenabstand, Zeichenabstand (keine Änderung)

Es wird ein zusätzlicher **Zeilenabstand** bzw. **Zeichenabstand** definiert (Angabe in % der Zeichenhöhe). Auch negative Werte sind erlaubt.

Leerzeichenbreite ändern

#CTW	TextStyle-Nr, LeerzeichenCode, Leerzeichenbreite(100)

Die Breite des Leerzeichens kann von einem beliebigen anderen Code (**LeerzeichenCode**) übernommen werden. Zusätzlich kann die Breite in % definiert werden: Standard: 100 (**Leerzeichenbreite**).

ButtonStyle

ButtonStyle mit Bild definieren

#CBP	ButtonStyle-Nr, <ButtonNameNormal>, <ButtonNameDown> (=Normal), Breite(0), Höhe(0), scale/pixel(0)

Der Befehl definiert einen ButtonStyle: Anzeige von zwei Bildern für den ungedrückten (**<ButtonNameNormal>**) bzw. gedrückten (**<ButtonNameDown>**) Zustand. Die Größe wird über Breite und Höhe bestimmt (=0 Originalgröße). Die Größe kann nicht kleiner als das Originalbild sein. Der letzte Parameter **scale/pixel** gibt an ob das Bild skaliert (=0) werden soll, oder ob sich die Pixel in der Mitte des Bildes wiederholen (=1 Rahmenvergrößerung)

ButtonStyle definieren

#CBD	ButtonStyle-Nr, DrawStyle-Normal, DrawStyle-Down (=Normal), Breite (0), Höhe (0), Radius(0)

Der Befehl definiert einen ButtonStyle: Anzeige von zwei DrawStyles für den ungedrückten (**DrawStyle-Normal**) bzw. gedrückten (**DrawStyle-Down**) Zustand. Es folgend weitere Parameter für die **Breite** und **Höhe** des Buttons und der Eckenabrundung (**Radius**).

Text definieren

#CBT	ButtonStyle-Nr, TextStyleNormal, TextStyleDown (=Normal), OffsetX(0), OffsetY(0)

Den Text des **ButtonStyles** definieren. Der **Offset** gibt einen Zusätzlichen Abstand in Pixeln an, wo der Text auf dem Button positioniert wird.

DownEvent definieren

CBO	ButtonStyle-Nr, OffsetX(0), OffsetY (=OffsetX), Größe(100), Winkel(0)

Das Verhalten des Buttons im gedrückten Zustandwird definiert. Der Button wird mit dem **Offset** (in Pixeln) gezeichnet. Die **Größe** ändert sich proportional prozentual. Auch der **Winkel** (in Grad) kann geändert werden.

Sound für DownEvent definieren

CBS	ButtonStyle-Nr, "SoundString"

Im DownEvent des **ButtonStyles** wird eine kurze Notenfolge (**"SoundString"**) abgespielt. Ist der Parameter **"SoundString"** leer wird das Jingle gelöscht.

Disabled ButtonStyle definieren

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

Der deaktivierte Zustand eines Buttons ist die prozentuale Farbänderung des ButtonStyle-Normal. Jeder Farbkanal ist dabei einzeln ansprechbar. Die Deckkraft (**Opacity**) kann ebenfalls verändert werden.

Farbverlauf

Farbverlauf definieren

#CCR	Farbverlauf-Nr, Offset1, RGB1, Transparenz1, ... Offset10, RGB10, Transparenz10

Der Befehl definiert einen Farbverlauf. Der Stützpunkt (**Offset**) definiert den Farbpunkt im Verlauf in Prozent, die Farbe wird durch **RGB** und Deckkraft (**Transparenz**) angegeben. Es können maximal 10 Stützpunkte angegeben werden.

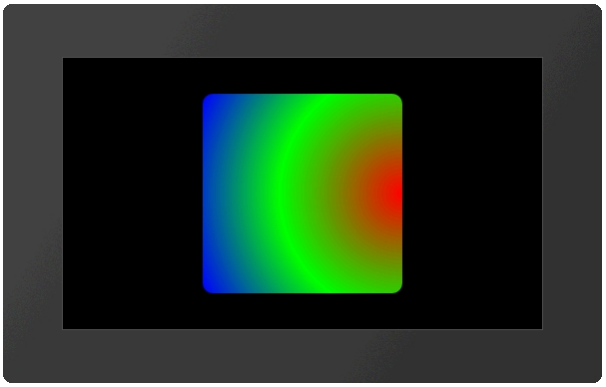
Farbverlauf animieren

#CAC	Farbverlauf-Nr, Typ (1), Zeit (100)

Die Position der Stützpunkte des Farbverlaufs werden verändert. Der Typ gibt den Animationstyp vor. Die Zeit in 1/100 s gibt die Zeitdauer an.

Typ

0	Animation stoppen
1	Zyklisch
2	Zyklisch rückwärts
3	PingPong
4	PingPong rückwärts



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

Makros #M

Einzelne oder mehrere Befehlsfolgen können als sogenannte Makros zusammengefasst und im internen FLASH fest abgespeichert werden. Diese können dann mit verschiedenen Befehlen gestartet werden. Ein Makro (*.emc) kann auch eine komplette Bildschirmseite aufbauen und ggfls. zuvor alle alten Objekte löschen (#ODI0).

Makros ausführen / Bildschirmseite bzw. Screen wechseln

Normalmakro ausführen (Macro Run Normal)	**#MRN**	<Makroname>
Normalmakro bedingt ausführen (Macro Run Conditionally)	**#MRC**	(Bedingung), <MakronameTrue>, <MakronameFalse>
Normalmakro verzögert ausführen (Macro Run Delayed)	**#MRD**	Verzögerung-Nr., Zeit, <Makroname>
I/O-Portmakro manuell ausführen (Macro Run Port)	**#MRP**	Port
I/O-Bitmakro manuell ausführen (Macro Run Bit)	**#MRB**	Portpin, Flanke
Analogmakro manuell ausführen (Macro Run Analogue)	**#MRA**	Kanal, Typ
Touchmakro manuell ausführen (Macro Run Touch)	**#MRT**	Obj-ID, Typ, Point-Nr.(0)

Makros definieren

Touchmakro definieren (Macro Define Touch)	**#MDT**	Obj-ID, <MakronameDown>, <MakronameUp>; <MakronameDrag>
Makro-Prozess definieren (Macro Process Define)	**#MPD**	Prozess-Nr, Zeit, <Makroname>, Startnummer(keine), Endnummer (Startnummer), Typ(1)
Bedingten Makro-Prozess definieren (Macro Process Conditionally)	**#MPC**	Prozess-Nr, Zeit, (Bedingung), <Makroname>, Startnummer(keine), Endnummer (Startnummer), Typ(1)
Automatischen Makro-Prozess definieren (Macro Process Autochange)	**#MPA**	Prozess-Nr, Zeit, (Berechnung), <Makroname>, Startnummer(keine), Endnummer (Startnummer), Typ(1)
Zeit von Makro-Prozess ändern (Macro Process Time)	**#MPT**	Prozess-Nr, Zeit
Action-Endmakro definieren (Macro Define Actionend)	**#MDA**	Obj-ID, <Makroname>
I/O Portmakro definieren (Macro Hardware Port)	**#MHP**	Port
I/O Bitmakro definieren (Macro Hardware Bit)	**#MHB**	Portpin, Flanke, <Makroname>
Analogmakro definieren (Macro Hardware Analogue)	**#MHA**	Kanal, Typ, <Makroname>
RS232 Empfangsmakro definieren (Macro Hardware RS232 master)	**#MHR**	Bufferanzahl,<Makroname>
Sekundenmakro (RTC) definieren (Macro Define Second)	**#MDS**	<Makroname>
Sound-Endmakro definieren (Macro Hardware Soundend)	**#MHS**	<Makroname>

Backlight Autodimmmakro definieren (Macro Define Led)	#MDL	<Makroname>
Gesten Makro definieren (Macro Define touch Gesture)	#MDG	Obj-ID, <MakronameDoubleClick>, <MakronameLongClick>
Makrodefinitionen löschen (Macro Clear Defines)	#MCD	Maske

Befehle innerhalb von Makros

Befehle überspringen (Macro File Skip)	#MFS	(Bedingung), Befehlen(1)
Sprung zu Ziel (Macro File Jump)	#MFJ	(Bedingung), Marker-Nr(0), Löschen(0)
Sprungziel setzen (Macro File Marker)	#MFM	Marker-Nr(0)
Sprung zu Ziel mit Rücksprung (Macro File Call)	#MFC	(Bedingung), Marker-Nr(0), Löschen(0)
Rücksprung zu Aufruf (Macro File Return)	#MFR	(Bedingung) (true)
Makro verlassen (Macro File Exit)	#MFE	(Bedingung) (true), <Makroname>
Marker löschen (Macro File Delete)	#MFD	Marker-Nr

Makros ausführen

Normalmakro ausführen / Seite starten

#MRN	<Makroname>

Der Befehl führt das Makro (**<Makroname>**) aus.

```
...
#MRN <P:macro/macro.emc>
...
```

```
...
#MRN
"macro";
...
```

Alternativ startet dieser Befehl eine komplette Bildschirmseite.

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

```
...
#MRN
"screen/S
creen1";
...
```

Normalmakro bedingt ausführen

#MRC	(Bedingung), <MakronameTrue>, <MakronameFalse>

Wenn die **Bedingung** wahr ist wird **<MakronameTrue>** ausgeführt, ansonsten **<MakronameFalse>**.

```
...
```

```
...
```

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

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

Normalmakro verzögert ausführen

#MRD	Verzögerung-Nr., Zeit, <Makroname>

Der Befehl führt das Makro (**<Makroname>**) verzögert aus. Es können gleichzeitig bis zu 10 Makros verzögert gestartet werden (**Verzögerung-Nr** 1...10). Die **Zeit** wird in 1/100 s angegeben.

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

```
...
#MRD
1
,100
,"macro";
...
```

I/O-Portmakro manuell ausführen

#MRP	Port

Der Befehl führt manuell ein Portmakro aus (**Port** 0...16).

I/O-Bitmakro manuell ausführen

#MRB	Portpin, Flanke

Der Befehl führt manuell ein Bitmakro aus (**Portpin** 0...136).

Flanke	
0	Fallend
1	Steigend

Analogmakro manuell ausführen

#MRA	Kanal, Typ

Der Befehl führt manuell ein Analogmakro aus (**Kanal** 0...3). Die Parametrisierung des Analogeingangs (Grenzen, Hysterese) werden mit der Befehlsgruppe 'Analog Input' eingestellt.

Typ	
0	Bei jeder Änderung
1	Dekrement

2	Inkrement
3	Kleiner als Grenze 1
4	Größer als Grenze 1
5	Kleiner als Grenze 2
6	Größer als Grenze 2
7	Fenster verlassen
8	Fenster eingetreten

Touchmakro manuell ausführen

#MRT	Obj-ID, Typ, Point-Nr.(0)

Der Befehl führt manuell ein Touchmakro aus (Objekt **Obj-ID**). **PointNr** (0...4) gibt den Finger an: 0=erster, 1=zweiter usw. Berührpunkt.

Typ	
1	Normal
2	Gedrückt
3	Drag

Makros definieren

Touchmakro definieren

#MDT	Obj-ID, <MakronameDown>, <MakronameUp>; <MakronameDrag>

Der Befehl definiert ein Touchmakro. Das Makro **<MakronameDown>** wird aufgerufen wenn die Taste gedrückt wird, **<MakronameUp>** beim Loslassen, **<MakronameDrag>** bei Ziehen (vor allem für Bargraphen und Instrumente sinnvoll). Bei einem Leerstring (**""**;) wird das entsprechende Makro gelöscht.
Module mit kapazitiven Touchpanel unterstützen auch Mehrfingerbedienung. Es werden bis zu 5 Punkte erkannt. Die ersten drei Makronamen gelten dann für den ersten Punkt, die nächsten drei für den zweiten usw. Ist kein spezielles Makro für den Punkt definiert, werden immer die Makros für den ersten Punkt aufgerufen.

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

```
1,"macroDOWN";"macroUP";"macroDRAG";...
```

Makro-Prozess definieren

#MPD	Prozess-Nr, Zeit, <Makroname>, Startnummer(keine), Endnummer (Startnummer), Typ(1)

Makroprozesse definieren einen automatischen zeitlichen Ablauf von Makros. Der Prozess (Prozess-Nr 1...10) ruft automatisch in (Zeit in 1/100s) das nächste Makro (<Makroname>) auf. Es können mehrere Makros aufgerufen werden (Startnummer bis Endnummer z.B.`#MPD 1,100,"MacroProcess";1,50` → MacroProcess1 .. MacroProcess50 werden aufgerufen. Der Typ gibt die Aufrufreihenfolge an:

Typ	
1	Zyklisch
2	Zyklisch rückwärts

3	PingPong
4	PingPong rückwärts
5	Einmalig
6	Einmalig rückwärts

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

```
...
#MPD 1,100,"MacroProcess";1,4
...
```

Bedingten Makro-Prozess definieren

#MPC	Prozess-Nr, Zeit, (Bedingung), <Makroname>, Startnummer(keine), Endnummer (Startnummer), Typ(1)

Bedingte Makroprozesse definieren einen automatischen zeitlichen Ablauf von Makros, wenn eine Bedingung erfüllt (wahr) ist. Der Prozess (**Prozess-Nr** 1...10) ruft automatisch in (**Zeit** in 1/100 s**)** das nächste Makro (**<Makroname>**) auf. Es können mehrere Makros aufgerufen werden (**Startnummer** bis **Endnummer** z.B.`#MPC 1,100,(R1<10),"MacroProcess";1,50` → MacroProcess1 .. MacroProcess50 werden aufgerufen. Der **Typ** gibt die Aufrufreihenfolge an:

Typ	
1	Zyklisch
2	Zyklisch rückwärts
3	PingPong
4	PingPong rückwärts
5	Einmalig
6	Einmalig rückwärts

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

```
...
#MPC 1,1000,(R1<10),"MacroProcess";...
```

Automatischen Makro-Prozess definieren

#MPA	Prozess-Nr, Zeit, (Berechnung), <Makroname>, Startnummer(keine), Endnummer (Startnummer), Typ(1)

Bedingte Makroprozesse definieren einen automatischen zeitlichen Ablauf von Makros, wenn sich der Wert der **Berechnung** geändert hat. Der Prozess (**Prozess-Nr** 1...10) ruft automatisch in (**Zeit** in 1/100 s**)** das nächste Makro (**<Makroname>**) auf. Es können mehrere Makros aufgerufen werden (**Startnummer** bis **Endnummer** z.B.`#MPA 1,100,(R1<10),"MacroProcess";1,50` → MacroProcess1 .. MacroProcess50 werden aufgerufen. Der **Typ** gibt die Aufrufreihenfolge an:

Typ	
1	Zyklisch
2	Zyklisch rückwärts
3	PingPong
4	PingPong rückwärts
5	Einmalig
6	Einmalig rückwärts

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

```
...
#MPA 1,100,(R1),"MacroProcess";1,50
...
```

```
s
"
;
.
.
.
```

Zeit von Makro-Prozess ändern

#MPT	Prozess-Nr, Zeit

Die **Zeit** (1/100 s) für den Makroprozess (**Prozess-Nr** =0 alle) wird geändert.

Zeit	
-1	Mit alter Zeit neu beginnen
0	Stoppen
>0	Zeit neu setzen

Action-Endmakro definieren

#MDA	Obj-ID, <Makroname>

Nach dem Ende einer Objekt-Animation (**Obj-ID**) wird das Makro (**<Makroname>**) aufgerufen.

```
...
#MDA 1,
...
```

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

I/O Portmakro definieren

#MHP	Port

Das Portmakro wird aufgerufen wenn sich der Status des **Ports** (0...15) ändert.

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

```
...
#MHP
0
,"Macro";
...
```

I/O Bitmakro definieren

#MHB	Portpin, Flanke, <Makroname>

Das Bitmakro wird aufgerufen wenn eine **Flanke** am **Portpin** (0...127) registriert wird.

Flanke

0	Fallend
1	Steigend
2	Beide Flanken

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

```
...
#MHB
16
,
2
,"Macro";
...
```

Analogmakro definieren

#MHA	Kanal, Typ, <Makroname>

Einem A/D-Eingang (**Kanal** 0...3) wird ein Makro (**<Makroname>**) zugewiesen. **Typ:**

Typ	
0	Bei jeder Änderung
1	Dekrement
2	Inkrement
3	Kleiner als Grenze 1
4	Größer als Grenze 1
5	Kleiner als Grenze 2
6	Größer als Grenze 2
7	Fenster verlassen
8	Fenster eingetreten

Die Parametrisierung des Analogeingangs (Grenzen, Hysterese) werden mit der Befehlsgruppe 'Analog Input' eingestellt.

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

```
...
#MHA
0
,
0
,"Macro";
...
```

RS232 Empfangsmakro definieren

#MHR	Bufferanzahl,<Makroname>

Das Makro (**<Makroname>**) wird aufgerufen wenn die **Bufferanzahl** (0=disable) im Master RS232 Empfangsbuffer erreicht wird.

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

```
...
#MHR
42
,"Macro";
...
```

Sekundenmakro (RTC) definieren

#MDS	<Makroname>

Das Makro (**<Makroname>**) wird jede Sekunde aufgerufen.

```
...
#MDS <P:macro/Macro.emc>
...
```

```
...
#MDS
"Macro";
...
```

Sound-Endmakro definieren

#MHS	<Makroname>

Das Makro (**<Makroname>**) wird aufgerufen wenn der Notenstring zu Ende gespielt ist. Die Definition muss vor dem Abspielen der Notenfolge erfolgen. Bei automatisch abgespielten Tönen (z.B. Button) wird das Makro nicht aufgerufen (nur bei manuellem Starten #HTN)

```
...
#MHS <P:macro/Macro.emc>
...
```

```
...
#MHS
"Macro";
...
```

Backlight Autodimmmakro definieren

#MDL	<Makroname>

Die automatische Dimmfunktion des Backlights ruft bei Änderung des Zustand das angegebene Makro (**<Makroname>**) auf. Siehe den Befehl #XAL für Parametereinstellungen, wie Helligkeit und Zeit.

```
...
#MDL <P:macro/Macro.emc>
...
```

```
...
#MDL
"Macro";
...
```

Gesten Makro definieren

#MDG	Obj-ID, <MakronameDoubleClick>, <MakronameLongClick>

Der Befehl definiert ein Gestenmakro. Das Makro **<MakronameDoubleClick>** wird bei einem Doppelklick, das Makro **<MakronameLongClick>** bei einem langen Klick aufgerufen.

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

```
...
#MDG
1
,"MacroDo
ubleClick
"
;"MacroLo
ngClick";
...
```

Makrodefinitionen löschen

#MCD	Maske

Der Befehl löscht Makrodefinitionen nach dem Typ:

Maske	
0	Sekunden Makros (RTC)
2	Prozess Makros
4	Port Makros
8	Bit Makros
16	Analog Makros
32	Touch/Gesten Makros
64	Action Makros
128	Verzögerte Makros
256	Backlight Makro
512	RS232 Empfangsmakro
1024	Sound Endmakro

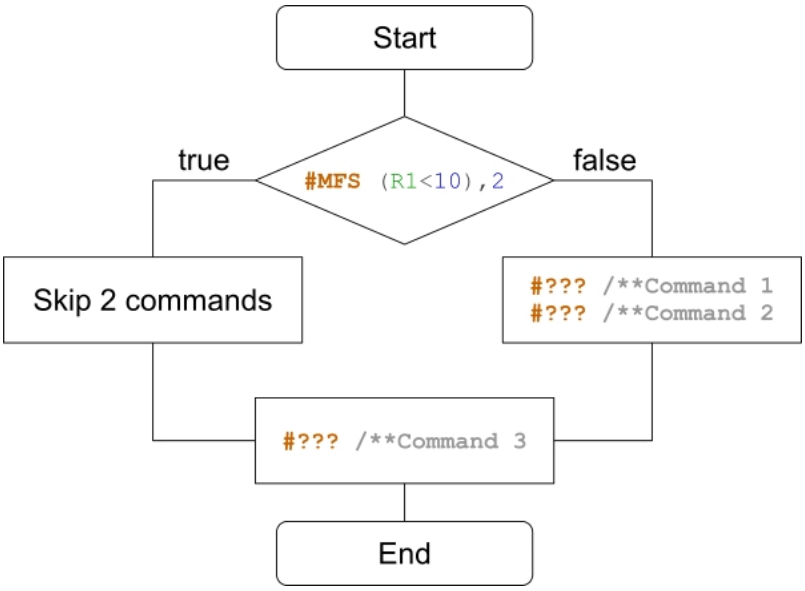
Die einzelnen Typen können addiert werden, z.B. alle Makros löschen: **Maske** = $FFFF

Befehle innerhalb von Makros

Befehle überspringen

#MFS	(Bedingung), Befehlen(1)

Wenn die **Bedingung** wahr ist, überspringt der Befehl die definierte Anzahl an **Befehlen** (Leerzeilen und Kommentare werden ignoriert) im Makro.

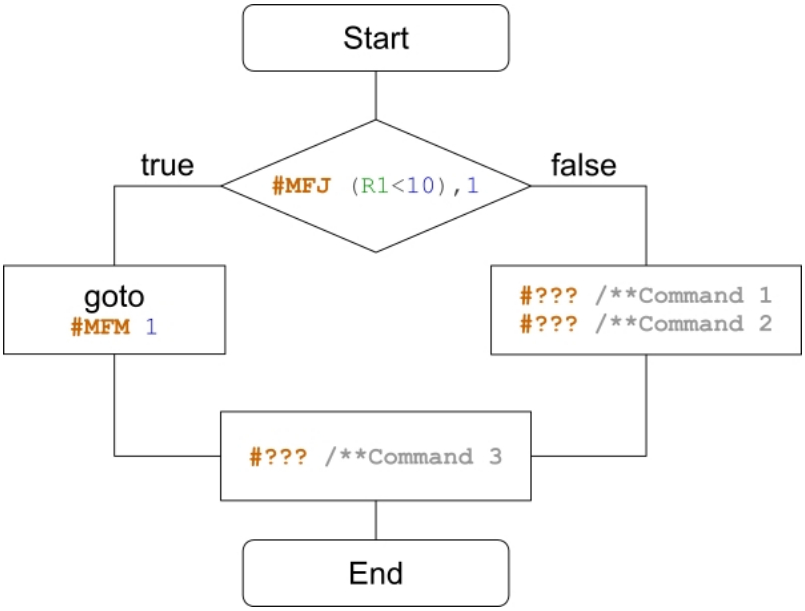



```
...
#MFS (R1<10),2
   #??? /**Comman
d 1
   #??? /**Comman
d 2
#??? /**Command 3
...
```

Sprung zu Ziel

#MFJ	(Bedingung), Marker-Nr(0), Löschen(0)

Wenn die **Bedingung** wahr ist, springt der Befehl zum Marker (**Marker-Nr.** 0..99) im Makro. Ein Marker kann in einem Makro mehrmals vorkommen. Der Parameter **Löschen** löscht den letzten gefundenen Marker und sucht in dem Makro den nächsten Marker mit der gleichen ID.




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

Sprungziel setzen

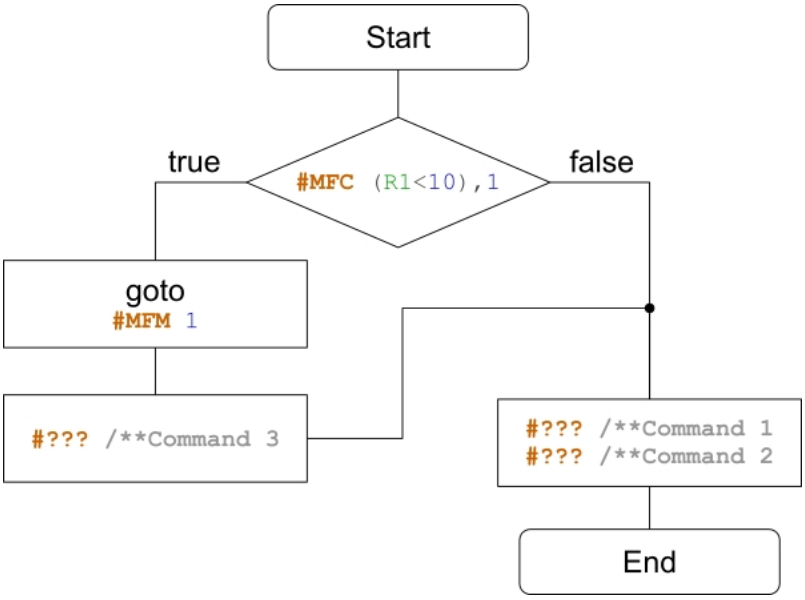
#MFM	Marker-Nr(0)

Der Befehl setzt ein Sprungziel (**Marker-Nr.** 0..99) im Makro.

Sprung zu Ziel mit Rücksprung

#MFC	(Bedingung), Marker-Nr(0), Löschen(0)

Wenn die **Bedingung** wahr ist, springt der Befehl zum Marker (**Marker-Nr.** 0..99) im Makro. Ein Marker kann in einem Makro mehrmals vorkommen. Für den Rücksprung zum Aufruf ist zwingend ein Return (#MFR) notwendig. Der Parameter **Löschen** löscht den letzten gefundenen Marker und sucht in dem Makro den nächsten Marker mit der gleichen ID.




```
...
#MFC (R1<10),1
#??? /**Command 1
#??? /**Command 2
#MFE

#MFM 1
   #??? /**Comman
d 3
#MFR
...
```

Rücksprung zu Aufruf

#MFR	(Bedingung) (true)

Wenn die **Bedingung** wahr ist, springt der Befehl zum Aufruf (Call).

Makro verlassen

#MFE	(Bedingung) (true), <Makroname>

Wenn die **Bedingung** wahr ist, wird das Makro verlassen. Optional kann ein weiteres Makro (**<Makroname>**) aufgerufen werden.

Marker löschen

#MFD	Marker-Nr

Der Befehl löscht den letzten Marker mit der **Marker-Nr**.

Vergleich zwischen C-Code und Makro-Code

if-Abfrage einzeilig	
C-Code	Makro-Code

`if(R1<10)`
`  //Command`
`else`
`  //Command`
	`#MFS (R1>=10),3`
	`  #??? /**Command`
	`#MFS (R1<10),1`
	`  #??? /**Command`

if-Abfrage mehrzeilig	
C-Code	Makro-Code

`if(R1<10)`
`{`
`  //Command 1`
`  //Command 2`
`}`
`else`
`{`
`  //Command 1`
`  //Command 2`
`}`
	`#MFJ (R1>=10),1`
	`  #??? /**Command 1`
	`  #??? /**Command 2`
	`  #MFJ (1),2`
	`#MFM 1`
	`  #??? /**Command 1`
	`  #??? /**Command 2`
	`#MFM 2`

for-Schleife	
C-Code	Makro-Code

`for(int i=0;i<10;i++)`
`{`
`  //Command 1`
`  //Command 2`
`}`
	`#VRI 0,0`
	`#MFM 1`
	`  #??? /**Command 1`
	`  #??? /**Command 2`
	`#MFJ (++R0<10),1`

do-Schleife	
C-Code	Makro-Code

`do`
`{`
`  //Command 1`
`  //Command 2`
`}while(R1<10)`
	`#MFM 1`
	`  #??? /**Command 1`
	`  #??? /**Command 2`
	`#MFJ (R0<10),1`

Funktionsaufruf	
C-Code	Makro-Code

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

Variablen / Register #V

Befehlsgruppe um interne Rechnungen und logische Operationen auszuführen. Mit Hilfe der Stringfiles kann eine Mehrsprachigkeit realisiert werden. Es sind 200 Register vorhanden. Stringregister können bis zu 200 Zeichen aufnehmen, bei Festkommaregistern wird mit signed 32 Bit, bei Fließkommaregistern wird mit 23 Bit Mantisse, 8 Bit Exponent, 1 Bit signed gerechnet.

Stringfiles / Mehrsprachigkeit

Stringfile laden (Variable stringFile Load)	**#VFL**	StringfileName>
Stringfile löschen (Variable stringFile Delete)	**#VFD**	<StringfileName> (alle)
Anzahl an geladenen Stringfiles senden (Variable stringFile Count)	**#VFC**	

Stringregister

Stringregister setzen (Variable Stringregister Set)	**#VSS**	String-ID, "String"; "String" [ID+1]; ...
Stringregister ab Position setzen (Variable Stringregister Postion)	**#VSP**	String-ID, Offset, "New String";
Stringregister ab Position ersetzen (Variable Stringregister Replace)	**#VSR**	String-ID, Offset, "New String";
Teilstring aus Stringregister ausschneiden und ersetzen (Variable Stringregister Truncate)	**#VST**	String-ID, Offset, Anzahl(bis Ende)
Teilstring aus Stringregister kopieren (Variable Stringregister Copy)	**#VSC**	String-ID Ziel, String-ID Quelle, Offset (0), Anzahl(bis Ende)
Stringregister in Teilstrings aufteilen (Variable Stringregister dElimiter)	**#VSE**	String-ID Ziel Start, String-ID Quelle, Seperator, Register-ID (=String-ID Ziel Start)
Stringregister mit Zeit/Datum setzen (Variable Stringregister Date)	**#VSD**	String-ID, "Datumsformat"; date (aktuelle Zeit)
Formatiertes Stringregister setzen (Variable Stringregister Formated)	**#VSF**	String-ID, "Formatstring"; Wert1, Wert2, ..., WertN
Objektsstrings auslesen (Variable Stringregister Object)	**#VSO**	String-ID, Obj-ID, ...
Stringregister senden (ASCII) (Variable string Send Ascii)	**#VSA**	String-ID, ...
Stringregister senden (Unicode) (Variable string Send Unicode)	**#VSU**	String-ID, ...
Stringregister sortieren (Variable Quicksort Strings)	**#VQS**	String-ID Start, String-ID Ende, Anzahl (0), Offset(0)
Codes im Stringregister sortieren (Variable Quicksort Codes)	**#VQC**	String-ID, Richtung (1), Anzahl (0), Offset (0)
Letzte Fehlermeldung in Stringregister (Variable Stringregister Last error)	**#VSL**	String-ID, Löschen(1)

Stringregister mischen (Variable Mix Strings)	**#VMS**	String-ID Start, String-ID Ende
Codes im Stringregister mischen (Variable Mix Codes)	**#VMC**	String-ID, Anzahl (0), Offset (0)

Register

Register setzen (Integer) (Variable Register Integer)	**#VRI**	Register-ID, Wert, Wert1 [ID+1], ...
Register setzen (Float) (Variable Register Float)	**#VRF**	Register-ID, Wert, Wert1 [ID+1], ...
Objektstring umwandeln (Variable Register Object)	**#VRO**	Register-ID, Obj-ID, Obj-ID1[ID+1], ...
Stringregister in Register umwandeln (Variable Register dElimiter)	**#VRE**	Register-ID Start, String-ID Quelle, Seperator, Register-ID Anzahl
Register in RTC-RAM schreiben (Variable Register rtc Write)	**#VRW**	ID, Register-ID, Register-ID1, ...
Register aus RTC-RAM lesen (Variable Register rtc Read)	**#VRR**	ID, Register-ID, Register-ID1, ...
Stringregister als Kalkulation in Register umwandeln (Integer) (Variable Calculate Integer)	**#VCI**	Register-ID, String-ID, String-ID1[ID+1], ...
Stringregister als Kalkulation in Register umwandeln (Float) (Variable Calculate Float)	**#VCF**	Register-ID, String-ID, String-ID1[ID+1], ...
Register senden (Variable Register Send)	**#VRG**	Register-ID, ...
Register sortieren (Variable Quicksort Register)	**#VQR**	Register-ID Start, Register-ID Ende
Register mischen (Variable Mix Register)	**#VMR**	Register-ID Start, Register-ID Ende

Array

Array definieren (Integer) (Variable Array Integer)	**#VAI**	Array-ID, Anzahl, Typ(0)
Array definieren (Float) (Variable Array Float)	**#VAF**	Array-ID, Anzahl, Typ(0)
Array löschen (Speicher freigeben) (Variable Array Delete)	**#VAD**	Array-ID
Array füllen (Variable Array Set)	**#VAS**	Array-ID, Wert(0, alle Elemente), element index, ...
Array-Elementen Werte zuweisen (mit Index) (Variable Array Value)	**#VAV**	Array-ID, Index, Wert, Wert[Index+1], ...
Array-Elementen Werte zuweisen (mit aktuellem Schreib-Pointer) (Variable Array Write)	**#VAW**	Array-ID, Wert 1, Wert 2, ...
Schreib- und/oder Lese-Pointer setzen (Variable Array Pointer)	**#VAP**	Array-ID, SchreibPointer, LesePointer(-1)

Array sortieren (Variable Quicksort Arrays)	#VQA	Array-ID, StartIndex, EndIndex(letzter Index)
Array mischen (Variable Mix Arrays)	#VMA	Array-ID, StartIndex, EndIndex(letzter Index)

Stringfiles / Mehrsprachigkeit

Es werden in 4 unterschiedlichen Sprachen "Hallo World" platziert. Es muss darauf geachtet werden, dass der ausgewählte Font alle notwendigen Zeichen unterstützt. Im folgenden Beispiel wurde "Arial Unicode MS" eingesetzt. Die unten stehende Befehle setzten voraus, dass die Stringfiles (Chinese.txt, English.txt, Cyrillic.txt und German.txt) bereits auf der SD-Card im Projektpfad im Unterordner Strings vorhanden sind:


```
...
#VFL
<P:string/German.txt>

#SSP1,1,10,10,7,!HELLO!;
#VFD

#VFL
<P:string/English.txt>
#SSP2,1,10,50,
```

```
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
...
```

Stringfile laden

#VFL	<StringfileName>

Einen Satz von Strings laden. Maximal können gleichzeitig 1000 Strings aus 8 verschiedenen Dateien geladen sein.

Stringfile löschen

#VFD	<StringfileName> (alle)

Einen Satz von Strings bzw. alle löschen. Die Dateien bleiben physikalisch im FLASH erhalten, sodass sie erneut geladen werden können

Anzahl an geladenen Strings senden

#VFC	

Stellt die Anzahl an geladenen Strings in den Sendepuffer. Die Rückmeldung ist folgendermaßen aufgebaut:

ESC	V	F	C	Anzahl	...
$1B	$56	$53	$43	16-Bit Wert	

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

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

Stringregister

Stringregister setzen

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

Der Befehl speichert den **String** im Registersatz (**String-ID** [0...499]) ab.

String-ID	Wert
0	"Hello World"
1	"Test Hello World"
2	"Hello World Test"

```
...
#VSS 0,"Hello World";"Test "S0;S0" Test";
...
```

Stringregister ab Position setzen

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

Der String der Speicherstelle **String-ID** wird ab der Postion **Offset** gelöscht und die neuen Daten (**"New String"**) werden angefügt.

String-ID	Wert
0	"Hello Test"
1	...
2	...

```
...
#VSS 0,"Hello World";
#VSP 0,6,"Test";
...
```

Stringregister ab Position ersetzen

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

Der String der Speicherstelle **String-ID** wird ab der Postion **Offset** mit den neuen Daten (**"New String"**) ersetzt.

String-ID	Wert
0	"Hello Test"
1	...
2	...

```
...
#VSS 0,"Hello World";
#VSR 0,6,"Test";
...
```

Teilstring aus Stringregister ausschneiden und verschieben

#VST	String-ID, Offset, Anzahl(bis Ende)

Den "linken" Teil des Stringregisters löschen und den Teil von Offset bis Offset+Anzahl nach vorne schieben. Ist Anzahl negativ, wird Anzahl als zweiter Offset genommen. Es handelt sich also dann um eine Bereichsangabe.

String-ID	Wert
0	"World"
1	...
2	...

```
...
#VSS 0,"Hello World";
#VST 0,6,5
...

...
#VSS 0,"Hello World";
#VST 0,6-10
...
```

Teilstring aus Stringregister kopieren

#VSC	String-ID Ziel, String-ID Quelle, Offset (0), Anzahl(bis Ende)

Aus dem String (**String-ID Quelle**) einen Teilstring beginnend mit dem **Offset** und der Länge **Anzahl** kopieren und in ein anderes Stringregister (**String-ID Ziel**) einfügen.

String-ID	Wert
0	"Hello World"
1	"World"
2	...

```
...
#VSS 0,"Hello World";
#VSC 1,0,6,5
...

...
#VSS 0,"Hello World";
#VSC 1,0,6-10
```

```
...
```

Stringregister in Teilstrings aufteilen

#VSE	String-ID Ziel Start, String-ID Quelle, Seperator, Register-ID (=String-ID Ziel Start)

Der String (**String-ID Quelle**) wird in Teilstrings zerlegt. Die Teilstrings werden ab **String-ID Ziel Start** abgelegt. Die Anzahl der Teilstrings wird in **Register-ID** abgelegt. Der Parameter **Seperator** gibt das Trennzeichen an.

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

Register-ID	Wert
10	3

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

Stringregister mit Zeit/Datum setzen

#VSD	String-ID, "Datumsformat"; date (aktuelle Zeit)

Im Stringregister wird die Zeit als formatierter String abgelegt. Die Darstellungsweise richtet sich nach dem **Datumsformat**. Der Aufbau ist im Unterkapitel Datumsformate näher beschrieben.

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

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

Formatiertes Stringregister setzen

#VSF	String-ID, "Formatstring"; Wert1, Wert2, ..., WertN

Eine formatierten Zeichenkette wird im Stringregister abgelegt (**String-ID**). Wird der Variablen Satz wiederholt, wird der Formatstring erneut angewendet und in **String-ID**+1 abgelegt

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

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

Objektsstrings auslesen

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

Objektstrings (**Obj-ID**) werden im Stringregister (**String-ID**) abgelegt. Diese Funktion wird vor allem für EditBoxen verwendet.

String-ID	Wert
0	...
1	"edit me"
2	...

```
...
#SED 1,"edit me"      /**Default text for
EditBox
#VSO 1,1
...
```

Stringregister senden (ASCII)

#VSA	String-ID, ...

Den Inhalt des Stringregisters (ASCII formatiert) in den Sendepuffer stellen. Die Rückmeldung ist folgendermaßen aufgebaut:

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

```
1Bh 56h 53h 41h 00h 00h 03h 00h 73h 74h 72h
```

```
...
#VSS0,"str";
#VSA 0
...
```

Stringregister senden (Unicode)

#VSU	String-ID, ...

Den Inhalt des Stringregisters (Unicode formatiert) in den Sendepuffer stellen. Die Rückmeldung ist folgendermaßen aufgebaut:

ESC	V	S	U	String-ID	Länge	Char 1	Char 2	...	Char n	...
$1B	$56	$53	$55	16-Bit Wert	16-Bit Wert	16-Bit Wert	16-Bit Wert	16-Bit Wert	16-Bit Wert	

```
1Bh 56h 53h 55h 00h 00h 03h 00h 73h 00h 74h 00h 72h 00h
```

```
...
#VSS0,"str";
#VSU 0
...
```

Stringregister sortieren

VQS	String-ID Start, String-ID Ende, Anzahl (0), Offset(0)

Der Bereich des Stringregisters (**String-ID Start bis String-ID Ende**) wird sortiert. **Anzahl** gibt den Bereich an, welcher für die Sortierung beachtet wird, bei **Anzahl**=0 wird die gesamte Länge untersucht. **Offset** gibt die Stelle im String an ab der die Sortierung beginnt.

String-ID	Wert davor	Wert danach
0	"Sort"	"Entry"

```
...
#VSS 0,"Sort";"Test";"Entry";"Exit";
```

1	"Test"	"Exit"
2	"Entry"	"Sort"
3	"Exit"	"Test"

```
#VQS 0,3,2,0
...
```

Codes im Stringregister sortieren

VQC	String-ID, Richtung (1), Anzahl (0), Offset (0)

Codes innerhalb des Stringregisters (**String-ID**) werden sortiert. **Anzahl** gibt den Bereich an, welcher für die Sortierung beachtet wird, bei **Anzahl**=0 wird die gesamte Länge untersucht. **Offset** gibt die Stelle im String an ab der die Sortierung beginnt. Zudem kann die Richtung angegeben werden:

Richtung	
0	Absteigend
1	Aufsteigend

String-ID	Wert
0	...
1	" HWdellloor"
2	...

```
...
#VSS 1,"Hello World"
#VQC 1
...
```

Letzte Fehlermeldung in Stringregister

#VSL	String-ID, Löschen(1)

Die Fehlermeldungen aus dem Terminal zusätzlich in ein Stringregister (**String-ID**) speichern. Der Parameter **Löschen** gibt das Löschverhalten der Fehlermeldung an:

Löschen	
0	Nicht löschen
1	Löschen

Stringregister mischen

#VMS	String-ID Start, String-ID Ende

Der Inhalt der Register bleibt bestehen, nur die String-ID ändert sich. Damit ist eine neue Zuordnung String-ID ⇔ Inhalt vorhanden.

Codes im Stringregister mischen

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

Der Inhalt eines Stringregister (**String-ID**) wird zufällig vertauscht.
Anzahl gibt die Anzahl der Stellen an (= 0 kompletter String), **Offset** den Startpunkt innerhalb des Registers.

Register

Register setzen (Integer)

#VRI	Register-ID, Wert, Wert1 [ID+1], ...

Der Befehl speichert im Registersatz (**Register-ID** [0...499]) einen Integerwert (32 Bit).

Register-ID	Wert
0	10
1	42
2	-8

```
...
#VRI 0,10,42,-8
...
```

Register setzen (Float)

#VRF	Register-ID, Wert, Wert1 [ID+1], ...

Der Befehl speichert im Registersatz (**Register-ID** [0...499]) einen Floatwert (32 Bit).

Register-ID	Wert
0	10.25
1	42.39
2	-8.19

```
...
#VRF 0,10.25,42.39,-8.19
...
```

Objektstring umwandeln

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

Objektstrings werden in Register abgelegt. Der Objektstring wird dabei in einen Zahlenwert gewandelt (automatisch passend als Integer oder Float). Diese Funktion wird vor allem für EditBoxen verwendet.

Register-ID	Wert
0	...
1	42.5
2	...

```
...
#SED 1,"42.5"      /**Default text for EditBox
#VRO 1,1
...
```

Stringregister in Register umwandeln

#VRE	Register-ID Start, String-ID Quelle, Seperator, Register-ID Anzahl

Numerischen String (**String-ID Quelle**) in Register (**Register-ID Start**) wandeln. **Separator** gibt das Trennzeichen zwischen den Werten an. Im optionalen Parameter **Register-ID Anzahl** wird die Anzahl gültiger Werte nach der Wandlung angegeben

Register-ID	Wert
0	10

```
...
#VSS 0,"10,42.39,-8";
#VRE 0,0,?,,10
```

1	42.39	
2	-8	...
10	3	

Register in RTC-RAM schreiben

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

Eine **Register-ID** im RAM der RTC puffern. **ID** [0...7] gibt den Speicherplatz an. Der Wert bleibt auch nach dem Ausschalten des Moduls erhalten. Eine RTC muss vorhanden sein (Achtung EA uniTFTs020-ATC und EA uniTFTs028-ATC)

Register aus RTC-RAM lesen

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

Aus dem RTC-RAM (**ID**) einen Wert zurück lesen und in Register (**Register-ID**) übernehmen. Eine RTC muss vorhanden sein (Achtung EA uniTFTs020-ATC und EA uniTFTs028-ATC)

Stringregister als Kalkulation in Register umwandeln (Integer)

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

Den Inhalt eines Stringregisters als Kalkulationsstring interpretieren. Das Ergebnis wird im Register (**Register-ID**) abgelegt

Register-ID	Wert
0	-8
1	50
2	42

```
...
#VSS 0,"R0+R1";
#VCI 2,0
...
```

Stringregister als Kalkulation in Register umwandeln (Float)

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

Den Inhalt eines Stringregisters als Kalkulationsstring interpretieren. Das Ergebnis wird im Register (**Register-ID**) abgelegt

Register-ID	Wert
0	-8.21
1	50.89
2	42.68

```
...
#VSS 0,"R0+R1";
#VCF 2,0
...
```

Register senden

#VRG	Register-ID, ...

Den Inhalt des Registers in den Sendepuffer stellen. Die Rückmeldung ist folgendermaßen aufgebaut:

ESC	V	R	G	Register-ID	Typ	Wert	...
$1B	$56	$52	$47	16-Bit Wert	16-Bit Wert	32-Bit Wert	

```
1Bh 56h 52h 47h 00h 00h 46h 00h 29h 5Ch 03h C1h
```

```
...
#VRF 0,...
#VRG 0
...
```

Register sortieren

#VQR	Register-ID Start, Register-ID Ende

Der Bereich der Register (**Register-ID Start bis Register-ID Ende**) werden sortiert.

Register-ID	Wert davor	Wert danach
0	2	-5
1	8	2
2	4	4
3	-5	8

```
...
#VRI 0,2,8,4,-5
#VQR 0,3
...
```

Register mischen

#VMR	Register-ID Start, Register-ID Ende

Der Inhalt der Register bleibt bestehen, nur die Register-ID ändert sich. Damit ist eine neue Zuordnung Register-ID ⇔ Inhalt vorhanden.

Array

Array definieren (Integer)

#VAI	Array-ID, Anzahl, Typ(0)

Der Befehl definiert ein Integer Array (**Array-ID** [0...499]) mit der gegebenen **Anzahl** an Einträgen. Die maximale Länge des Arrays bzw. ob das Arrays mit der gewünschten Größe angelegt werden konnte kann mit der Kalkulation arE() überprüft werden. Der **Typ** gibt das Verhalten beim Schreiben am Ende des Arrays an.

Typ	
0	Am Ende abbrechen
1	Am Ende umbrechen (Ringbuffer)

Array definieren (Float)

#VAF	Array-ID, Anzahl, Typ

Der Befehl definiert ein Float Array (**Array-ID** [0...499]) mit der gegebenen **Anzahl** an Einträgen. Die maximale Länge des Arrays bzw. ob das Arrays mit der gewünschten Größe angelegt werden konnte kann mit der Kalkulation arE() überprüft werden. Der **Typ** gibt das Verhalten beim Schreiben am Ende des Arrays an.

Typ	
0	Am Ende abbrechen
1	Am Ende umbrechen (Ringbuffer)

Array löschen (Speicher freigeben)

#VAD	Array-ID

Der Befehl löscht ein Array (**Array-ID** [0...499]) und gibt den Speicher frei.

Array füllen

#VAS	Array-ID, Wert(0), element index, ...

Der Befehl füllt alle Elemente des Arrays (**Array-ID** [0...499]) mit dem gegeben **Wert**. Ist kein **Wert** angegeben wird das komplette Array gefüllt. **Element index** ermöglicht das füllen bestimmter Elemente des Arrays mit dem angegebenen Wert.

Array-Elementen Werte zuweisen (mit Index)

#VAV	Array-ID, Index, Wert, Wert[Index+1], ...

Der Befehl weist Array-Elementen, beginnend mit dem Array-**Index** neue **Werte** zu.

Array-Elementen Werte zuweisen (mit aktuellem Schreib-Pointer)

#VAW	Array-ID, Wert 1, Wert 2, ...

Der Befehl weist Array-Elementen, beginnend mit dem aktuellen Schreib-Pointer neue **Werte** zu.

Schreib- und/oder Lese-Pointer setzen

#VAP	Array-ID, SchreibPointer, LesePointer(-1)

Der Befehl setzt den Schreib- und/oder Lese-Pointer des Arrays (**Array-ID** [0...499]). Soll der Pointer unverändert bleiben muss der jeweilige Parameter auf -1 gesetzt werden.

Array sortieren

#VQA	Array-ID, StartIndex, EndIndex(letzter Index)

Die Werte des Arrays (**Array-ID** [0...499]) werden im angegebenen Bereich (**StartIndex** bis **EndIndex**) sortiert.

Array mischen

#VMA	Array-ID, StartIndex, EndIndex(letzter Index)

Der Befehl mischt die Werte des Arrays im angegebenen Bereich (**StartIndex** bis **EndIndex**). Die Werte bleiben dabei unverändert. Nur die Reihenfolge (Indexes) wird angepasst. Damit ist eine neue Zuordnung der Array-Indexes ⇔ Werte vorhanden....

I/O Port #H

Das Modul verfügt über 8 I/O Portleitungen, welche auf bis zu 136 erweitert werden können. Bei Änderungen der Porteingangspins können Makros gestartet werden, siehe #MHP, und #MHB.

Port-Zugriff (8 I/Os)

Port definieren (Eingang/Ausgang) (Hardware Port Control)	**#HPC**	Port, I/O, I/O [Port+1], ...
Port Ausgang setzen (Hardware Port Write)	**#HPW**	Port, Zustand, Zustand [Port+1], ..
Port Eingänge lesen (Hardware Port Read)	**#HPR**	Port (0), Anzahl(1)
Portinformation senden (Hardware Port Information)	**#HPI**	

Pin-Zugriff

Portpin definieren (Eingang/Ausgang) (Hardware Bit Control)	**#HBC**	Portpin, I/O, I/O [Port+1], ...
Portpin Ausgang setzen (Hardware Bit Write)	**#HBW**	Portpin, Zustand, Zustand [Port+1], ..
Portpin Eingänge lesen (Hardware Bit Read)	**#HBR**	Portpin(0), Anzahl(8)

Port-Zugriff (8 I/Os)

Port definieren (Eingang/Ausgang)

#HPC	Port, I/O, I/O [Port+1], ...

Der Befehl definiert für einen ganzen **Port** [0...16] bitweise die Richtung (**I/O**) der einzelnen Portpins:

I/O	
0	Ausgang
1	Eingang

	Port 0							
	0	1	2	3	4	5	6	7
Eingang	●			●		●	●	●
Ausgang		●	●		●			

```
...
#HPC 0,$E9
...

...
#HPC 0,%11101001
...
```

Port Ausgang setzen

#HPW	Port, Zustand, Zustand [Port+1], ..

Der Befehl setzt für einen ganzen **Port** bitweise den **Zustand** der Ausgänge.

Zustand	
0	Low
1	High

	Port 0							
	0	1	2	3	4	5	6	7
Zustand	○	○	●	○	●	●	○	●

```
...
#HPC 0,0
#HPW 0,$B4
...

...
#HPC 0,0
#HPW 0,%10110100
...
```

Port Eingänge lesen

#HPR	Port (0), Anzahl(1)

Der Befehl stellt den Zustand eines oder mehrerer (**Anzahl**) Ports (beginnend mit **Port**) in den Sendepuffer. Die Rückmeldung ist folgendermaßen aufgebaut:

ESC	H	P	R	Port	Anzahl	Zustand 1	Zustand 2	...
$1B	$48	$50	$52	8-Bit Wert	8-Bit Wert	8-Bit Wert	8-Bit Wert	

```
1Bh 48h 50h 52h 00h 02h AAh FFh
```

```
...
#HPR 0,2
...
```

Siehe auch port(a)

Portinformation senden

#HPI	

Gibt an welche der 16 möglichen Portbausteine angeschlossen (=1) sind und stellt diese Information in den Sendepuffer. Die Rückmeldung ist folgendermaßen aufgebaut:

ESC	H	P	I	Rückmeldung (1 Bit po Baustein)
$1B	$48	$50	$49	16-Bit Wert

```
1Bh 48h 50h 49h 01h 00h
```

```
...
#HPI
...
```

Der interne Port wird nicht zurückgegeben. Es geht lediglich um die extern angeschlossenen PortExpanderbauseteine

Pin-Zugriff

Portpin definieren (Eingang/Ausgang)

#HBC	Portpin, I/O, I/O [Port+1], ...

Der Befehl definiert für den **Portpin** die Richtung (**I/O**):

I/O	
0	Ausgang
1	Eingang

Portpin Ausgang setzen

#HBW	Portpin, Zustand, Zustand [Port+1], ..

Der Befehl setzt für den **Portpin** den **Zustand** des Ausgangs.

Zustand	
0	Low
1	High
2	Invert

Portpin Eingänge lesen

#HBR	Portpin(0), Anzahl(8)

Der Befehl stellt den Zustand eines oder mehrerer (**Anzahl**) Portpins (beginnend mit **Portpin**) in den Sendepuffer. Die Rückmeldung ist folgendermaßen aufgebaut:

ESC	H	B	R	Portpin	Anzahl	Zustand 1	Zustand 2	...
$1B	$48	$42	$52	8-Bit	8-Bit	8-Bit	8-Bit	

				Wert	Wert	Wert	Wert	

```
1Bh 48h 42h 52h 00h 04h 01h 00h 01h 00h
```

```
...
#HBR 0,4
...
```

Siehe auch bit(a)

Analog Input #H

Befehlsgruppe um den Analogeingang des Moduls zu parametrisieren und auszulesen. Das Modul hat vier 12-Bit Analogeingänge. Bei Änderungen des Analogeingangs kann ein Makro gestartet werden, siehe #MHA.

Analogeingang lesen (Hardware Analog Read)	**#HAR**	Kanal(0), Anzahl(4)
Grenzen/ Schwellwert festlegen (Hardware Analog Limit)	**#HAL**	Kanal, Grenze1, Grenze2, Grenze1 [Kanal+1], Grenze2 [Kanal+1], ...
Hysterese einstellen (Hardware Analog Hyteresis)	**#HAH**	Kanal, Hysterese , Hystese [Kanal+1], ...

Analogeingang lesen

#HAR	Kanal(0), Anzahl(4)

Der Befehl liest einen oder mehrere (**Anzahl**) Analogkanäle (beginnend mit **Kanal** [0...4])aus und stellt den Wert in den Sendepuffer. Die Rückmeldung ist folgendermaßen aufgebaut:

ESC	H	A	R	Kanal	Anzahl	Wert 1	Wert 2	...
$1B	$48	$41	$52	8-Bit Wert	8-Bit Wert	16-Bit Wert	16-Bit Wert	

```
1Bh 48h 41h 52h 00h 02h 5Ch 0Dh B8h 06h
```

```
...
#HAR 0,2
...
```

Siehe auch bit(a)

Grenzen/ Schwellwert festlegen

#HAL	Kanal, Grenze1, Grenze2, Grenze1 [Kanal+1], Grenze2 [Kanal+1], ...

Für jeden Analogeingang (**Kanal**) können 2 Schwellwerte festgelegt werden bei denen Makros aufgerufen werden können (#MHA). Die **Grenzen** werden in ADC-Counts angegeben.

Hysterese einstellen

#HAH	Kanal, Hysterese , Hystese [Kanal+1], ...

Die **Hysterese** für den jeweiligen **Kanal** in ADC-Counts festlegen. Default ist die Hysterese für jeden Kanal 4. Erst nach überschreiten der Hysterese wird das jeweilige definierte Makro aufgerufen.

PWM Output #H

Befehlsgruppe um den PWM Output einzustellen

PWM Frequenz und DutyCycle einstellen (Hardware PWM Output)	**#HFO**	Frequenz [32-Bit], On Value (keine Änderung), Total Value (keine Änderung)
PWM DutyCycle ändern (Hardware PWM Dutycycle)	**#HFD**	On Value, Total Value (keine Änderung)

PWM Frequenz und DutyCycle einstellen

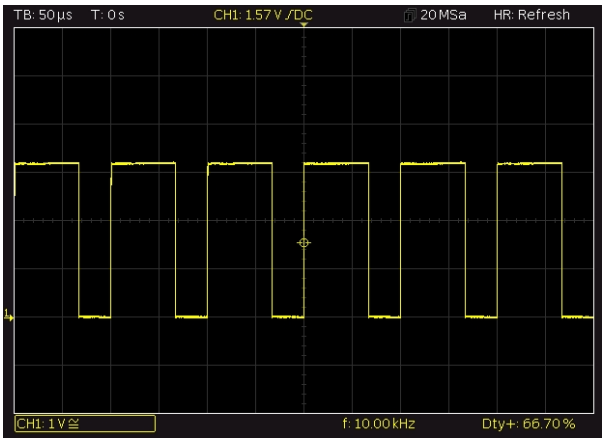
#HFO	Frequenz [32-Bit], On Value (keine Änderung), Total Value (keine Änderung)

Einstellung der PWM-**Frequenz** (32-Bit Wert) (2Hz... 1MHz).

Frequenz	
0	Permanent low
1	Permanent high

Die beiden optionalen Parameter **On Value** und **Total Value** stellen das Tastverhältnis ein:

$$\text{DutyCycle} = \frac{\text{On Value}}{\text{Total Value}}$$




```
...
#HFO 10000,2,3
...
```

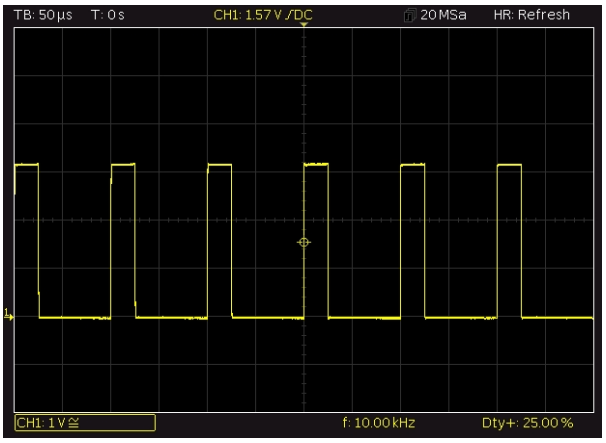
PWM DutyCycle ändern

#HFD	On Value, Total Value (keine Änderung)

Der Befehl stellt mit den beiden Parameter **On Value** und **Total Value** das Tastverhältnis ein. Die Frequenz bleibt unverändert:

$$\text{DutyCycle} = \text{On Value}$$

Total Value




```
...
#HFD 1,4
...
```

Serielle Master Schnittstellen #H

Befehlsgruppe um die 3 seriellen Schnittstellen als Master zu verwenden z.B. zum Anschluss weiterer Peripherie an das Modul wie ein Temperaturfühler.

RS232 Baudrate einstellen (Hardware Rs232 Parameter)	**#HRP**	Baudrate [32-Bit]
SPI Parameter einstellen (Hardware Spi Parameter)	**#HSP**	Frequenz, Modus, Datenreihenfolge
SPI Chip Select einstellen (Hardware Spi Chipselect)	**#HSC**	ChipSelect
I²C Parameter einstellen (Hardware I2c Parameter)	**#HIP**	Adresse, Frequenz
8-Bit (ASCII) String senden (Hardware RS232/SPI/I2c Ascii)	**#HRA** **#HSA** **#HIA**	"String";
16-Bit (Unicode) String senden (Hardware RS232/SPI/I2c Unicode)	**#HRU** **#HSU** **#HIU**	"String";
32-Bit Signed Werte senden (Hardware RS232/SPI/I2c Integer)	**#HRI** **#HSI** **#HII**	Wert, Wert1...
32-Bit Float Werte senden (Hardware RS232/SPI/I2c floaT)	**#HRT** **#HST** **#HIT**	Wert, Wert1...
Binärdaten senden (Hardware RS232/SPI/I2c Send binary)	**#HRS** **#HSS** **#HIS**	Anzahl, Daten...
Binärdaten aus Register senden (Hardware RS232/SPI/I2c send values)	**#HRX** **#HSX** **#HIX**	Typ, Register-ID, Anzahl(1)
Binärdaten aus Array senden (Hardware RS232/SPI/I2c send array)	**#HRY** **#HSY** **#HIY**	Typ, Array-ID, StartIndex(0), Anzahl(alle Elemente)
Datei senden (Hardware RS232/SPI/I2c send File)	**#HRF** **#HSF** **#HIF**	<Filename>
Daten empfangen und in den Sendepuffer stellen (Hardware RS232/SPI/I2c Receive to	**#HRR**	Anzahl [32-Bit] (max 1024)

buffer)	#HSR #HIR	
8-Bit Daten empfangen und in ein Stringregister schreiben (Hardware RS232/SPI/I2c Bytes to string)	#HRB #HSB #HIB	String-ID, Anzahl (max 250)
16-Bit Daten empfangen und in ein Stringregister schreiben (Hardware RS232/SPI/I2c Words to string)	#HRW #HSW #HIW	String-ID, Anzahl (max 250)
Binäre Daten empfangen und in ein Register schreiben (Hardware RS232/SPI/I2c Values to register)	#HRV #HSV #HIV	Typ, Register-ID, Anzahl(1)
Binäre Daten empfangen und in ein Array schreiben (Hardware RS232/SPI/I2c Values to array)	#HRZ #HSZ #HIZ	Typ, Array-ID, StartIndex(0), Anzahl(alle Elemente)

Das entsprechende Interface kann nach einem dieser Befehle nicht mehr als Slave-Schnittstelle genutzt werden um Kommandos entgegen zunehmen. Es bekommt nun die Masterfunktion zum Steuern externer Peripherie.

RS232 Baudrate einstellen

#HRP	Baudrate

Mit dem Befehl wird die **Baudrate** (32-Bit Wert) eingestellt:

Baudrate	Fehler
9600	+0.04
19200	-0.08
38400	+0.16
57600	-0.08
115200	+0.64
230400	-0.80
460800	+2.08
921600	-3.68

SPI Parameter einstellen

#HSP	Frequenz, Modus, Datenreihenfolge

Mit dem Befehl wird die **Frequenz** (15600...1000000 Hz), der SPI-**Modus** (0..3) und die **Datenreihenfolge** der Master

SPI Schnittstelle eingestellt

Datenreihenfolge	
0	MSB first
1	LSB first

SPI Chip Select einstellen

#HSC	ChipSelect

Der Befehl definiert die ChipSelect Einstellung:

ChipSelect	
0	Low
1	High
2	Automatisch (low aktiv)

I²C Parameter einstellen

#HIP	Adresse, Frequenz

Mit dem Befehl wird die **Adresse** des anzusteuernden Busteilnehmers und die **Frequenz** (3900...1000000 Hz) eingestellt.

8-Bit (ASCII) String senden

#HRA	(RS232)	"String";
#HSA	(SPI)	
#HIA	(I²C)	

Der Befehl sendet einen **String** oder einzelne Codes als ASCII Wert(e) (8 Bit pro Zeichen).

16-Bit (Unicode) String senden

#HRU	(RS232)	"String";
#HSU	(SPI)	
#HIU	(I²C)	

Der Befehl sendet einen **String** oder einzelne Codes als Unicode Wert(e) (16 Bit pro Zeichen).

32-Bit Signed Werte senden

#HRI	(RS232)	Wert, Wert1...
#HSI	(SPI)	
#HII	(I²C)	

Der Befehl sendet einen oder mehrere 32-Bit Signed Integer **Wert(e)** (little endian).

32-Bit Float Werte senden

#HRT	(RS232)	Wert, Wert1...
#HST	(SPI)	
#HIT	(I²C)	

Der Befehl sendet einen oder mehrere 32-Bit Float **Wert(e)** (little endian).

Binärdaten senden

#HRS	(RS232)	Anzahl, Daten...
#HSS	(SPI)	
#HIS	(I²C)	

Der Befehl sendet eine **Anzahl** an **Daten** direkt über die Master Schnittstelle. Die Daten werden direkt übernommen und gesendet, eine Interpretation, wie Kalkulationsinterpretation, erfolgt nicht.

Binärdaten aus Register senden

#HRX	(RS232)	Typ, Register-ID, Anzahl(1)
#HSX	(SPI)	
#HIX	(I²C)	

Der Befehl sendet eine **Anzahl** an Registereinträge (**Register-ID**) binär über die Master Schnittstelle.

Typ			
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	

Binärdaten aus Array senden

#HRY	(RS232)	Typ, Array-ID, StartIndex(0), Anzahl(alle Elemente)
#HSY	(SPI)	
#HIY	(I²C)	

Der Befehl sendet eine **Anzahl** an Array-Elemente (**Array-ID**), beginnend mit dem Start Index, binär über die Master Schnittstelle.

Typ			
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	

Datei senden

#HRF (RS232) #HSF (SPI) #HIF (I²C)	<Filename>

Der Befehl sendet eine Datei (**<Filename>**) über die Master Schnittstelle.

Daten empfangen und in den Sendepuffer stellen

#HRR (RS232) #HSR (SPI) #HIR (I²C)	Anzahl [32-Bit] (max 1024)

Der Befehl liest eine **Anzahl** (32-Bit Wert) an Daten aus dem Master Empfangspuffer und stellt sie in den Sendepuffer. Die Rückmeldung ist folgendermaßen aufgebaut:

ESC	H	R/S/I	R	Länge	Daten 1	Daten 2	...	Daten n	...
$1B	$48	$52/$53/$49	$52	32-Bit	8-Bit Wert	8-Bit Wert	8-Bit Wert	8-Bit Wert	

Siehe auch mstRA()

8-Bit Daten empfangen und in ein Stringregister schreiben

#HRB (RS232) #HSB (SPI) #HIB (I²C)	String-ID, Anzahl (max 250)

Der Befehl liest eine **Anzahl** an Daten aus dem Master Empfangspuffer und schreibt sie in das angegebene Stringregister (**String-ID**).

Siehe auch mstRA()

16-Bit Daten empfangen und in ein Stringregister schreiben (ab V1.2)

#HRW	(RS232)	String-ID, Anzahl (max 250)
#HSW	(SPI)	
#HIW	(I²C)	

Der Befehl liest eine **Anzahl** an Daten aus dem Master Empfangspuffer und schreibt sie in das angegebene Stringregister (**String-ID**).

Siehe auch mstRA()

Binäre Daten empfangen und in ein Register schreiben (ab V1.4)

#HRV	(RS232)	Typ, Register-ID, Anzahl(1)
#HSV	(SPI)	
#HIV	(I²C)	

Der Befehl liest eine **Anzahl** an Daten aus dem Master Empfangspuffer und schreibt sie in das angegebene Register (**Register-ID**).

Typ			
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	

Siehe auch mstRA()

Binäre Daten empfangen und in ein Array schreiben

#HRZ (RS232) **#HSZ** (SPI) **#HIZ** (I^2C)	Typ, Array-ID, StartIndex(0), Anzahl(alle Elemente)

Der Befehl liest eine **Anzahl** an Daten aus dem Master Empfangspuffer und schreibt sie, beginnend mit dem Start Index in das angegebene Array (**Array-ID**). Vor dem Empfangen muss ein Array definiert werden (siehe #VAI, #VAF).

Typ			
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	

Sound #H

Befehlsgruppe um eine Tonfolge abzuspielen

Sound abspielen (Hardware Tone Notes)	**#HTN**	"SoundString"
Sound stoppen (Hardware Tone Stop)	**#HTS**	

Sound / Tonfolge abspielen

#HTN	"SoundString"

Der Befehl spielt den angegebene **SoundString** ab. Unten finden Sie die Möglichkeiten der Töne.

Sound stoppen

#HTS	

Der Befehl stoppt das zur Zeit abgespielte Soundfile.

Töne

Der Aufbau des Notenstrings besteht aus Teiler für nachfolgende Noten/Pausen, eventueller Halbtonerhöhung für die nächste Note und den Noten selbst:

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

Uhrzeit #W

Befehlsgruppe um mit der internen (EA uniTFTs035-ATC / EA uniTFTs043-ATC) bzw. extern verbundenen RTC zu arbeiten.

Uhrzeit setzen (Watch Time Date set)	**#WTD**	Stunde, Minute (aktueller Wert), Sekunde (aktueller Wert), Tag (aktueller Wert), Monat (aktueller Wert), Jahr (aktueller Wert), Abgleich (0)
Bestehende Gruppe als Uhr definieren (Watch Group Clock)	**#WGC**	Group-ID, Stundenzeiger-ID, Minutenzeiger-ID, Sekundenzeiger-ID(none)
Ausgabeformat für RTC definieren (Watch Define Format)	**#WDF**	"Datumsformat"
Namen für Monate definieren (Watch Define Month strings)	**#WDM**	"JAN";"FEB";"MAR";"APR";"MAI";"JUN";"JUL";"AUG";"SEP";"OCT";"NOV";"DEC"
Namen für Wochentage definieren (Watch Define Day strings)	**#WDW**	"SO";"MO";"DI";"MI";"DO";"FR";"SA"
Uhrzeit senden (ASCII) (Watch Send Ascii)	**#WSA**	"Datumsformat"; date (aktuelle Zeit)
Uhrzeit senden (Unicode) (Watch Send Unicode)	**#WSU**	"Datumsformat"; date (aktuelle Zeit)
Uhrzeit senden (Binär) (Watch Send Binary)	**#WSB**	"Datumsformat"; date (aktuelle Zeit)
Basisjahr für Uhrzeitberechnung definieren (Watch Define base Year)	**#WDY**	Jahr

Uhrzeit setzen

#WTD	Stunde, Minute (aktueller Wert), Sekunde (aktueller Wert), Tag (aktueller Wert), Monat (aktueller Wert), Jahr (aktueller Wert), Abgleich (0)

Der Befehl setzt die aktuelle Uhrzeit. Wird der optionale Parameter **Abgleich** auf 1 gesetzt, wird der interne Quarz beim nächsten setzen der Zeit (**Abgleich** muss ebenfalls 1 sein) kalibriert.

Bestehende Gruppe als Uhr definieren

#WGC	Group-ID, Stundenzeiger-ID, Minutenzeiger-ID, Sekundenzeiger-ID(none)

Der Befehl wandelt eine bestehende Gruppe in eine Uhr um. **Stundenzeiger-ID** gibt die Obj-ID für den Stundenzeiger, **Mintutenzeiger-ID** die Obj-ID für den Minutenzeiger, **Sekundenzeiger-ID** die Obj-ID für den Sekundenzeiger an.

```
...
#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
...
```

Ausgabeformat für RTC definieren

#WDF	"Datumsformat"

Der Befehl ändert das Datumsformat.

Namen für Monate definieren

#WDM	"JAN";"FEB";"MAR";"APR";"MAI";"JUN";"JUL";"AUG";"SEP";"OCT";"NOV";"DEC"

Mit dem Befehl können 12 einzelne Strings für die Monatsnamen festgelegt werden.

Namen für Wochentage definieren

#WDW	"SO";"MO";"DI";"MI";"DO";"FR";"SA"

Mit dem Befehl können 7 einzelne Strings für den Wochentagesnamen (beginnend mit Sonntag) festgelegt werden.

Uhrzeit senden (ASCII)

#WSA	"Datumsformat"; date (aktuelle Zeit)

Der Befehl stellt Datum und Uhrzeit als ASCII String in den Sendepuffer. Die Rückmeldung ist folgendermaßen aufgebaut:

ESC	W	S	A	ASCII-String	Absc hluss	...
$1B	$57	$53	$41		$00	

Siehe auch year(), month(), day(), weekday(), hour(), minute(), second()

Uhrzeit senden (Unicode)

#WSU	"Datumsformat"; date (aktuelle Zeit)

Der Befehl stellt Datum und Uhrzeit als Unicode String in den Sendepuffer. Die Rückmeldung ist folgendermaßen aufgebaut:

ESC	W	S	U	Unicode-	Absc	...

				String	hluss	
$1B	$57	$53	$55		$00	

Siehe auch year(), month(), day(), weekday(), hour(), minute(), second()

Uhrzeit senden (Binär)

#WSB	"Datumsformat"; date (aktuelle Zeit)

Der Befehl stellt Datum und Uhrzeit als signed 32-Bit Wert in den Sendepuffer. Die Rückmeldung ist folgendermaßen aufgebaut:

ESC	W	S	B	Stunde	Minute	Sekunde	Tag	Monat	Jahr	Wochentag	...
$1B	$57	$53	$42	16-Bit Wert	16-Bit Wert	16-Bit Wert	16-Bit Wert	16-Bit Wert	16-Bit Wert	16-Bit Wert	

Siehe auch year(), month(), day(), weekday(), hour(), minute(), second()

Basisjahr für Uhrzeitberechnung definieren (ab V1.3)

#WDY	Jahr

Der Befehl ändert das Basisjahr für die Uhrzeitberechnung Mögliche Werte sind 1970,1980,1990,2000,2010,2020,2030. Der Wertebereich ist -68 bis +67 Jahre. Die voreingestellte Sekundenzählung beginnt am 1.1.2000 um 0:0:0 Uhr. Damit ist der mögliche Bereich von 1932 bis Ende 2067.

Datumsformate

Format	
%[]h	Stunde
%[]m	Minute
%[]s	Sekunde
%[]D	Tag
%[]M	Monat
%[]Y	Jahr
%[]W	Wochentag (String)
%[]N	Monat (String)

Optional []

0	Zwei Digits mit führender 0 (Default)
1	Minimum 1 Digits ohne führendem Zeichen
2	Zwei Digits mit führendem Leerzeichen
4	Vier Digits (Default für Jahr)

Für Wochentag und Monat (String)

Optional []	
0-9	x Zeichen aus dem Wochen-Monatsstring anzeigen

Beispiele	
`"%h:%m:%s";`	09:25:04
`"%D.%M.%Y";`	20.12.2019
`"%D %N %Y";`	20 Dezember 2019
`"%W, %D.%M.%Y";`	Freitag, 20.12.2019

Files auf dem internen Speicher #F

Befehlsgruppe um Dateizugriffe zu realisieren

Ordner

Ordner erstellen (File Directory Create)	**#FDC**	<Pfad>
Ordner löschen (File Directory Delete)	**#FDD**	<Pfad>, Löschen
Arbeitsverzeichnis setzen (File Directory Set)	**#FDS**	<Pfad>
Arbeitsverzeichnis senden (File Directory Get)	**#FDG**	
Alle Ordner und Dateien des Verzeichnisses senden (File Directory Read binary)	**#FDR**	<Pfad> (aktuelles Arbeitsverzeichnis)
Alle Ordner und Dateien des Verzeichnisses senden (ASCII) (File Directory read Ascii)	**#FDA**	<Pfad> (aktuelles Arbeitsverzeichnis)
Alle Ordner des Verzeichnisses senden (File Directory read dironly List)	**#FDL**	<Pfad> (aktuelles Arbeitsverzeichnis)

Dateien

Datei zum schreiben öffnen (File Write Open)	**#FWO**	<Dateiname>, Position [32-Bit] (Ende), Truncate(1), size(4096)
Datei schließen (Schreiboperation) (File Write Close)	**#FWC**	Zeit, Datum
Schreibposition von Datei setzen (File Write Position)	**#FWP**	Position [32-Bit]
Daten in die Datei schreiben (File Write Data binary)	**#FWD**	Anzahl [32-Bit], Binäre Daten
ASCII-String in die Datei schreiben (File Write Ascii)	**#FWA**	"String"
Unicode-String in die Datei schreiben (File Write Unicode)	**#FWU**	"String"
Registerwerte in die Datei schreiben (File Write Register)	**#FWR**	Register-ID, ...
Stringregister in die Datei schreiben (File Write Stringregister)	**#FWS**	String-ID, ...
Array in die Datei schreiben (File Write Array)	**#FWY**	Array-ID, ...
Datei zum lesen öffnen (File Read Open)	**#FRO**	<Dateiname>, Position [32-Bit] (Anfang)
Datei schließen (Leseoperation) (File Read Close)	**#FRC**	

Leseposition von Datei setzen (File Read Position)	**#FRP**	Position [32-Bit]
Daten lesen und senden (File Read Data binary)	**#FRD**	Anzahl [32-Bit] (ganze Datei)
ASCII-String lesen und in ein Stringregister schreiben (File Read Ascii)	**#FRA**	String-ID, ...
Unicode-String lesen und in ein Stringregister schreiben (File Read Unicode)	**#FRU**	String-ID, ...
Daten in ein Register laden (File Read Register)	**#FRR**	Register-ID, ...
Daten in ein Stringregister laden (File Read Stringregister)	**#FRS**	String-ID, ...
Daten lesen (8-Bit) und in ein Stringregister schreiben (File Read Bytes to stringregister)	**#FRB**	String-ID, Anzahl, Anzahl [ID+1],...
Daten lesen (16-Bit) und in ein Stringregister schreiben (File Read Words to stringregister)	**#FRW**	String-ID, Anzahl, Anzahl [ID+1],...
Daten in ein Array laden (File Read Array)	**#FRY**	Array-ID, ...
Datei löschen (File File Delete)	**#FFD**	<Dateiname>

Allgemeine Befehle

Datei Information senden (File File Info)	**#FFI**	<Pfad> (aktuelles Arbeitsverzeichnis)
Datei umbenennen (File File Rename)	**#FFR**	<Pfad>, <Neuer Dateiname>, Replace (0)
Datei kopieren (File File Copy)	**#FFC**	<Pfad>, <Neuer Pfad>, Replace (0)
Datei verschieben (File File Move)	**#FFM**	<Pfad>, <Neuer Pfad>, Replace (0)
Zeitstempel von Datei ändern (File change Timestamp)	**#FFT**	<Pfad>, Zeit, Datum
Attribute von Datei ändern (File change Attribut)	**#FFA**	<Pfad>, Attribut
Dateinamen in ein Stringregister laden (File Names Files to stringregister)	**#FNF**	<Pfad> (aktuelles Arbeitsverzeichnis), ID (1)
Unterverzeichnisse in ein Stringregister laden (File Names Directory to stringregister)	**#FND**	<Pfad> (aktuelles Arbeitsverzeichnis), ID (1)

Ordner

Ordner erstellen

#FDC	<Pfad>

Der Befehl legt einen neuen Ordner an. Der Parameter **<Pfad>** gibt den Namen und den Ort an.

```
...
#FDC <Project/NewPath>
...
```

Ordner löschen

#FDD	<Pfad>, Löschen

Der Befehl löscht einen Ordner. Der Parameter **<Pfad>** gibt den Namen und den Ort an.

Löschen	
0	Ordner mit Inhalt wird gelöscht
1	Nur der Inhalt wird gelöscht

```
...
#FDD <Project/NewPath>,0
...
```

Arbeitsverzeichnis setzen

#FDS	<Pfad>

Der Befehl setzt das aktuelle Arbeitsverzeichnis. Mit dem Pfad **</>** erreicht man das Root-Verzeichnis.

```
...
#FDS <Project>
...
```

Arbeitsverzeichnis senden

#FDG	

Der Befehl stellt das aktuelle Arbeitsverzeichnis in den Sendepuffer. Die Rückmeldung ist folgendermaßen aufgebaut:

ESC	F	D	G	Pfad	
$1B	$46	$44	$47	'String' mit $00 abgeschlossen	...

Alle Ordner und Dateien des Verzeichnisses senden

#FDR	<Pfad> (aktuelles Arbeitsverzeichnis)

Der Befehl stellt alle Ordner und Dateien des aktuellen Arbeitsverzeichnisses in den Sendepuffer. Die Rückmeldung ist folgendermaßen aufgebaut:

ESC	F	D	R	Verzeichnis- / Dateiname	Größe	Attribut	Zeit	Datum	...	Abschluss
$1B	$46	$44	$52	'String' mit $00 abgeschlossen	32-Bit Wert	8-Bit Wert	16-Bit Wert	16-Bit Wert		$00

Attribut	
$01	Schreibgeschützt
$02	Versteckt
$04	System
$20	Archiv

```
1Bh 46h 44h 52h 50h 72h 6Fh 6Ah 65h 63h 74h 00h
23h 00h 00h 00h 20h 8Fh 43h 27h 50h ... 00h
```

```
...
#FDR
...
```

Alle Ordner und Dateien des Verzeichnisses senden (ASCII)

#FDA	<Pfad> (aktuelles Arbeitsverzeichnis)

Der Befehl stellt alle Ordner und Dateien des aktuellen Arbeitsverzeichnisses als ASCII Strings in den Sendepuffer.

ESC	F	D	A	String	Abschluss	...
$1B	$46	$44	$41	Größe, Attribut, Zeit, Datum, 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
...
```

Alle Ordner des Verzeichnisses senden

#FDL	<Pfad> (aktuelles Arbeitsverzeichnis)

Der Befehl stellt alle Ordner des aktuellen Arbeitsverzeichnisses in den Sendepuffer. Die Rückmeldung ist

folgendermaßen aufgebaut:

#	F	D	L	Verzeichnisname	...	Abschluss
$1B	$46	$44	$4C	'String' mit $00 abgeschlossen		$00

Dateien

Achtung:
Flash-Speicher haben Bauartbedingt begrenzte Lösch/Schreibzyklen. Das im uniTFTs eingesetzte Speichermodul kann typischerweise 100.000 Zyklen sicher ausführen. Um Daten zu schreiben kann es sein, dass ein Speicherblock gelöscht werden muss, typischerweise werden für das Löschen 30 ms benötigt, es können aber bis zu 400 ms werden. Das ist bei der Makroabfolge zu beachten, wenn Schreibbefehle ausgeführt werden.

Datei zum schreiben öffnen

#FWO	<Dateiname>, Position [32-Bit] (Ende), Truncate(1), size(4096)

Der Befehl öffnet (nur schreiben) bzw. erstellt eine Datei. Der Parameter **Position** (32-Bit Wert) gibt die Stelle in der Datei an, an die geschrieben werden soll. Die Dateigröße (**size**) muss angegeben werden. Ein vergrößern der Dateigröße im nach hinein ist unmöglich.

Truncate	
0	Daten überschreiben
1	Daten werden gelöscht

Datei schließen (Schreiboperation)

#FWC	Zeit, Datum

Der Befehl schließt eine geöffnete Datei (Schreiboperation). Der Schreibvorgang wird abgeschlossen und es wird sichergestellt, dass alle Daten geschrieben worden sind. Sind **Zeit** und **Datum** angegeben oder die Uhrzeit vorher gesetzt wird der Zeitstempel eingetragen, ansonsten bleibt 1.1.1980 und kann später eingetragen werden.

Schreibposition von Datei setzen

#FWP	Position [32-Bit]

Der Befehl setzt die Schreib-**Position** (32-Bit Wert) an die bestimmte Stelle in der Datei. Bei einem Wert <0 wird die Postion vom Dateiende aus berechnet

Siehe auch fposW()

Daten in die Datei schreiben

#FWD	Anzahl [32-Bit], Binäre Daten

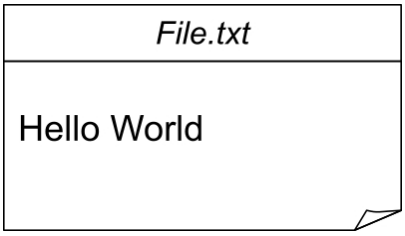
Der Befehl schreibt eine **Anzahl** (32-Bit Wert) an **Binären Daten** in die geöffnete Datei. Für die Makroprogrammierung

ist dieser Befehl ungeeignet. Er dient zur Übertagung von binären Daten über die Schnittstelle.

ASCII-String in die Datei schreiben

#FWA	"String"

Mit dem Befehl wird ein ASCII-String (8 Bit pro Zeichen) in die geöffnete Datei geschrieben.

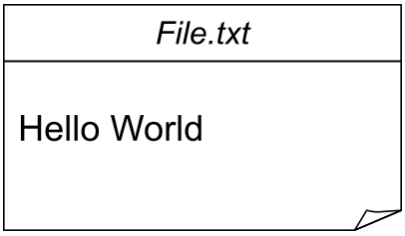



```
...
#FWO <Project/File.txt>
#FWA "Hello World"
#FWC
...
```

Unicode-String in die Datei schreiben

#FWU	"String"

Mit dem Befehl wird ein Unicode-String (16 Bit pro Zeichen) in die geöffnete Datei geschrieben.




```
...
#FWO <Project/File.txt>
#FWU "Hello World"
#FWC
...
```

Registerwerte in die Datei schreiben

#FWR	Register-ID, ...

Mit dem Befehl wird der Registerwert (Register-ID) in die geöffnete Datei geschrieben. Ausgelesen werden kann der Wert mit dem Befehl #FRR. Für jedes Register werden 5 Bytes benötigt.

Stringregister in die Datei schreiben

#FWS	String-ID, ...

Mit dem Befehl wird der Inhalt des Stringregisters (String-ID) in die geöffnete Datei geschrieben. Ausgelesen werden kann der String mit dem Befehl #FRS. Für jedes Register werden 2·(n+1) Bytes (n=Anzahl der Buchstaben) benötigt.

Array in die Datei schreiben (ab V1.4)

#FWY	Array-ID, ...

Mit dem Befehl wird der Inhalt des Arrays (**Array-ID**) in die geöffnete Datei geschrieben. Ausgelesen werden kann der String mit dem Befehl #FRY. Für jedes Register werden 6+4·n Bytes (n=Länge des Arrays) benötigt.

Datei zum lesen öffnen

#FRO	<Dateiname>, Position [32-Bit] (Anfang)

Der Befehl öffnet (nur lesen) eine Datei. Der Parameter **Position** (32-Bit Wert) gibt die Stelle in der Datei an, ab der gelesen werden soll.

Datei schließen (Leseoperation)

#FRC	

Der Befehl schließt eine geöffnete Datei (Leseoperation).

Leseposition von Datei setzen

#FRP	Position [32-Bit]

Der Befehl setzt die Lese-**Position** (32-Bit Wert) an die bestimmte Stelle in der Datei. Bei einem Wert <0 wird die Postion vom Dateiende aus berechnet

Siehe auch fposR()

Daten lesen und senden

#FRD	Anzahl [32-Bit] (ganze Datei)

Der Befehl liest eine **Anzahl** (32-Bit Wert) an Bytes aus der geöffneten Datei aus und stellt die Daten in den Sendepuffer. Die Rückmeldung ist folgendermaßen aufgebaut:

ESC	F	R	D	Anzahl	Daten 1	Daten 2	...	Daten n	...
$1B	$46	$52	$44	32-Bit Wert	8-Bit Wert	8-Bit Wert		8-Bit Wert	

ASCII-String lesen und in ein Stringregister schreiben

#FRA	String-ID, ...

Mit dem Befehl wird ein ASCII-String (8 Bit pro Zeichen) bis zum Zeichen "\n" gelesen und in ein Stringregister (**String-ID**) gespeichert.

Unicode-String lesen und in ein Stringregister schreiben

#FRU	String-ID, ...

Mit dem Befehl wird ein Unicode-String (16 Bit pro Zeichen) bis zum Zeichen "\n" gelesen und in ein Stringregister (**String-ID**) gespeichert.

Daten in ein Register laden

#FRR	Register-ID, ...

Der Befehl liest ein mit #FRW geschriebenes Register zurück und speichert es in das Register (**Register-ID**).

Daten in ein Stringregister laden

#FRS	String-ID, ...

Der Befehl liest ein mit #FRS geschriebenes Stringregister zurück und speichert es in das Stringregister (**String-ID**).

Daten lesen (8-Bit) und in ein Stringregister schreiben

#FRB	String-ID, Anzahl, Anzahl [ID+1],...

Der Befehl liest eine beliebige **Anzahl** (1...250) an Bytes und speichert es in das Stringregister (**String-ID**).

Daten lesen (16-Bit) und in ein Stringregister schreiben

#FRW	String-ID, Anzahl, Anzahl [ID+1],...

Der Befehl liest eine beliebige **Anzahl** (1...250) an Words und speichert es in das Stringregister (**String-ID**).

Daten in ein Array laden

#FRY	Array-ID, ...

Der Befehl liest ein mit #FWY geschriebenes Array zurück und speichert es in das Array (**Array-ID**).

Datei löschen

#FFD	<Dateiname>

Der Befehl löscht die Datei (**<Dateiname>**)

Allgemeine Befehle

Datei Information senden

#FFI	<Pfad> (aktuelles Arbeitsverzeichnis)

Der Befehl stellt alle Informationen über die Datei (wie z.B. Zeitstempel, Größe) in den Sendepuffer. Sollte die Datei nicht vorhanden sein, wird ein Leerstring zurückgegeben, die restlichen Parameter werden nicht mehr übertragen. Die Rückmeldung ist folgendermaßen aufgebaut:

ESC	F	F	I	Dateiname	Größe	Attribut	Zeit	Datum	
$1B	$46	$46	$49	'String' mit $00 abgeschlossen	32-Bit Wert	8-Bit Wert	16-Bit Wert	16-Bit Wert	...

Siehe auch fileS(), fileA(), fileT()

Datei umbenennen

#FFR	<Pfad>, <Neuer Dateiname>, Replace (0)

Der Befehl ändert den angegeben **<Pfad>** in ein neuen Namen (**<Neuer Dateiname>** ist nur der neue Name, ohne Pfad).

Replace	
0	Nicht umbenennen wenn Ordner/Datei bereits vorhanden
1	Vorhandenen Ordner/Datei löschen und dann umbenennen

Datei kopieren

#FFC	<Pfad>, <Neuer Pfad>, Replace (0)

Der Befehl kopiert die angegebene Datei (**<Pfad>**) an einen neuen Ort (**<Neuer Pfad>**).

Replace	
0	Nicht umbenennen wenn Datei bereits vorhanden
1	Vorhandenen Datei löschen und dann umbenennen

Datei verschieben

#FFM	<Pfad>, <Neuer Pfad>, Replace (0)

Der Befehl verschiebt den angegeben Ordner, die angegebene Datei (**<Pfad>**) an einen neuen Ort (**<Neuer Pfad>**).

Replace	
0	Nicht umbenennen wenn Ordner/Datei bereits vorhanden
1	Vorhandenen Ordner/Datei löschen und dann umbenennen

Zeitstempel von Datei ändern

#FFT	<Pfad>, Zeit, Datum

Der Befehl ändert den Zeitstempel der Datei (**<Pfad>**):

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Zeit	Stunde [0...23]					Minute [0...59]						Sekunde/2 [0...29]				
Datum	Jahr (ab 1.1.1980 0:0:0 Uhr) [0...127]							Monat [1...12]				Tag [1...31]				

Umrechnungsbeispiel:
Zeit = (Stunde<<11) + (Minute<<5) + (Sekunde>>1);
Datum = ((Jahr-1980)<<9) + (Monat<<5) + Tag;

Siehe auch fatT(datetime), fatD(datetime), fattime(Fat-Time, Fat-Date)

Attribute von Datei ändern

#FFA	<Pfad>, Attribut

Der Befehl setzt die **Attribute** der Datei. Die Attribute können mit Bitveroderung gleichzeitig gesetzt werden.

Attribut	
$01	Schreibgeschützt
$02	Versteckt
$04	System
$20	Archiv

Siehe auch fileA()

Dateinamen in ein Stringregister laden

#FNF	<Pfad> (aktuelles Arbeitsverzeichnis), ID (1)

Der Befehl speichert alle Dateinamen die im **<Pfad>** vorhanden sind in die Stringregister (String-ID = **ID**...**IDn**). Die Anzahl wird im Register (Register-ID = **ID**) abgelegt. (Es kann mit Widlcards ?/* gesucht werden z.B. *.txt legt alle Textdateien ab)

Unterverzeichnisse in ein Stringregister laden

#FND	<Pfad> (aktuelles Arbeitsverzeichnis), ID (1)

Der Befehl speichert alle Ordnernamen die im **<Pfad>** vorhanden sind in die Stringregister (String-ID = **ID**...**IDn**). Die Anzahl wird im Register (Register-ID = **ID**) abgelegt. (Es kann mit Widlcards ?/* gesucht werden z.B. *Neu* legt alle Verzeichnisse ab in denen das Wort "Neu" vorkommt)

Systembefehle #X

Einstellung des EA uniTFTs-Serie.

Interfaceeinstellung für Kommunikation mit externer Steuerung (Slave Interfaces)

Slave RS232 Parameter einstellen (System Configure Rs232 slave)	**#XCR**	Baudrate [32-Bit], RS485 (keine Änderung), Flash(0)
Slave SPI Parameter einstellen (System Configure Spi slave)	**#XCS**	Modus, Datenreihenfolge (keine Änderung), Flash(0)
Slave I²C Parameter einstellen (System Configure I2c slave)	**#XCI**	Adresse, Flash(0)

Modulbefehle

Projektpfad setzen (System Projectpath Set)	**#XPS**	<Pfad>
Projektpfad senden (System Projectpath Get)	**#XPG**	
Hintergrundbeleuchtung: Helligkeit einstellen (System Configure backlight Brightness)	**#XCB**	Helligkeit, Zeit (keine Änderung), Flash(0)
Hintergrundbeleuchtung: Änderungskurfe und Frequenz einstellen (System Configure backlight Frequency)	**#XCF**	Potenz, Frequenz (no change), Flash(0)
Hintergrundbeleuchtung: Status automatische Dimmung (System led AutoState)	**#XAS**	Status
Hintergrundbeleuchtung: Automatische Dimmung einstellen (System led Autostate mask)	**#XAL**	Maske, Zeit1(60), Helligkeit1(50), Zeit2(60), Helligkeit2(10)
ASCII-String senden (System Send Ascii)	**#XSA**	"String"
Unicode-String senden (System Send Unicode)	**#XSU**	"String"
Hardcopy senden (System Hardcopy Send)	**#XHS**	Format(1), x(0), y(0), Anker(7), Breite(Display Breite), Höhe(Display Höhe)
Hardcopy als Datei speichern (System Hardcopy File)	**#XHF**	<Name>, Format(1), x(0), y(0), Anker(7), Breite(Display Breite), Höhe(Display Höhe)
Hardcopy als Bildobjekt anzeigen (System Hardcopy to Object)	**#XHO**	Obj-ID, x,y,Anker, Breite, Höhe
Objekt als neues Bildobjekt anzeigen (System Hardcopy Id to object)	**#XHI**	Obj-ID, Obj-ID Quelle
Display Ausrichtung einstellen (System Configure Orientation)	**#XCO**	Ausrichtung
Firmwareversion senden (System Info Verion)	**#XIV**	
Modulparameter senden (System Info Display)	**#XID**	

Speicherübersicht senden (System Info Storage)	**#XII**	
RAM Speicherübersicht senden (System Info RAM)	**#XIR**	
Display-Refresh-Rate einstellen (System Configure display Update)	**#XCU**	Option, Flash(0)
Programmlauf pausieren (System Wait hs)	**#XXW**	Zeit
Protokoll aktivieren/deaktivieren (System Configure Protocol)	**#XCP**	Protokoll
Display neu starten (System Firmaware reset)	**#XFB**	Option(0)

Interfaceeinstellung für Kommunikation mit externer Steuerung (Slave Interfaces)

Slave RS232 Parameter einstellen

#XCR	Baudrate [32-Bit], RS485 (keine Änderung), Flash(0)

Mit dem Befehl wird die **Baudrate** (32-Bit Wert) eingestellt:

Baudrate	Fehler
9600	+0.04
19200	-0.08
38400	+0.16
57600	-0.08
115200	+0.64
230400	-0.80
460800	+2.08
921600	-3.68

Der Parameter **Flash** bestimmt ob die Einstellung gespeichert werden soll:

Flash	
0	Einstellung nicht speichern
1	Einstellung dauerhaft speichern

Slave SPI Parameter einstellen

#XCS	Modus, Datenreihenfolge (keine Änderung), Flash(0)

Mit dem Befehl wird der SPI-**Modus** (0..3) und die **Datenreihenfolge** der Slave SPI Schnittstelle eingestellt

Datenreihenfolge	
0	MSB first
1	LSB first

Der Parameter **Flash** bestimmt ob die Einstellung gespeichert werden soll:

Flash	
0	Einstellung nicht speichern
1	Einstellung dauerhaft speichern

Slave I²C Parameter einstellen

#XCI	Adresse, Flash(0)

Mit dem Befehl wird die **Adresse** der Slave I²C Schnittstelle eingestellt. Default ist das Modul mit der Adresse $DE ansprechbar. Der Parameter **Flash** bestimmt ob die Einstellung gespeichert werden soll:

Flash	
0	Einstellung nicht speichern
1	Einstellung dauerhaft speichern

Modulbefehle

Projektpfad setzen

#XPS	<Pfad>

Mit dem Befehl wird der Projektpfad festgelegt. Unter diesem Pfad sucht das Modul automatisch nach Dateinamen, wie z.B. Makros. In Pfadangaben kann dann mit <P:...> gearbeitet werden.

Projektpfad senden

#XPG	

Der Befehl stellt den aktuellen Projektpfad in den Sendepuffer. Die Rückmeldung ist folgendermaßen aufgebaut:

ESC	X	P	G	Pfad	...

$1B	$58	$50	$47	'String' mit $00 abgeschlossen	

Hintergrundbeleuchtung: Helligkeit einstellen

#XCB	Helligkeit, Zeit (keine Änderung), Flash(0)

Mit dem Befehl wird die **Helligkeit** der Hintergrundbeleuchtung [0...150] in % angegeben. Der Parameter **Zeit** (in 1/100 s) gibt an wie schnell diese erreicht wird. Im Auslieferungszustand ist die Helligkeit 100% und ändert sich innerhalb 1 Sekunde (**time** = 100). Bei Helligkeiten über 100% muss mit einem Derating der Lebensdauer gerechnet werden - wir empfehlen diese Einstellung nur kurzzeitig zu verwenden, z.B. bei direkter Sonneneinstrahlung. Der Parameter **Flash** bestimmt ob die Einstellung gespeichert werden soll:

Flash	
0	Einstellung nicht speichern
1	Einstellung dauerhaft speichern

Hintergrundbeleuchtung: Abstufung und Frequenz einstellen

#XCF	Potenz, Frequenz (no change), Flash(0)

Die Helligkeitsstufen des Backlights werden mit einer Potenzfunktion festgelegt. Je nach Einsatzgebiet ist es sinnvoll mehr Stufen im niedrigen Bereich (Nightvision) zu haben. Hierfür muss der Parameter **Potenz** vergrößert werden. Default ist der Wert 10. Auch die PWM-**Frequenz** [5000...65535] des Backlights kann verändert werden, wenn es zu Indifferenzen mit Umgebungslicht kommt. Default: **Frequenz=5000.** Der Parameter **Flash** bestimmt ob die Einstellung gespeichert werden soll:

Flash	
0	Einstellung nicht speichern
1	Einstellung dauerhaft speichern

Hintergrundbeleuchtung: Status automatische Dimmung

#XAS	Status

Mit dem Befehl wird der **Status** der automatischen Dimmung der Hintergrundbeleuchtung eingestellt, default ist die Dimmung deaktiviert:

Status

0	Aus: keine Dimmung
1	Ein: Autodimmung aktiv
2	Manueller Retrigger

Hintergrundbeleuchtung: Automatische Dimmung einstellen

#XAL	Maske, Zeit1(60), Helligkeit1(50), Zeit2(120), Helligkeit2(10)

Bei aktivierter automatischer Dimmung (siehe #XAS) stellt der Befehl mit dem Parameter **Maske** ein, welche Events einen Retrigger der Zeit auslösen:

Maske	
$01	Touch
$02	USB
$04	RS232
$08	SPI
$10	I²C
$20	Master RS232

Die Maskenbits können mit Bitveroderung gleichzeitig gesetzt werden. **ZeitX** gibt in Sekunden an, wann die neue Helligkeit (**HelligkeitX**) eingestellt werden soll. Der neue Helligkeitswert wird relativ zur aktuellen Helligkeit (0..100) angegeben.

ASCII-String senden

#XSA	"String"

Der Befehl stellt einen String oder einzelne Codes als ASCII-Werte (8-Bit pro Zeichen) in den Sendepuffer.

Unicode-String senden

#XSU	"String"

Der Befehl stellt einen String oder einzelne Codes als Unicode-Werte (16-Bit pro Zeichen) in den Sendepuffer.

Hardcopy senden

#XHS	Format(1), x(0), y(0), Anker(7), Breite(Display Breite), Höhe(Display Höhe)

Der Befehl erstellt ein Bildschirmfoto von der Position (**x,y,Anker**) und stellt es in den in den Sendepuffer. Je nachdem in welchen **Format** das Bild angefordert wurde wird ein Header und die Daten zurückgegeben.

Format	
1	BMP 24-Bit
2	BMP 16-Bit
3	BMP 8-Bit Graustufen
11	epg 32-Bit
12	epg 16-Bit
13	epg 8-Bit Graustufen
21	epg 32-Bit compressed
22	epg 16-Bit compressed
23	epg 8-Bit Graustufen compressed

Die Rückmeldung ist folgendermaßen aufgebaut:

ESC	X	H	S	Header	Daten	...
$1B	$58	$48	$53	...	...	

Hardcopy als Datei speichern

#XHF	<Name>, Format(1), x(0), y(0), Anker(7), Breite(Display Breite), Höhe(Display Höhe)

Der Befehl erstellt ein Bildschirmfoto von der Position (**x,y,Anker**) und schreibt es in eine Datei (**<Name>**).

Format	
1	BMP 24-Bit
2	BMP 16-Bit
3	BMP 8-Bit Graustufen
11	epg 32-Bit
12	epg 16-Bit
13	epg 8-Bit Graustufen
21	epg 32-Bit compressed
22	epg 16-Bit

	compressed
23	epg 8-Bit Graustufen compressed

Hardcopy als Bildobjekt anzeigen (ab V1.4)

#XHO	Obj-ID, x,y,Anker, Breite, Höhe

Der Befehl erstellt ein Bildschirmfoto von der Position (**x,y,Anker**) mit der Größe (**Breite, Höhe**)und zeigt es als neues Bildobjekt mit der **Obj-ID** an.

Objekt als neues Bildobjekt anzeigen (ab V1.4)

#XHI	Obj-ID, Obj-ID Quelle

Der Befehl erstellt aus dem Quellobjekt (**Obj-ID Quelle**) ein neues Bildobjekt mit der **Obj-ID** und zeigt es an.

Display Ausrichtung einstellen

#XCO	Ausrichtung

Der Befehl definiert die **Ausrichtung** (0, 90, 180, 270) des Displays. Default ist 0° Landscape.

Firmwareversion senden

#XIV	

Der Befehl stellt die Firmwareversion und den erkannten Touch in den Sendepuffer. Die Rückmeldung ist folgendermaßen aufgebaut:

ESC	X	I	V	Versionsstring	
$1B	$58	$49	$56	'String' mit $00 abgeschlossen	...

Siehe auch version()

Modulparameter senden

#XID	

Der Befehl stellt Modulparameter (u.a. Auflösung und Interfaceeinstellungen) in den Sendepuffer. Die Rückmeldung ist folgendermaßen aufgebaut:

ESC	X	I	D	Breite	Höhe	Farbtiefe	Touch	VideoBreite	VideoHöhe	...

$1B	$58	$49	$44	16-Bit Wert	16-Bit Wert	8-Bit Wert	8-Bit Wert	16-Bit Wert	16-Bit Wert	

Touch	
$00	Kein Touch
$03	Resistiver Touch
$07	Kapazitiver Touch
$0F	Simulator

Siehe auch scrW(), scrH(), touchT(), vidW(), vidH()

Speicherübersicht senden

#XII	

Der Befehl stellt die Größe und den freier Speicherplatz des speichers in den Sendepuffer. Die Rückmeldung ist folgendermaßen aufgebaut:

ESC	X	I	I	Gesamt	Frei	...
$1B	$58	$49	$53	32-Bit Wert	32-Bit Wert	

Siehe auch memST(), memSF()

RAM Speicherübersicht senden

#XIR	

Der Befehl stellt die Größe und den freier Speicherplatz des Objekt-RAMs in den Sendepuffer. Die Rückmeldung ist folgendermaßen aufgebaut:

ESC	X	I	R	Gesamt	Frei	...
$1B	$58	$49	$52	32-Bit Wert	32-Bit Wert	

Siehe auch memRT(), memRF()

Display-Refresh-Rate einstellen

#XCU	Option, Flash(0)

Der Befehl stellt die Display-Refresh-Rate ein. Default steht der Parameter **Option** auf 3.

Option	
0	Kein Displayupdate

1	Einmaliger Displayupdate
2..1000	Zyklischer Displayupdate (Zeit in 1/100s)

Der Parameter **Flash** bestimmt ob die Einstellung gespeichert werden soll:

Flash	
0	Einstellung nicht speichern
1	Einstellung dauerhaft speichern

Programmlauf pausieren

#XXW	Zeit

Mit dem Befehl wird die Ausführung von Befehlen für die eingestellte **Zeit** (in 1/100s) unterbrochen. Wir empfehlen diesen Befehl nur für Debugzwecke während der Entwicklungszeit.

Protokoll aktivieren/deaktivieren

#XCP	Protokoll

Der Befehl aktiviert bzw. deaktiviert das Small-/Short-**Protokoll**.

Protokoll	
0	deaktivieren
1	aktivieren

Display neu starten

#XFB	Option(0)

Mit dem Befehl kann das Modul neu gestartet werden:

Option	
0	Normaler Reset
1	Testmodus
2	Disable PowerOnMakro
3	Disable Default
4	Bootmenü

5	Reserviert

Antworten / Rückmeldungen

Das Modul stellt nach Anfragen oder Touch-Ereignissen Informationen in seinen Sendepuffer.
Die Antworten sind, falls nicht anders angegeben binär codiert:
<ESC> = 0x1B, die Größe (Bitanzahl) der einzelnen Parameter sind in der Erklärung zur jeweiligen Rückmeldung angegeben.
Das Modul arbeitet mit little-endian (Intel-Format), das niederwertige Byte wird zuerst übertragen.

EditBox

Inhalt von EditBox	**<ESC> SEU**	Obj-ID, Inhalt

Touch

Zustand von Taster/Schalter	**<ESC> TQS**	Obj-ID, Zustand
Aktiver Schalter der Radiogroup	**<ESC> TQR**	Obj-ID, Group-ID
Taste von Keyboard	**<ESC> TQK**	Obj-ID, Code
Bargraph- /Instrument-Wert	**<ESC> TQI**	Obj-ID, Wert
Menü-Eintrag	**<ESC> TQM**	Obj-ID, ItemNummer
ComboBox-Eintrag	**<ESC> TQC**	Obj-ID, ItemNummer
SpinBox-Eintrag	**<ESC> TQB**	Obj-ID, ItemNummer

Variablen/Register

Anzahl geladener Stringfiles	**<ESC> VFC**	Anzahl
Inhalt aus Stringregister (ASCII)	**<ESC> VSA**	String-ID, Länge, Char1, ..., Char n
Inhalt aus Stringregister (Unicode)	**<ESC> VSU**	String-ID, Länge, Char1, ..., Char n
Registerwert ausgeben	**<ESC> VRG**	Register-ID, Typ, Wert

I/O Port

Anzahl der Portbausteine	**<ESC> HPI**	Available
Port lesen	**<ESC> HPR**	Port, Anzahl, Zustand 1, Zustand 2, ...
Portpin lesen	**<ESC> HBR**	Portpin, Anzahl, Zustand 1, Zustand 2, ...

Analog Input

Analogeingang Wert	**<ESC>**	Kanal, Anzahl, Wert 1, Wert 2, ...

	HAR	

Master Schnittstellen

RS232 Daten	**<ESC> HRR**	Länge, Daten 1, Daten 2, ..., Daten n
SPI Daten	**<ESC> HSR**	Länge, Daten 1, Daten 2, ..., Daten n
I²C Daten	**<ESC> HIR**	Länge, Daten 1, Daten 2, ..., Daten n

Uhrzeit

Uhrzeit ASCII Ausgabe	**<ESC> WSA**	ASCII-String
Uhrzeit Unicode Ausgabe	**<ESC> WSU**	Unicode-String
Uhrzeit binäre Ausgabe	**<ESC> WSB**	Stunde, Minute, Sekunde, Tag, Monat, Jahr, Wochentag

Dateizugriffe

Aktuelles Arbeitsverzeichnis	**<ESC> FDG**	Pfad
Alle Ordner und Dateien aus Verzeichnis (binäre Ausgabe)	**<ESC> FDR**	Verzeichnis-/Dateiname, Größe, Attribut, Zeit, Datum, ...
Alle Ordner und Dateien aus Verzeichnis (ASCII Ausgabe)	**<ESC> FDA**	String
Alle Ordner aus Verzeichnis (ASCII Ausgabe)	**<ESC> FDL**	Name 1, Name 2, ..., Name n
Ordner-/ Datei-Information	**<ESC> FFI**	Verzeichnis-/Dateiname, Größe, Attribut, Zeit, Datum
Daten aus Datei	**<ESC> FRD**	Anzahl, Daten1, Daten 2, ..., Daten n

Systembefehle

Aktueller Projektpfad	**<ESC> XPG**	Pfad
Versionsinformationen	**<ESC> XIV**	Versionsstring
Displayinformationen	**<ESC> XID**	Breite, Höhe, Farbtiefe, Touch, VideoBreite, VideoHöhe
RAM-Speicherinformationen	**<ESC> XIR**	Gesamt, Frei
Speicherinformationen	**<ESC> XII**	Gesamt, Frei
Hardcopy	**<ESC> XHS**	Header, Daten

Inhalt von EditBox

ESC	S	E	U	Obj-ID	Inhalt	
$1B	$53	$45	$55	16-Bit Wert	'String' mit $00 abgeschlossen	...

Der Inhalt der EditBox (16 Bit pro Zeichen) wird übertragen. Der String ist mit einer $00 abgeschlossen. Ausgelöst wird die Rückmeldung durch den Keyboard Code 13 ($0D).

Zustand von Taster/Schalter

ESC	T	Q	S	Obj-ID	Zustand	
$1B	$54	$51	$53	16-Bit Wert	16-Bit Wert	...

Der **Zustand** eines Tasters / Schalters (**Obj-ID**) wird übertragen. Welche Ereignisse (Down, Up, Drag) zum Senden der Rückmeldung führen, wird mit dem Befehl #TCR eingestellt. Sollen keine Antworten über die serielle Schnittstelle übertragen werden, erfolgt dies mit `#TCR 0,0,Obj-ID`.

Zustand	
1	Up (nicht gedrückt)
2	Down (gedrückt)

Aktiver Schalter der Radiogroup

ESC	T	Q	R	Obj-ID	Group-ID	
$1B	$54	$51	$52	16-Bit Wert	16-Bit Wert	...

Der aktive Schalter (**Obj-ID**) einer Radiogroup (**Group-ID**) wird übertragen bei jeder Änderung. Sollen keine Antworten über die serielle Schnittstelle übertragen werden, erfolgt dies mit `#TCR 0,0,Obj-ID1,...,Obj-IDn` (alle Objekt IDs der Radiogroup Elemente).

Taste von Keyboard

ESC	T	Q	K	Obj-ID	Code	
$1B	$54	$51	$4B	16-Bit Wert	16-Bit Wert	...

Die letzte gedrückte Taste (**Code**) des Keyboards (**Obj-ID**) wird ausgegeben. Voraussetzung ist, dass die Tastatur nicht mit einer EditBox verbunden ist. Sollen keine Antworten über die serielle Schnittstelle übertragen werden, erfolgt dies mit `#TCR 0,0,Obj-ID`.

Bargraph- /Instrument-Wert

ESC	T	Q	I	Obj-ID	Wert	
$1B	$54	$51	$49	16-Bit Wert	16-Bit Wert	...

Der neue **Wert** des Bargraphs/ Instrument (**Obj-ID**) wird ausgegeben. Welche Ereignisse (Down, Up, Drag) zum

Senden der Rückmeldung führen, wird mit dem Befehl #TCR eingestellt. Sollen keine Antworten über die serielle Schnittstelle übertragen werden, erfolgt dies mit `#TCR 0,0,Obj-ID`.

Menü-Eintrag

ESC	T	Q	M	Obj-ID	ItemNummer	...
$1B	$54	$51	$4D	16-Bit Wert	16-Bit Wert	

Der ausgewählte Menüeintrag (ItemNummer) wird ausgegeben. Welche Ereignisse (Down, Up, Drag) zum Senden der Rückmeldung führen, wird mit dem Befehl #TCR eingestellt. Sollen keine Antworten über die serielle Schnittstelle übertragen werden, erfolgt dies mit `#TCR 0,0,Obj-ID`.

ComboBox-Eintrag

ESC	T	Q	C	Obj-ID	ItemNummer	...
$1B	$54	$51	$43	16-Bit Wert	16-Bit Wert	

Der ausgewählte SpinBox-Eintrag (**ItemNummer**) wird ausgegeben. Welche Ereignisse (Down, Up, Drag) zum Senden der Rückmeldung führen, wird mit dem Befehl #TCR eingestellt. Sollen keine Antworten über die serielle Schnittstelle übertragen werden, erfolgt dies mit `#TCR 0,0,Obj-ID`.

SpinBox-Eintrag

ESC	T	Q	B	Obj-ID	ItemNummer	...
$1B	$54	$51	$42	16-Bit Wert	16-Bit Wert	

Der ausgewählte SpinBox-Eintrag (ItemNummer) wird ausgegeben. Welche Ereignisse (Down, Up, Drag) zum Senden der Rückmeldung führen, wird mit dem Befehl #TCR eingestellt. Sollen keine Antworten über die serielle Schnittstelle übertragen werden, erfolgt dies mit `#TCR 0,0,Obj-ID`.

Anzahl geladener Stringfiles

ESC	V	F	C	Anzahl	...
$1B	$56	$53	$43	16-Bit Wert	

Die Anzahl an verwendeten Strings aus den Stringfiles wird ausgegeben.

Inhalt aus Stringregister (ASCII)

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

Der Inhalt (8 Bit pro Zeichen) des Stringregisters **(String-ID)** und die **Länge** wird ausgegeben. Der String wird nicht mit $00 abgeschlossen.

Inhalt aus Stringregister (Unicode)

ESC	V	S	U	String-ID	Länge	Char 1	Char 2	...	Char n	...
$1B	$56	$53	$55	16-Bit Wert	16-Bit Wert	16-Bit Wert	16-Bit Wert	16-Bit Wert	16-Bit Wert	

Der Inhalt (16 Bit pro Zeichen) des Stringregisters (**String-ID**) und die Länge wird ausgegeben. Der String wird nicht mit $00 abgeschlossen.

Registerwert ausgeben

ESC	V	R	G	Register-ID	Typ	Wert	...
$1B	$56	$52	$47	16-Bit Wert	16-Bit Wert	32-Bit Wert	

Der Inhalt des Registers (**Register-ID**) und der **Typ** wird ausgegeben:

Typ	
'I'	Integer
'F'	Float

Anzahl der Portbausteine

ESC	H	P	I	Available
$1B	$48	$50	$49	16-Bit Wert

Alle verfügbaren **Adressen** der angeschlossenen Portbausteine werden ausgegeben. Intern ist ein Baustein mit der Adresse 0 vorhanden, sodass ohne externe Hardware $01 zurückgegeben wird.

Port lesen

ESC	H	P	R	Port	Anzahl	Zustand 1	Zustand 2	...
$1B	$48	$50	$52	8-Bit Wert	8-Bit Wert	8-Bit Wert	8-Bit Wert	

Der Zustand (**Zustand 1**) des **Ports** wird ausgegeben. Ist die **Anzahl** >1, so werden die auf den Portbaustein folgenden Zustände gesendet **(Zustand 2, Zustand n**).

Portpin lesen

ESC	H	B	R	Portpin	Anzahl	Zustand 1	Zustand 2	...
$1B	$48	$42	$52	8-Bit Wert	8-Bit Wert	8-Bit Wert	8-Bit Wert	

Der Zustand (**Zustand 1**) des **Portpins** wird ausgegeben. Ist die **Anzahl** >1, so werden die auf den Portbaustein folgenden Zustände gesendet **(Zustand 2, Zustand n**).

Analogeingang Wert

ESC	H	A	R	Kanal	Anzah	Wert 1	Wert 2	...

					I			
$1B	$48	$41	$52	8-Bit Wert	8-Bit Wert	16-Bit Wert	16-Bit Wert	

Der Wert (**Wert 1**) des Analog**Kanals** wird ausgegeben. Ist die **Anzahl** >1, so werden die auf den Kanal folgenden Messwerte gesendet **(Wert 2**).

RS232 Daten

ESC	H	R	R	Länge	Daten 1	Daten 2	...	Daten n	...
$1B	$48	$52	$52	32-Bit	8-Bit Wert	8-Bit Wert	8-Bit Wert	8-Bit Wert	

Die Daten (**Daten 1, Daten2, ..., Daten n**) die über die Master RS232 Schnittstelle empfangen worden sind werden ausgegeben. **Länge** gibt an wie viele Daten gesendet werden.

SPI Daten

ESC	H	S	R	Länge	Daten 1	Daten 2	...	Daten n	...
$1B	$48	$53	$52	32-Bit	8-Bit Wert	8-Bit Wert	8-Bit Wert	8-Bit Wert	

Die Daten (**Daten 1, Daten2, ..., Daten n**) die über die Master SPI Schnittstelle empfangen worden sind werden ausgegeben. **Länge** gibt an wie viele Daten gesendet werden.

I²C Daten

ESC	H	I	R	Länge	Daten 1	Daten 2	...	Daten n	...
$1B	$48	$49	$52	32-Bit	8-Bit Wert	8-Bit Wert	8-Bit Wert	8-Bit Wert	

Die Daten (**Daten 1, Daten2, ..., Daten n**) die über die Master I²C Schnittstelle empfangen worden sind werden ausgegeben. **Länge** gibt an wie viele Daten gesendet werden.

Uhrzeit ASCII Ausgabe

ESC	W	S	A	ASCII-String	Abschluss	...
$1B	$57	$53	$41		$00	

Die angeforderte Uhrzeit wird im eingestellten Format als ASCII übertragen. Der String ist mit einer $00 abgeschlossen.

Uhrzeit Unicode Ausgabe

ESC	W	S	U	Unicode-String	Abschluss	...

$1B	$57	$53	$55		$00	

Die angeforderte Uhrzeit wird im eingestellten Format als Unicode übertragen. Der String ist mit einer $00 abgeschlossen.

Uhrzeit binäre Ausgabe

ESC	W	S	B	Stunde	Minute	Sekunde	Tag	Monat	Jahr	Wochentag	...
$1B	$57	$53	$42	16-Bit Wert	16-Bit Wert	16-Bit Wert	16-Bit Wert	16-Bit Wert	16-Bit Wert	16-Bit Wert	

Die angeforderte Uhrzeit wird im eingestellten Format binär übertragen. Wochentag =0 bedeutet Sonntag

Aktuelles Arbeitsverzeichnis

ESC	F	D	G	Pfad	...
$1B	$46	$44	$47	'String' mit $00 abgeschlossen	

Das aktuelle Arbeitsverzeichnis wird ausgegeben.

Alle Ordner und Dateien aus Verzeichnis (binäre Ausgabe)

ESC	F	D	R	Verzeichnis-/ Dateiname	Größe	Attribut	Zeit	Datum	...	Abschluss
$1B	$46	$44	$52	'String' mit $00 abgeschlossen	32-Bit Wert	8-Bit Wert	16-Bit Wert	16-Bit Wert		$00

Alle Ordner und Dateien im aktuellen Arbeitsverzeichnis werden ausgegeben.

Attribut	
$01	Schreibgeschützt
$02	Versteckt
$04	System
$20	Archiv

Die Zeit und das Datum ergeben den Zeitstempel der letzten Änderung der Datei an.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Zeit	Stunde [0...23]					Minute [0...59]						Sekunde/2 [0...29]				
Datum	Jahr (ab 1.1.1980 0:0:0 Uhr) [0...127]							Monat [1...12]				Tag [1...31]				

Alle Ordner und Dateien aus Verzeichnis (ASCII Ausgabe)

ESC	F	D	A	String	Abschluss	...
$1B	$46	$44	$41	Größe, Attribut, Zeit, Datum, Name	CRLF	

Alle Ordner und Dateien im aktuellen Arbeitsverzeichnisses werden als ASCII Strings ausgegeben.

Alle Ordner aus Verzeichnis (ASCII Ausgabe)

ESC	F	D	L	Verzeichnisname	...	Abschluss
$1B	$46	$44	$4C	'String' mit $00 abgeschlossen		$00

Alle Ordnernamen im aktuellen Arbeitsverzeichnisses werden als ASCII Strings ausgegeben.

Ordner-/ Datei-Information

ESC	F	F	I	Verzeichnis-/ Dateiname	Größe	Attribut	Zeit	Datum	...
$1B	$46	$46	$49	'String' mit $00 abgeschlossen	32-Bit Wert	8-Bit Wert	16-Bit Wert	16-Bit Wert	

Ordner-/Datei-Informationen werden ausgegeben.

Attribut	
$01	Schreibgeschützt
$02	Versteckt
$04	System
$20	Archiv

Die Zeit und das Datum ergeben den Zeitstempel der letzten Änderung der Datei an.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Zeit	Stunde [0...23]					Minute [0...59]						Sekunde/2 [0...29]			
Datum	Jahr (ab 1.1.1980 0:0:0 Uhr) [0...127]							Monat [1...12]				Tag [1...31]			

Daten aus Datei

ESC	F	R	D	Anzahl	Daten 1	Daten 2	...	Daten n	...
$1B	$46	$52	$44	32-Bit Wert	8-Bit Wert	8-Bit Wert	...	8-Bit Wert	

Die Daten aus der Datei werden ausgegeben. **Anzahl** gibt die Länge der Datei an.

Aktueller Projektpfad

ESC	X	P	G	Pfad	
$1B	$58	$50	$47	'String' mit $00 abgeschlossen	...

Der aktuelle Projektpfad wird ausgegeben.

Versionsinformationen

ESC	X	I	V	Versionsstring	
$1B	$58	$49	$56	'String' mit $00 abgeschlossen	...

Die Versionsinformationen des Displays werden ausgegeben (z.B. "EA uniTFT V1.4 with capacitive touch")

Displayinformationen

ESC	X	I	D	Breite	Höhe	Farbtiefe	Touch	VideoBreite	VideoHöhe	
$1B	$58	$49	$44	16-Bit Wert	16-Bit Wert	8-Bit Wert	8-Bit Wert	16-Bit Wert	16-Bit Wert	...

Displayinformationen werden ausgegeben.

Touch	
$00	Kein Touch
$03	Resistiver Touch
$07	Kapazitiver Touch
$0F	Simulator

RAM-Speicherinformationen

ESC	X	I	R	Gesamt	Frei	
$1B	$58	$49	$52	32-Bit Wert	32-Bit Wert	...

RAM-Speicherinformationen werden ausgegeben.

Speicherinformationen

ESC	X	I	I	Gesamt	Frei	...

$1B	$58	$49	$53	32-Bit Wert	32-Bit Wert	

Speicherinformationen werden ausgegeben.

Hardcopy

ESC	X	H	S	Header	Daten	...
$1B	$58	$48	$53	...	...	

Eine Hardcopy des Displayinhalts wird ausgegeben. **Header** und **Daten** richten sich nach dem ausgewählten Format.

Funktionen und Kalkulationen

Die EA uniTFTs-Serie kann zur Laufzeit kleine mathematische Aufgaben lösen. Zusätzlich bieten logische Operatoren die Möglichkeit, Entscheidungen zu treffen (ähnlich einer if-Anweisung). Um Benutzereingaben auswerten oder das Layout optimieren zu können, sind ebenso Kalkulationsbefehle vorhanden, die Objekteigenschaften auslesen können (wie zum Beispiel Bargraphwert, letzte Touchposition oder aber Objektbreite und -position). Die meisten Funktionen sind sowohl als Integer- als auch als Fließkommaberechnung vorhanden. Es muss darauf geachtet werden im jeweiligen
Zahlenbereich zu bleiben, bzw. mit dem cast-Operator (float bzw. int) zu wandeln.
Das Modul arbeitet mit little-endian (Intel-Format), das kleinstwertige Byte wird also zuerst übertragen.
Hinweis:
Kalkulationen müssen immer innerhalb von Klammern (...) ausgeführt werden. Siehe Parameterübergaben für weitere Informationen.

Mathematische Funktionen

		signed Integer (32 Bit)	Float IEEE 754 (32-Bit)
Arithmetische Funktionen	+, -, * , /, ()	•	•
Betrag |x|	abs(x)	•	•
Vorzeichen von x (-1, 0, 1): x<0, x==0, x>0	sign(x)	•	•
Modulo x%y	mod(x,y)	•	•
Potenz x^y	pow(x,y)	•	•
Wurzel	sqrt(var)		•
Kommastellen abschneiden	trunc(var)		•
Kommastellen runden	round(var)		•
Logarithmus (log)	log(var)		•
Natürlicher Logarithmus (ln, Basis e)	ln(var)		•
Exponentialfunktion mit Basis e	exp(var)		•
Grad in Rad	rad(deg)		•
Rad in Grad	deg(rad)		•
Sinus	sin(deg)		•
Kosinus	cos(deg)		•
Tangens	tan(deg)		•
Arkussinus	asin(var)		•
Arkuskosinus	acos(var)		•
Arkustangens	atan(var)		•
Arkustangens, Quadranten-richtig	atan(y,x)		•

Minimum	min(a,b,c…)	•	•
Maximum	max(a,b,c…)	•	•
Durchschnitt	avg(a,b,c…)	•	•
Zufallswert sv <= x <= ev (max 65535)	rand(sv,ev)	•	•
Zufallswert 0 <=x <= ev (max 65535)	rand(ev)	•	•
Zufallswert 0<= x<=1000	rand()	•	•

Register increment / decrement

pre-/post- increment	++Rx, Rx++	•	•
pre-/post- decrement	--Rx, Rx--	•	•

Cast Integer ↔ Float

Integer Kalkulation ausführen, Float zurückgeben	int(Kalkulation/Funktion)		•
Float Kalkulation ausführen, Integer zurückgeben	float(Kalkulation/Funktion)	•	

Bit Operatoren

Low Byte	loB(x)	•	
High Byte	hiB(x)	•	
Low Word	loW(x)	•	
High Word	hiW(x)	•	
Low Byte von High Word	hwloB(x)	•	
High Byte von High Word	hwhiB(x)	•	
AND	&	•	
OR	|	•	
NOT	~	•	
XOR	^	•	
Shift links / rechts	<<, >>	•	
Bit setzen (Bit-Nr. 0..31)	bitS(Wert, Bit-Nr.)	•	
Bit löschen (Bit-Nr. 0..31)	bitC(Wert, Bit-Nr.)	•	
Exor Bit (Bit-Nr. 0..31)	bitX(Wert, Bit-Nr.)	•	
Test Bit (Bit-Nr. 0..31) returns true or false	bitT(Wert, Bit-Nr.)	•	

Logische Operatoren

UND	&&	•	•
ODER	||	•	•
NICHT	!	•	•
Gleich, Ungleich	==, !=	•	•
Kleiner, Kleiner gleich	<, <=	•	•

Größer, Größer gleich	>, >=	•	•

Entscheidung

If-Then-Else-Funktion	ifte(Bedingung, Wert true, Wert false)	•	•

Objektbefehle Allgemein

Breite (ohne Transformationen)	objW(id)	•	•
Höhe (ohne Transformationen)	objH(id)	•	•
Position X (Aktueller Anker, bei Gruppen: relativ zum Elternobjekt)	objX(id)	•	•
Position Y (Aktueller Anker, bei Gruppen: relativ zum Elternobjekt)	objY(id)	•	•
Screen-Position X (angegebener Anker, auch bei Gruppen: Screenkoordinaten)	objX(id, Anker)	•	•
Screen-Position Y (angegebener Anker, auch bei Gruppen: Screenkoordinaten)	objY(id, Anker)	•	•
Skalierung Breite	objSW(id)	•	•
Skalierung Höhe	objSH(id)	•	•
Scherung X	objSX(id)	•	•
Scherung Y	objSY(id)	•	•
Rotation	objR(id)	•	•
Transparenz	objO(id)	•	•
Layer	objL(id)	•	•
Aktuellen Style auslesen	objC(id)	•	•
Aktuellen Anker auslesen	objA(id)	•	•
Testen ob Objekt existiert	objE(id)	•	•
Testen ob Objekt sichtbar	objV(id)	•	•
Erhalten der Obj-ID an Screenkoordinate	objXY(x,y)	•	•
Erhalten der Obj-ID an Screenkoordinate. Nur das Objektrechteck wird beachtet (schneller, Transparenzen werden ignoriert)	objXY(x,y, onlyrect)	•	•

Objektbefehle Menü

Letztes gültiges Menültem auslesen	objML(id)	•	•
Aktuell angezeigtes Menültem (0=geschlossen)	objMV(id)	•	•
Check state von Menültem (1=checked, 0=unchecked)	objMC(id, item)	•	•
Enable state von Menültem (1=enabled, 0=disabled)	objME(id, item)	•	•

Objektbefehle ComboBox

Letztes gültiges Item auslesen	objCL(id)	•	•
Aktuell angezeigtes Item (0=geschlossen, -1=kein item sichtbar)	objCV(id)	•	•
Enable state von Item (1=enabled, 0=disabled)	objCE(id, item)	•	•

Objektbefehle SpinBox

Letzte gültige Items auslesen	objBL(id)	•	•
Letzte gültiges Item aus Box der Spin-Gruppe auslesen	objBL(id, boxnr)	•	•
Aktuell angezeigte Items auslesen	objBV(id)	•	•
Aktuell angezeigtes Item aus Box der Spin-Gruppe auslesen	objBV(id, boxnr)	•	•
Enable state von Item (1=enabled, 0=disabled)	objBE(id, item)	•	•

Objektbefehle StringBox

Sichtbare Zeilen	objTV(id)	•	•
Absatzanzahl (=Zeilenanzahl ohne AutoWrap)	objTA(id)	•	•
Zeilenanzahl (nach AutoWrap)	objTN(id)	•	•
Stringzeilenanzahl (nach AutoWrap)	objTN(id, Absatz nr)	•	•
Sichtbare oberste Zeile	objTL(id)	•	•
Stringzeilenstart (nach AutoWrap)	objTL(id, Absatz nr)	•	•
Absatz aus Zeilennummer	objTS(id, line nr)	•	•

Objekteigenschaften

Userwert Integer (#OUI) auslesen	objUI(id)	•	•
Userwert Float (#OUI) auslesen	objUF(id)	•	•

Objekteigenschaften Bargraph/Instrument

Eingestellter Wert	objIV(id)	•	•
Auf dem Screen gezeichneter Wert (bei Animationen nicht zwingend gleich mit objIV(id))	objID(id)	•	•
EndWert	objIE(id)	•	•
StartWert	objIS(id)	•	•

Objekteigenschaften Pfade

Länge	pathL(id)	•	•
X-Koordinate anhand der Länge	pathX(id,distance)	•	•
Y-Koordinate anhand der Länge	pathY(id,distance)	•	•
Tangentenwinkel eines Punktes	pathR(id,distance)	•	•

Touchfunktionen

Touchbutton / Touchtaste Zustand =1 ungedrückt	butS(id)	•	•

=2 gedrückt			
Radiogroup: Aktiver Touchswitch (id)	butR(id)	•	•
Letzte Touchtaste	butI()	•	•
Letzter Keyboard code	butC()	•	•
Anzahl der Touchpunkte (Down-Event)	touchA()	•	•
Letzte Touchposition X (Down- oder Drag-Event)	touchX()	•	•
Letzte Touchposition Y (Down- oder Drag-Event)	touchY()	•	•
Touchposition X von Punktnummer nr (Down- oder Drag-Event)	touchX(nr)	•	•
Touchposition Y von Punktnummer nr (Down- oder Drag-Event)	touchY(nr)	•	•

Zerlegungsfunktionen Menü

RootItem von Item	menR(item)	•	•
Menü von Item	menM(item)	•	•
Submenü von Item	menS(item)	•	•
Item von Root, Menü, Submenü	menRMS(r,m,s)	•	•

Zerlegungsfunktionen SpinBox

Eintrag Box 1 (8 Bit value)	spin1(item)	•	•
Eintrag Box 2 (8 Bit value)	spin2(item)	•	•
Eintrag Box 3 (8 Bit value)	spin3(item)	•	•
Eintrag Box 4 (8 Bit value)	spin4(item)	•	•
Item aus individuellen SpinBox Werten ((e4<<24) | (e3<<16) | (e2<<8) | e1)	spinE(e1,e2,e3,e4)	•	•

I/O Ports

Portzustand (a = Baustein 0..15)	port(a)	•	•
Portpinzustand (a = Pinnummer 0..127)	bit(a)	•	•

Analogeingang

Analogwert (a=0..3)	analog(a)	•	•

RS232 Masterschnittstelle

Anzahl empfangener Bytes	mstRA()	•	

Timer

Startwert setzen (10 ms Timer) (ab Firmware V1.1)	timer(startvalue)	•	
Auslesen (10 ms Timer)	timer()	•	

Datum/ Uhrzeit

Datum und Uhrzeit wird ab dem 1.1.2000 um 00:00:00 Uhr in Sekunden mit SINT32 berechnet (=Datumszeitwert). So ist der maximale vom Modul verwendbare Zeitraum von 1932 - 2067. Das Basisdatum kann mit dem Befehl #WDY geändert werden. Um Zeiträume zu berechnen

muss das Datum oder die Uhrzeit immer erst in Sekunden bzw. dem Datumszeitwert gewandelt werden, danach wird die Berechnung ausgeführt. Am Ende kann wieder in Minuten, Stunden, Tag, Monat und Jahr zurückgewandelt werden.

Aktuelles Datum in Datumszeitwert wandeln	date()	•	
Anzahl Tage in Sekunden wandeln	date(D)	•	
Tag + Monat + Jahr (1932 - 2067) in Datumszeitwert wandeln	date(D,M,Y)	•	
aktuelle Uhrzeit in Sekunden wandeln (berechnet ab 0:00:00 Uhr)	time()	•	
Anzahl Stunden in Sekunden wandeln (≙ h·3600)	time(h)	•	
Stunden und Minuten in Sekunden wandeln (≙ h·3600 + m·60)	time(h, m)	•	
Stunden, Minuten und Sekunden in Datumszeitwert wandeln (≙ h·3600 + m·60 + s)	time(h, m, s)	•	
aktuelle Zeit und Datum in Datumszeitwert wandeln	datetime()	•	
Stunde+Minute+Sekunde+Tag+Monat+Jahr in Datumszeitwert wandeln	datetime(h,m,s,D,M,Y)	•	
aktuelles Jahr	year()	•	
Aus dem Datumszeitwert das Jahr berechnen	year(a)	•	
aktueller Monat	month()	•	
Aus dem Datumszeitwert den Monat berechnen	month(a)	•	
aktueller Tag	day()	•	
Aus dem Datumszeitwert den Tag berechnen	day(a)	•	
aktueller Wochentag (0-6=Sonntag..Samstag)	weekday()	•	
Aus dem Datumszeitwert den Wochentag berechnen	weekday(a)	•	
aktuelle Stunde	hour()	•	
Aus dem Datumszeitwert die Stunde berechnen	hour(a)	•	
aktuelle Minute	minute()	•	
Aus dem Datumszeitwert die Minute berechnen	minute(a)	•	
aktuelle Sekunde	second()	•	
Aus dem Datumszeitwert die Sekunde berechnen	second(a)	•	

Stringfunktionen

Länge Stringregister	strL(nr)	•	
ASCII-Code aus Stringregister	strA(nr, offset)	•	
Unitcode aus Stringregister	strU(nr, offset)	•	

Nummerische Zeichenkette in SINT32 bzw. float wandeln	strV(nr)	•	•
Vergleich zweier Stringregister =0 beide Strings sind gleich >0 erste ungleiche Zeichen in n1 ist größer als in n2 <0 erste ungleiche Zeichen in n1 ist kleiner als in n2	strC(n1, n2)	•	
Vergleich zweier Stringregister von Anfang bis len	strC(n1, n2, len)	•	
Vergleich zweier Stringregister von offset mit Anzahl len Codes	strC(n1, n2, len, offset)	•	
Vergleich zweier Stringregister von offset1 und offset 2 mit Anzahl len Codes	strC(n1, n2, len, offset1, offset2)	•	
Von links nach einem Code suchen =0 nicht gefunden >0 Offset des ersten gefunden Codes	strFL(nr, code)	•	
Von links nach einem Code suchen (ab offset)	strFL(nr, code, offset)	•	
Von rechts nach einem Code suchen	strFR(nr, code)	•	
Von rechts nach einem Code suchen (ab offset)	strFR(nr, code, offset)	•	
Im Stringregister nach einem anderen String eines Stringregister suchen	strFS(n1, n2)	•	
Im Stringregister nach einem anderen String eines Stringregister suchen (ab offset)	strFS(n1, n2, offset)	•	
Prüfung ob Code ein Buchstabe ist	isAL(code)	•	
Prüfung ob Code ein Buchstabe oder Ziffer ist	isAN(code)	•	
Prüfung ob Code ein kleiner Buchstabe ist	isLO(code)	•	
Prüfung ob Code ein großer Buchstabe ist	isUP(code)	•	
Prüfung ob Code ein White-Space-Code ist	isWS(code)	•	
Prüfung ob Code eine dezimale Ziffer ist	isDD(code)	•	
Prüfung ob Code eine hexadezimale Ziffer ist	isDH(code)	•	
Prüfung ob Code eine binäre Ziffer ist	isDB(code)	•	

Array-Funktionen

Maximale Elemente für neuen Array	arE(-1)	•	•
Anzahl an Array Einträge (0= Array existiert nicht)	arE(id)	•	•
Array-Eintrag Wert	arV(id, index)	•	•
Nächsten Array-Eintrag Wert von Lese-Pointer (Inkrement Lese-Pointer)	arV(id)	•	•
Lese-Pointer Index	arR(id)	•	•
Schreib-Pointer Index	arW(id)	•	•

Farbbefehle

Aus einem RGB 24 Bit-Farbwert den roten Kanal extrahieren	getR(x)	•	
Aus einem RGB 24 Bit-Farbwert den grünen Kanal extrahieren	getG(x)	•	
Aus einem RGB 24 Bit-Farbwert den blauen Kanal extrahieren	getB(x)	•	
Aus einzelnen RGB-Bytes eine 24 Bit-Farbwert berechnen	RGB(R, G, B)	•	
RGB 24 Bit-Farbwert aus einem Farb-Rampen / Farbverlauf auslesen	rampRGB(nr, offset)	•	
Opacity aus einem Farb-Rampen / Farbverlauf auslesen	rampO(nr,offset)	•	
RGB 24 Bit-Farbwert von einem dargestellten Pixel auf dem Display auslesen	tftRGB(x,y)	•	

SD-Card (Ordner- und Dateibefehle)

Datei vorhanden? (<Pfad/Dateiname> in Stringregister nr)	fileE(nr)	•	
Dateigröße abfragen (<Pfad/Dateiname> in Stringregister nr)	fileS(nr)	•	
Dateiattribut abfragen (<Pfad/Dateiname> in Stringregister nr)	fileA(nr)	•	
FatTime der Datei abfragen (<Pfad/Dateiname> in Stringregister nr)	fileT(nr)	•	
FatDate der Datei abfragen (<Pfad/Dateiname> in Stringregister nr)	fileD(nr)	•	
Datumszeitwert in FatTime wandeln	fatT(datetime)	•	
Datumszeitwert in FatDate wandeln	fatD(datetime)	•	
Fat-Zeitstempel in Datumszeitwert umrechnen	fattime(Fat-Time, Fat-Date)	•	
Aktuellen Read-Pointer (#FRO) abfragen	fposR()	•	
Aktuellen Read-Pointer vom Fileende aus abfragen (=negativ)	fposR(-1)	•	
Aktuelle Filegröße des offenen Lese-Files abfragen	fposR(1)	•	
Aktuellen Write-Pointer (#FWO) abfragen	fposW()	•	
Aktuellen Write-Pointer vom Fileende aus abfragen (=negativ)	fposW(-1)	•	
Aktuelle Filegröße des offenen Schreib-Files abfragen	fposW(1)	•	
Aktuelle maximale Filegröße	fposW(2)	•	

Maximale Filegröße des Files aus dem Stringregister nr	fileM(nr)	•	
Gesamtspeicherplatz abfragen	memST()	•	
Freien Gesamtspeicherplatz abfragen	memSF()	•	
Objektramspeicherplatz abfragen	memRT()	•	
Freien Objektramspeicherplatz abfragen	memRF()	•	
Maximalen Block im Objektramspeicherplatz abfragen	memRB()	•	

Modulbefehle

Firmware Version	version()		•
Letzte Framrate (fps) auslesen	fps()	•	•
Touchtyp abfragen (=0 kein, =1 resistiv, =2 PCAP)	touchT()	•	•
Screen Breite (#XCV)	scrW()	•	•
Screen Höhe (#XCV)	scrH()	•	•
Screen Breite der Hardware, unabhängig von (#XCV)	scrW(1)	•	•
Screen Höhe der Hardware, unabhängig von (#XCV)	scrH(1)	•	•
Video Breite	vidW()	•	•
Video Höhe	vidH()	•	•
Anzahl Video Objekte	vidC()	•	•
Aktuelle LED Helligkeit	ledB()	•	•
LED Helligkeit von Status (automatische Dimmung) (status 0..2)	ledB(status)	•	•
LED Status (automatische Dimmung) (status 0..2)	ledS()	•	•
Error-String vorhanden?	error()	•	•
Error-String vorhanden? String wird in das Stringregister nr kopiert (#VSL)	error(nr)	•	•
Error-String vorhanden? String wird in das Stringregister nr kopiert und der Error-String gelöscht	error(nr,1)	•	•

Liste der Operatoren nach Priorität

12	()	Klammern / Funktionsaufruf (höchste)
11	++	Register Inkrement

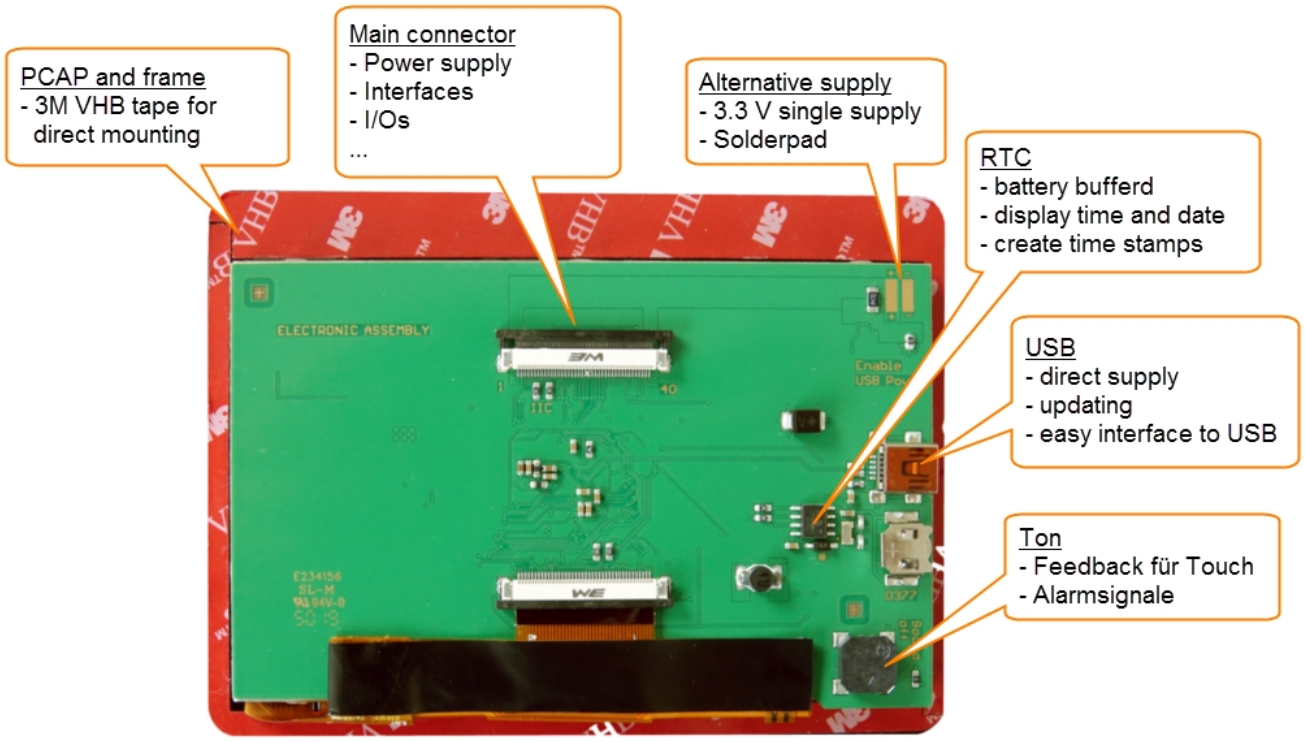
	- -	Register Dekrement
	+	Vorzeichen
	-	Vorzeichen
	!	logisches NICHT
	~	bitweises NICHT
10	*	Multiplikation
	/	Division
9	+	Addition
	-	Subtraktion
8	<<	Linksshift
	>>	Rechtsshift
7	<	kleiner
	<=	kleiner gleich
	>	größer
	>=	größer gleich
6	==	gleich
	!=	ungleich
5	&	bitweises UND
4	^	bitweises exklusives ODER
3	|	bitweises ODER
2	&&	logisches UND
1	||	logisches ODER (niedrigste)

HARDWARE

Die EA uniTFTs-Serie besteht aus einem TFT-Display mit integrierter Hintergrundbeleuchtung welche durch den integrierten LED-Treiber per Softwarebefehl steuerbar ist. So kann im 24h Betrieb die Beleuchtung automatisch gedimmt werden um die Lebensdauer der LEDs zu vergrößern und Strom einzusparen.
Das Modul ist für 3,3 V Betriebspannung ausgelegt. Die Datenübertragung erfolgt seriell im RS232 Format, per SPI, I²C oder direkt mit USB.
Für kleine Steuerungsaufgaben stehen dem Modul zusätzlich 8 frei verwendbare I/Os (erweiterbar auf bis zu 136), 4 Analogeingänge, ein PWM-Ausgang sowie 3 serielle Schnittstellen (RS232, SPI und I²C) zur Verfügung.
Die Module besitze alle ein integriertes kapazitives Touchpanel: Durch Berühren des Displays können hier Eingaben gemacht und Einstellungen per Menü oder Bargraph getätigt werden. Die Beschriftung und Größe sowie die Form der "Tasten" ist flexibel und auch während der Laufzeit änderbar (verschiedene Sprachen, Icons). Das Zeichnen der einzelnen "Tasten", sowie das Beschriften wird von der eingebauten Software komplett übernommen. Das kapazitive Touchpanel besitzt eine gehärteter Glasoberfläche, welches auch mit dünnen Handschuhen bedienbar ist.

Frontansicht (exemplarisch EA uniTFTs043-ATC)

Rückansicht (exemplarisch EA uniTFTs043-ATC)



Pinbelegung

Pinbelegung für den ZIF Stecker. Es handelt sich um einen 40 poligen Stecker im 0,5 mm Raster. Bottom Contact.

Pin	Symbol	I/O	Beschreibung	
1	GND		Ground 0 V	
2	VDD		Power Supply 3,3 V	Stromversorgung bzw. 3,3V Ausgang bei USB Betrieb (r
3	$\overline{\text{RES}}$	I	$\overline{\text{Reset}}$	interner Pull-Up: (10..75 kΩ)
4	$\overline{\text{CS}}$	I	SPI: Chip Select	interner Pull-Up: (1 MΩ)
5	MOSI	I	SPI: MOSI	
6	MISO	O	SPI: MISO	
7	CLK	I	SPI: CLK	interner Pull-Up: (1 MΩ)
8	RxD	I	RS232: Receive Data	interner Pull-Up: (1 MΩ)
9	TxD	O	RS232: Transmit Data	
10	DE	O	RS485: Transmit Enable	
11	SDA	I/O	I²C: Serial Data	interner Pull-Up: (10 kΩ), nachträgliche Änderung mögli
12	SCL	I	I²C: Serial Clock	interner Pull-Up: (10 kΩ), nachträgliche Änderung mögli
13	$\overline{\text{SBUF}}$ TESTMODE	I	Low: Daten im Sendepuffer PowerOn Low: Testmode aktiv	interner Pull-Up: (10 kΩ)
14	$\overline{\text{DPROT}}$	I	High: Small-/Shortprotokoll aktiv Low: deaktiviert	interner Pull-Up: (10 kΩ)
15	A/D 0	I	Analogeingang 0	interner Pull-Down: (1 MΩ)
16	A/D 1	I	Analogeingang 1	interner Pull-Down: (1 MΩ)
17	A/D 2	I	Analogeingang 2	interner Pull-Down: (1 MΩ)
18	A/D 3	I	Analogeingang 3	interner Pull-Down: (1 MΩ)
19	I/O 0.0	I/O	I/O 0.0 (Bit 0)	interner Pull-Up: (1 MΩ), Reset-state: Tri-State, Default
20	I/O 0.1	I/O	I/O 0.1 (Bit 1)	interner Pull-Up: (1 MΩ), Reset-state: Tri-State, Default
21	I/O 0.2	I/O	I/O 0.2 (Bit 2)	interner Pull-Up: (1 MΩ), Reset-state: Tri-State, Default
22	I/O 0.3	I/O	I/O 0.3 (Bit 3)	interner Pull-Up: (1 MΩ), Reset-state: Tri-State, Default
23	I/O 0.4	I/O	I/O 0.4 (Bit 4)	interner Pull-Up: (1 MΩ), Reset-state: Tri-State, Default
24	I/O 0.5	I/O	I/O 0.5 (Bit 5)	interner Pull-Up: (1 MΩ), Reset-state: Tri-State, Default
25	I/O 0.6	I/O	I/O 0.6 (Bit 6)	interner Pull-Up: (1 MΩ), Reset-state: Tri-State, Default
26	I/O 0.7	I/O	I/O 0.7 (Bit 7)	interner Pull-Up: (1 MΩ), Reset-state: Tri-State, Default
27	PWM	O	PWM-Output	
28	DNC	---	Do not connect	Reserved for future use.
29	DNC	---	Do not connect	Reserved for future use.
30	DNC	---	Do not connect	Reserved for future use.
31	DNC	---	Do not connect	Reserved for future use.
32	DNC	---	Do not connect	Reserved for future use.
33	DNC	---	Do not connect	Reserved for future use.
34	BUZZER	O	Tonsignal	PWM Tonsignal für den Anschluss an ein anderen/weite

Spannungsversorgung

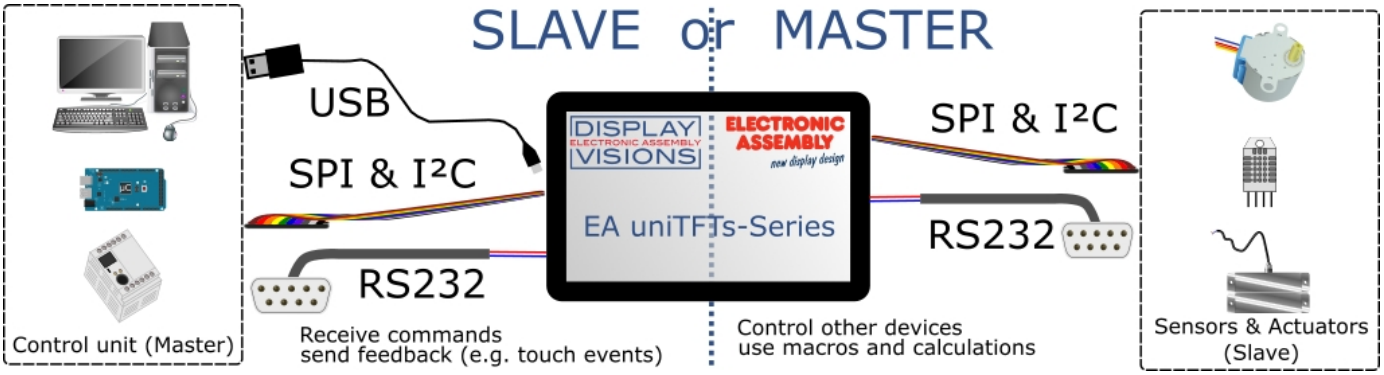
Die Module können über drei verschiedene Weisen mit Spannung versorgt werden:

	Ziff-Connector	**Solderpadas**	**USB**
Spannung	3,3 V	3,3 V	5 V (Power over USB)
USB Lötrbücke	offen	offen	geschlossen
			

Hinweis:
Um Fehlströme zu vermeiden muss die Lötbrücke "USB Power Enable" korrekt gesetzt werden. Default ist die Lötbrücke geschlossen. Damit ist der interne Spannungsregler aktiv und generiert 3,3 V aus der angeschlossenen USB-Versorgung. Sollte nun extern zusätzlich 3,3 V zugeführt werden kommt es zu Fehlströmen.

Serielle Schnittstellen

Das Modul verfügt über 4 serielle Schnittstellen, RS232, SPI, I²C und ein USB Port. Außer der USB-Schnistelle können die anderen Schnittstellen Ihr Verhalten verändern:
Sie können entweder die Verbindung zu einem externen Host also zu einer übergeordneten Steuerung oder als Masterschnittstelle verwendet werden.
Default sind alle Interfaces als Slave parametrisiert und nehmen die Befehlskommandos entgegen.
Als Masterschnittstelle parametrisiert, ermöglicht das Steuern externer Sensoren und Aktoren. Das Displaymodul verhält sich hier als Master.
Wie bereits beschrieben verhalten sich die Schnittstellen per default als Slave Schnittstelle und nehmen Befehle entgegen. Sobald aber ein Master-Schnistellenbefehl (#H...) ausgeführt wird, bekommt die Schnittstelle Masterfunktionen.



RS232

RS232 ist ein Standard für eine serielle Schnittstelle.

Das EA uniTFT bietet eine RS232 Schnittstelle die als Slave (default) oder Master parametrisiert werden kann: Als Slave-Schnittstelle verwenden Sie die Schnittstelle um mit dem Display zu arbeiten und zu kommunizieren. Alles was empfangen wird, wird als Kommando interpretiert (mit und ohne Small-Protokoll). Möchten Sie über eine RS-232 x-beliebige Daten senden und empfangen, verwenden Sie die Master-Schnittstelle. Diese bedienen Sie über die #H Befehle:

Die Übertragung erfolgt seriell asynchron. Die Daten werden also in ein Bitstrom gewandelt und übertragen. Es existiert keine Taktleitung, jeder Busteilnehmer muss also mit der selben Übertragungsrate (sogenannte Baudrate) arbeiten. RS232 ist eine Spannungsschnittelle, die Dateninformationen werden durch Spannungspegel übertragen. In der PC-Welt und Industriesteuerungen sind Pegel von +12V bzw. - 12V als Standard definiert. Innerhalb von Platinen bzw. in Mikrokontrollersteuerungen wird mit 0V bzw. VDD (im Fall des EA uniTFTs-Serie 3,3 V) gearbeitet. Um die Signalpegel anzupassen gibt es einige Möglichkeiten in Form von Levelshiftern (z.B. ICL232, MAX202). RS232 besteht aus "hörenden" und "sprechenden" Leitungen, die zwischen den beiden Teilnehmern gekreuzt werden.
In der EA uniTFTs-Serie ist das Datenformat fest auf 8-N-1 festgelegt:

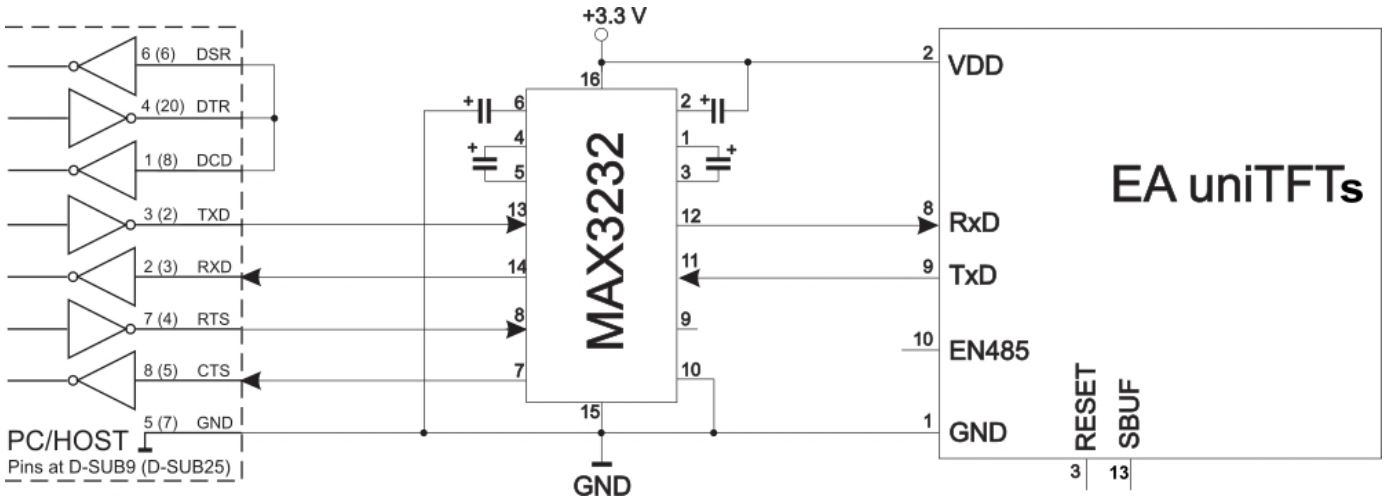
Das EA uniTFTs-Serie kann mit folgenden Baudraten arbeiten:

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

Die Schnittstellenparameter zur übergeordneten Steuerung werden mit dem Befehl #XCR eingestellt (Slave), die der Masterschnittstelle wird mit dem Befehl #HRP gesetzt. Alternativ kann der Befehl direkt in die Boot-Datei <start.emc> geschrieben werden:

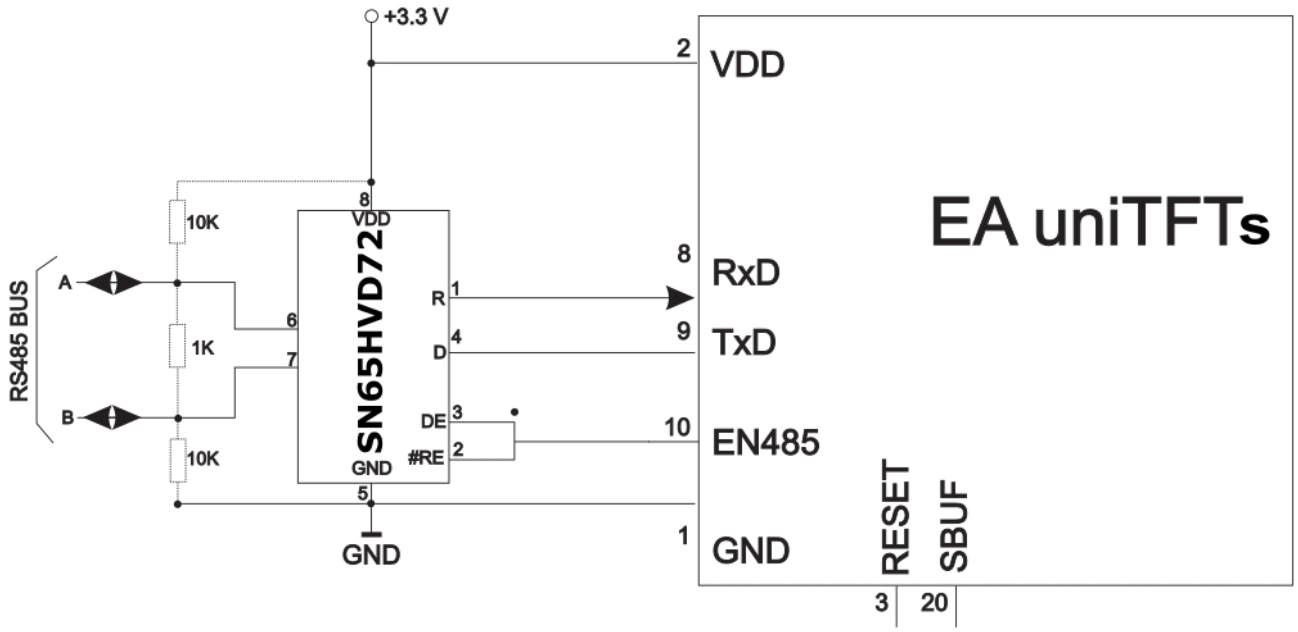
Applikationsbeispiel



RS232 V24 - Verbindung zu einem PC (EA uniTFTs)

RS485 / RS422

Mithilfe eines externen Wandlerbausteins können alle Displays der EA uniTFTs-Serie auch an einer RS-485 und RS-422 Schnittstelle betrieben werden.



RS485 - Verbindung zu einer SPS (EA uniTFTs)

SPI

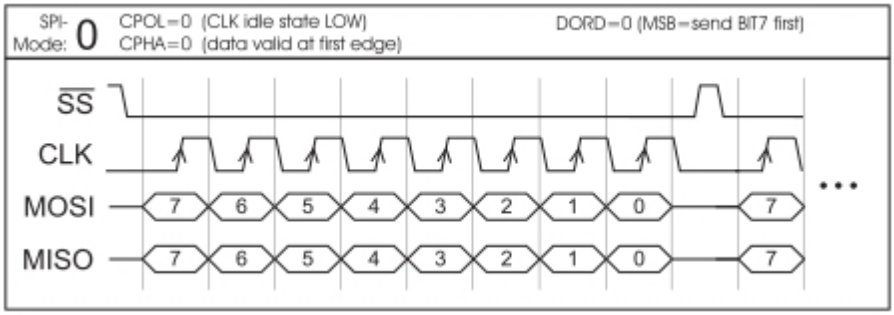
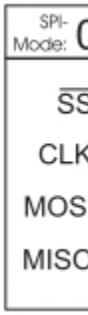
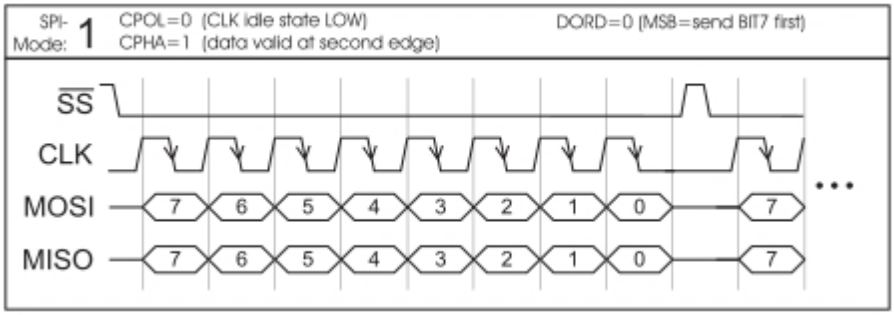
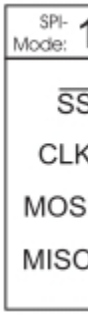
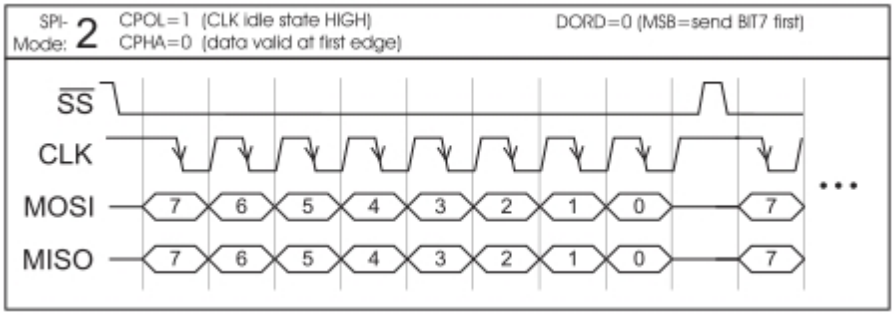
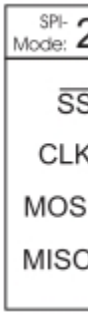
Das **S**erial **P**eripheral **I**nterface ist ein Bussystem für eine serielle synchrone Datenübertragung zwischen verschiedenen ICs.

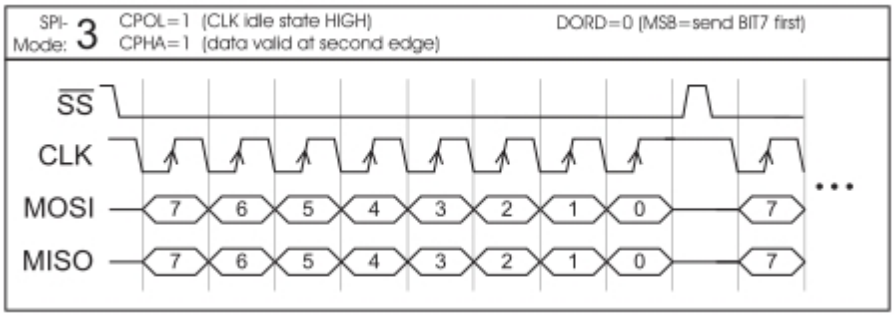
Das EA uniTFT bietet eine SPI Schnittstelle: Als default ist es eine Slave-Schnittstelle. Verwenden Sie sie um mit dem Display zu arbeiten und zu kommunizieren. Alles was hierüber empfangen wird, wird als Kommando interpretiert (mit und ohne Small-Protokoll). Möchten Sie über eine SPI x-beliebige Daten senden und empfangen, verwenden Sie diese als Master-Schnittstelle. Diese bedienen Sie über die #H Befehle:

Der SPI-Bus besteht aus folgenden Leitungen:

- MOSI (**M**aster **O**ut → **S**lave **I**n) auch SDO (Serial Data Out) oder DO
- MISO (**M**aster **I**n ← **S**lave **O**ut) auch SDI (Serial Data In) oder DI
- SCK (**S**erial **C**lock) - Schiebetakt
- SS (**S**lave **S**elect → Adressierung des Partners) auch CS (Chip Select)

SPI arbeitet mit einem bidirektionalem Übertragungsprinzip, es werden also zeitgleich Daten zwischen den Partner ausgetauscht. Jede Kommunikation wird vom Master mit Hilfe der SCK-Leitung bestimmt.
Das Protokoll für die Datenübertragung ist bei SPI nicht festgelegt, daher gibt es verschiedene Einstellmöglichkeiten. Diese werden durch die Parameter Clock Polarity, Clock Phase sowie Data Order festgelegt. Voreingestellt ist der SPI-Mode 3 mit DORD=0. Über den Befehl #XCS (Slave) bzw. #HSP (Masterschnittstelle) kann er auf einen der folgenden Modi 0..3 umgeschaltet werden. Alternativ kann der Befehl direkt in die Boot-Datei <start.emc> geschrieben werden:

Mode	CPOL	CPHA	DORD (0) - MSB First	DORD (1)
0	0	0		
1	0	1		
2	1	0		

3	1	1	

Die maximale Frequenz des Moduls im Slave-Modus ist 1 MHz. Die Masterschnittstelle kann ebenfalls bis zu 1 MHz übertragen. Das Modul benötigt eine bestimmte Zeit um die Daten bereit zu stellen; deshalb muss vor den zu lesenden Byte mindestens **50 µs** gewartet werden (keine Aktivität auf der SCK-Leitung).

I²C

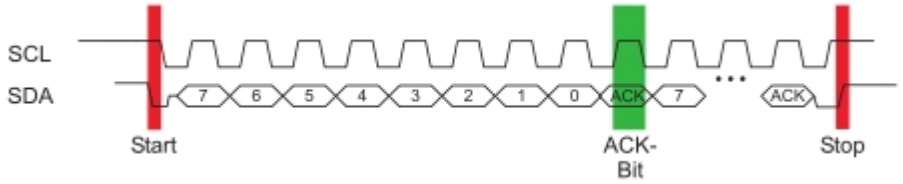
I²C steht für **I**nter-**I**ntegrated **C**ircuit und ist ein von Phillips entwickelter serieller Datenbus.

Das EA uniTFT bietet eine I²C Schnittstelle; Als Default ist die Schnittstelle eine Slave-Schnittstelle, verwenden Sie um mit dem Display zu arbeiten und zu kommunizieren. Alles was hierüber empfangen wird, wird als Kommando interpretiert (mit und ohne Small-Protokoll). Möchten Sie über eine I²C-Schnittstelle x-beliebige Daten senden und empfangen, verwenden Sie es als Master-Schnittstelle. Diese bedienen Sie über die #H Befehle:

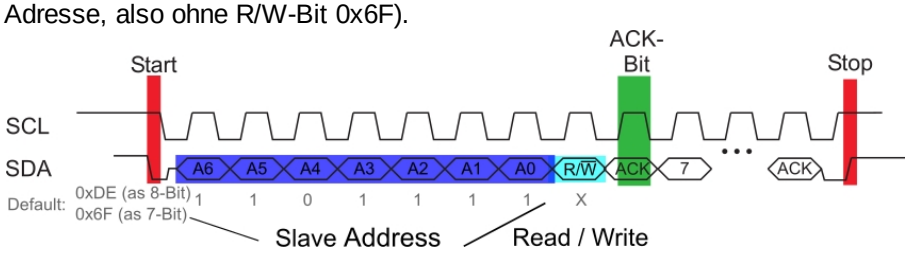
Der als Master-Slave-Bus konzipierte Bus benötigt 2 Signalleitungen:

- SCL (**S**erial **C**lock **L**ine)
- SDA (**S**erial **D**ata **L**ine)

Die elektrische Spezifikation sieht vor, dass beide Leitungen mit einem Pull-Up-Widerstand an VDD abgeschlossen werden, denn sämtliche an dem Bus angeschlossene Geräte haben Open-Collector-Ausgänge. Der Bustakt wird immer durch den Master vorgegeben, der die gesamte Kommunikation bestimmt:



Nach der Startbedingung folgt in einem Übertragungsprotokoll immer die Slaveadresse. Hierbei ist das Bit 0 das sogenannte R/W-Bit und bestimmt ob vom Slave gelesen (1) oder Daten übermittelt (0) werden sollen. Der Datenaustausch erfolgt bis der Master die Stopbedingung ausführt. Genauere Informationen sind in der I²C Spezifikation zu finden. Die voreingestellte I²C-Bus Adresse ist 0xDE (als 8-Bit Adresse inklusive R/W bit, als 7-Bit Adresse, also ohne R/W-Bit 0x6F).



Über den Befehl #XCI bzw. #HIP (Masterschnittstelle) kann auf eine x-beliebige andere Adresse umgestellt werden. Alternativ kann der Befehl direkt in die Boot-Datei <start.emc> geschrieben werden.
Die maximale Frequenz des Moduls im Slave-Modus ist 400 kHz, auf dem Masterschnittstelle können bis zu 1 MHz übertragen werden. Das Modul benötigt eine bestimmte Zeit um die Daten bereit zu stellen; deshalb muss vor den zu lesenden Byte mindestens **50 µs** gewartet werden (keine Aktivität auf der SCL-Leitung).

USB

Der **U**niversal **S**erial **B**us ist ein serielles Bussystem zur Verbindung mit einem Computer oder anderem Gerät. Er basiert auf einer differentiellen Datenübertragung. Die Bustopologie ist eine strikte Master-Slave-Kommunikation (Ausnahme: On the Go Geräte). Im Fall der EA uniTFTs-Serie muss immer der PC/Master die Kommunikation leiten. Das Modul verfügt über eine CDC Geräteklasse und meldet sich damit als virtuelle serielle COM-Schnittstelle am PC an:

Beschreibung	Wert
Geräteklasse	2
USB Vendor ID	0x2DA9
USB Product ID	0x2454
Gerätebeschreibung	EA uniTFT

Um das Modul zu programmieren, Einstellungen vorzunehmen oder für erste Tests empfehlen wir die USB-Schnittstelle. Sie ist einfach anzuschließen, schnell und es müssen keine Schnittstellenparameter angepasst werden. Der Windowstreiber kann direkt von der Homepage unter http://www.lcd-module.de/fileadmin/downloads/EA_CDCdriver_V5_2.zip heruntergeladen werden.

Hinweis:
Das Protokoll ist bei USB immer zu verwenden. Es ist nicht möglich die USB-Schnittstelle zu nutzen und das Protokoll zu deaktivieren, Pin 22 darf nicht auf GND gelegt werden. Die hohe Geschwindigkeit am USB führt sonst zu Pufferüberläufen, die nur durch das Protokoll verhindert werden.

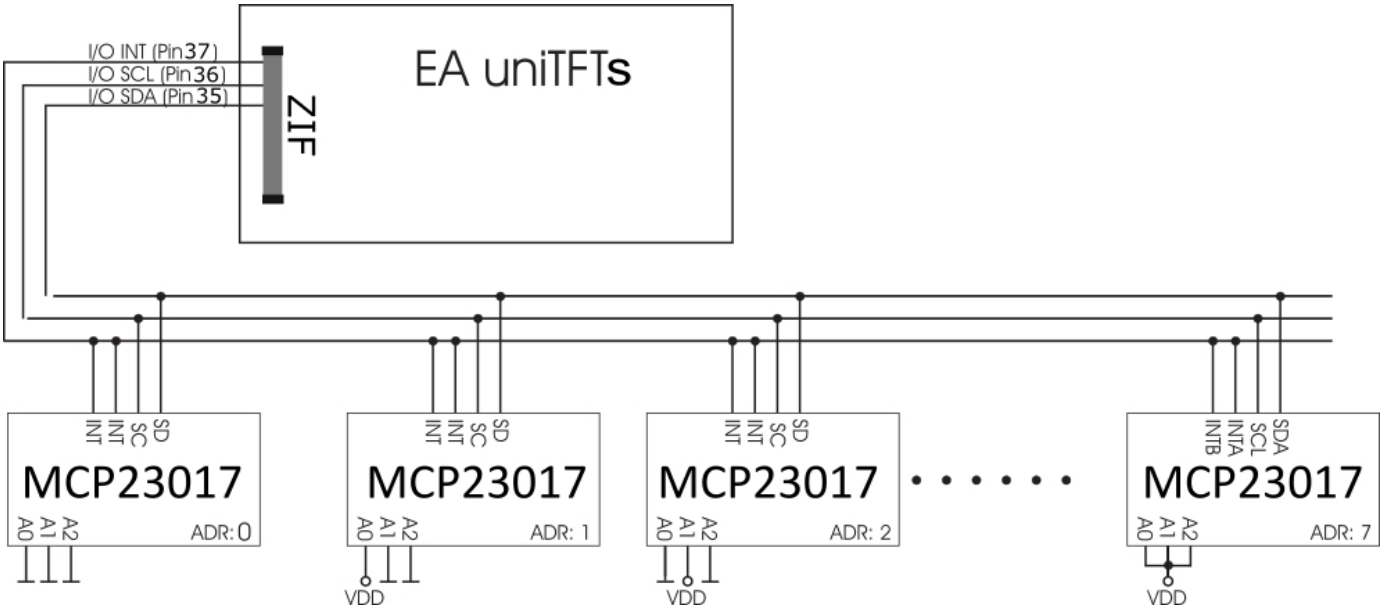
Touchpanel

Die Module haben alle ein optisch gebondetes kapazitives Touchpanel, welches auf für die Montage verwendet wird. Durch Berühren des Displays können hier Eingaben gemacht und Einstellungen per Menü oder Bargraph getätigt werden. Die Beschriftung der "Tasten" ist flexibel und auch während der Laufzeit änderbar (verschiedene Sprachen, Icons). Das Zeichnen der einzelnen "Tasten", sowie das Beschriften wird von der eingebauten Software komplett übernommen.

I/O - digitale Ein- und Ausgänge

Das Modul verfügt standardmäßig über 8 digitale I/O's (CMOS Pegel, nicht potentialfrei). Der Eingangsspannungsbereich beträgt 0..3,3V. Alle 8 I/O's sind nach einem Reset als Eingang mit einem 1 MΩ Weak Pull-up geschaltet.
Anmerkung: Die Logik ist nicht für zeitkritische Vorgänge ausgelegt; d.h. es handelt sich nicht um ein Echtzeitbetriebssystem.

Durch den Einsatz von einem oder mehreren externen (maximal 8) Portexpandern MCP23017-E (16 I/O's pro Baustein) kann die Anzahl der gesamten I/O auf bis zu 136 erweitert werden. Dazu werden die Port-Expander an den Pins 35-37 angeschlossen (siehe Applikationsbeispiel).



Die maximale Leistung des MCP23017-E beträgt 700mW. Die maximale Strombelastung für einen einzelnen Pin liegt bei 25mA, womit z.B. direkt eine low current LED betrieben werden kann. Sollte eine höhere Last vorliegen, muss mit einer geeigneten Schaltung der I/O-Strom verstärkt werden, z.B. durch einen externen Transistor oder MOSFET. Sie auch Elektrische Spezifikation

Die Übersicht über die Softwarebefehle zu den I/O's finden Sie unter dem Punkt 'I/O Port'.

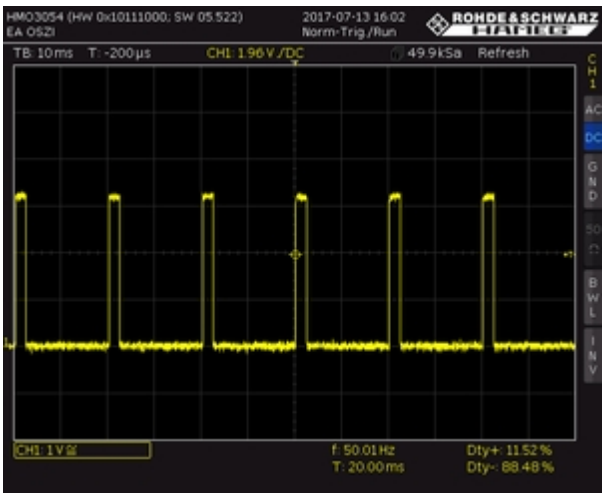
Analog Input

Das Modul verfügt über 4 analoge Eingänge mit einer Auflösung von 12 Bit und einem Spannungsbereich von 0..VDD. Der Spannungsbereich kann durch einen externe Spannungsteiler oder Messverstärker beliebig erweitert werden. Jeder Eingang hat einen Bezug zu GND und einen Eingangswiderstand von ca. 1 MΩ. Die absolute Genauigkeit liegt bei 11 Bit, als Referenz dient VDD/2.
Dadurch ist das Display in der Lage analoge Spannungen zu messen, und z.B. anzuzeigen oder zur weiteren Bearbeitung zu speichern. Das Über- oder Unterschreiten eines Grenzbereiches kann auch z.B. einen Alarm auslösen.

Die Übersicht über die Softwarebefehle rund um die Analogeingänge finden Sie unter dem Punkt 'Analog Input'

PWM Output

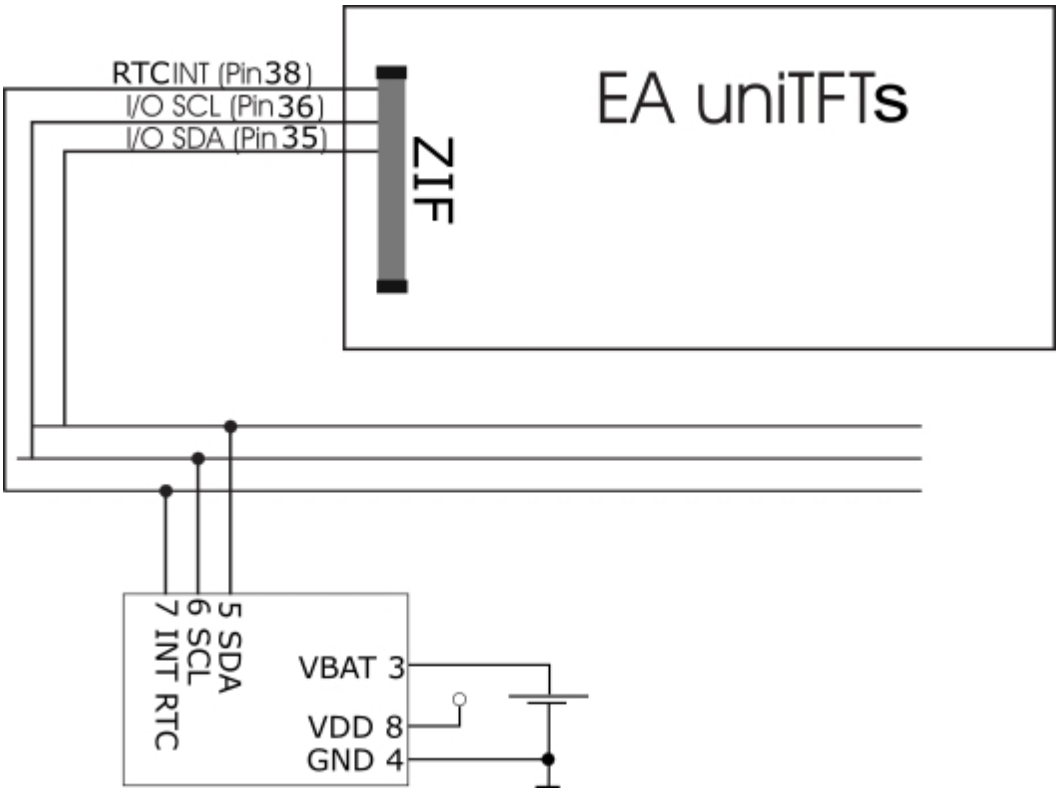
Das Modul verfügt über die Möglichkeit über ein PWM-Signal (Pulsweitenmodulation) externe Komponenten anzusteuern. Dabei wird bei konstanter Frequenz (einstellbar von 2 Hz bis 1 MHz #HFQ) das Tastverhältnis eines rechteckigen Impulses geändert. Durch die Modulation ändert sich das Verhältnis zwischen An- und Ausschaltzeit und somit die Charakteristik des Ausgangssignals. Auf diese Art können elektromechanische Bauteile wie z.B. Motoren angesteuert werden oder auch eine quasi-analoge Spannung erzeugt werden. Die Variation des Tastverhältnisses sorgt dann für eine geringe Motordrehzahl/Spannung bei kurzer Anschaltzeit oder eine hohe Motordrehzahl/Spannung bei langer Anschaltzeit. Die Ausgangspegel liegen bei 0V und VDD.



Uhrzeit / RTC

EA uniTFTs035 und uniTFTs043 besitzen eine eingebaute RTC mit Batteriepuffer. Neben einem Zeitstempel auf log-Dateien kann die Zeit und das Datum auch direkt angezeigt werden.
Bei Auslieferung wird die Uhrzeit auf die Mitteleuropäische Zeit (CET / MEZ) gesetzt. Je nach Einsatzort muss die Zeit auf die Gültige vor Ort eingestellt (#WTD) werden. Bei Spannungseinbruch oder Abschalten des Moduls wird der Uhrenbaustein durch eine Knopfzelle (D377) gepuffert, sodass die Zeit auch im abgeschalteten Zustand weiter läuft.
Durch Bauteiltoleranzen und Temperaturschwankungen sind Abweichungen von bis zu 0,02% möglich. Die Abweichung kann durch wiederholtes Abgleichen der Zeit (#WTD) in größeren Abständen verringert werden. Sie wird dadurch immer genauer.

An die kleineren Displays EA uniTFTs020 und uniTFTs028 kann extern eine RTC direkt anschließen. Wir empfehlen den Baustein **MCP7940N:**



Speicher

Das Modul verfügt über einen integrierten FLASH-Speicher. Die Speichergröße beträgt 31 MByte.
In diesem Speicher werden alle Daten abgelegt, sowohl zur Laufzeit generierte wie z.B. log-Files aber auch die Projektdaten, darunter z.B. Makro-Files, Bilder, Animationen und Icons.

Achtung:
FLASH-Speicher haben bauartbedingt begrenzte Lösch/Schreibzyklen. Das im uniTFTs eingesetzte Speichermodul kann typischerweise 100.000 Zyklen sicher ausführen. Um Daten zu schreiben kann es sein, dass ein Speicherblock gelöscht werden muss. Typischerweise werden für das Löschen 30 ms benötigt, es können aber bis zu 400 ms sein. Das ist bei der Makroabfolge zu beachten, wenn Schreibbefehle ausgeführt werden.

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/m²

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/m²

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/m²

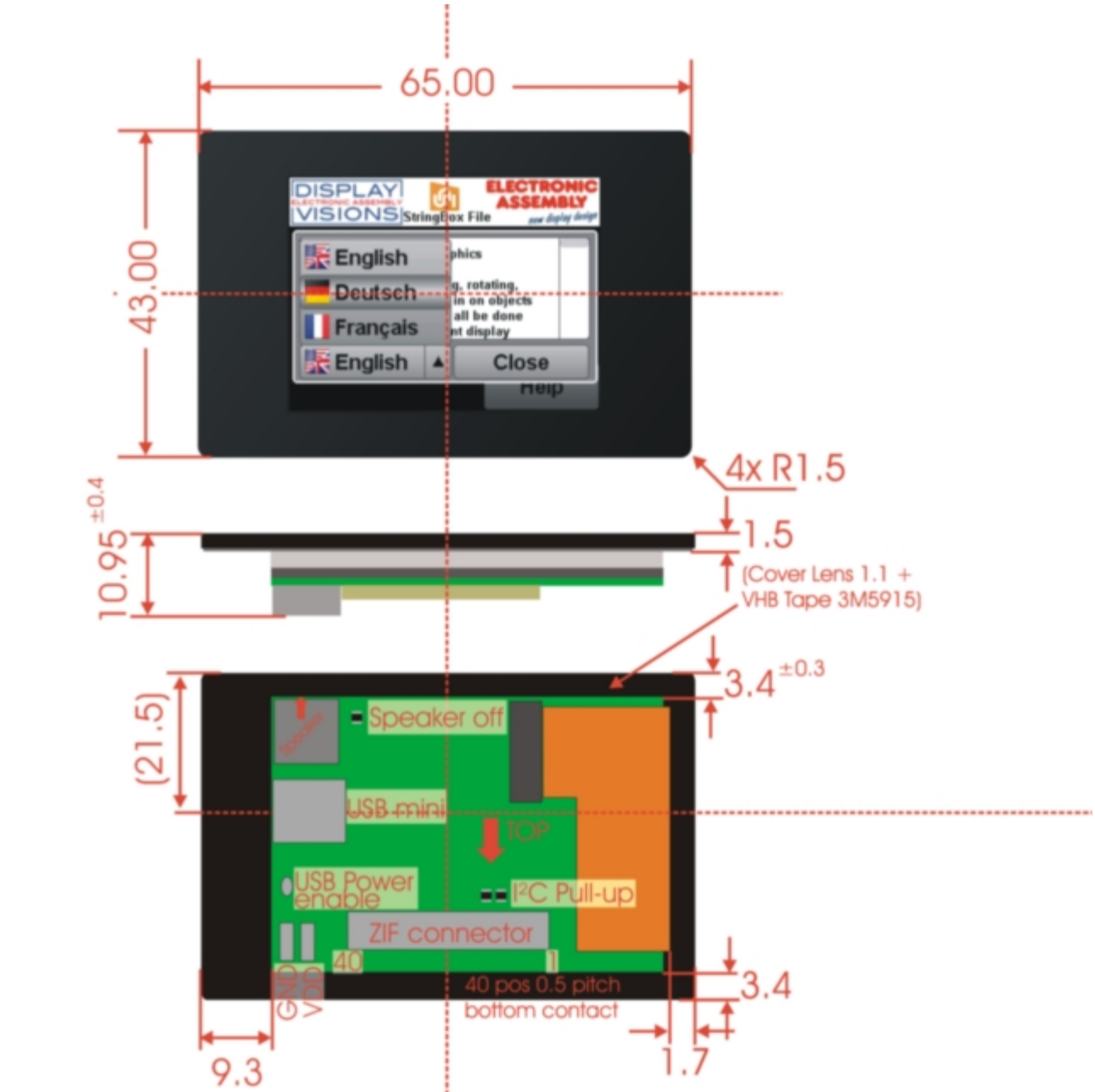
Elektrische Spezifikation EA uniTFTs043-ATC

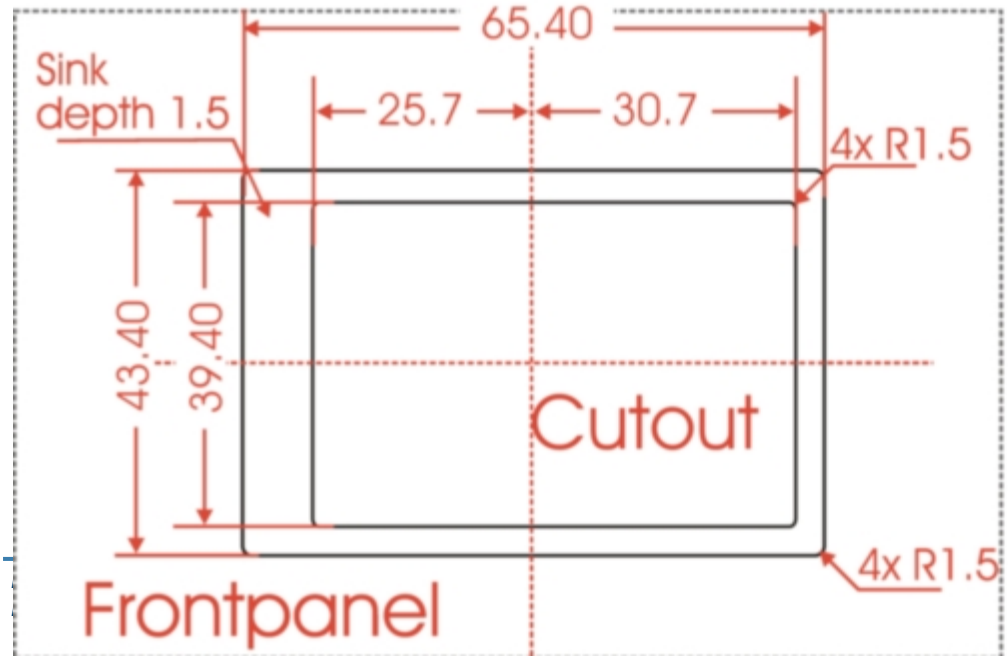
Value	Condition	min.	typ.	max.	Unit
Supply current 3.3 V	Backlight 0%		131		mA
	Backlight 100%		362		mA
	Backlight 150%		562		mA
Supply current USB (5 V)	Backlight 0%		89		mA
	Backlight 100%		252		mA
	Backlight 150%		382		mA
Brightness (100%)	with PCAP	750	820		cd/m²

Elektrische Spezifikation Allgemein

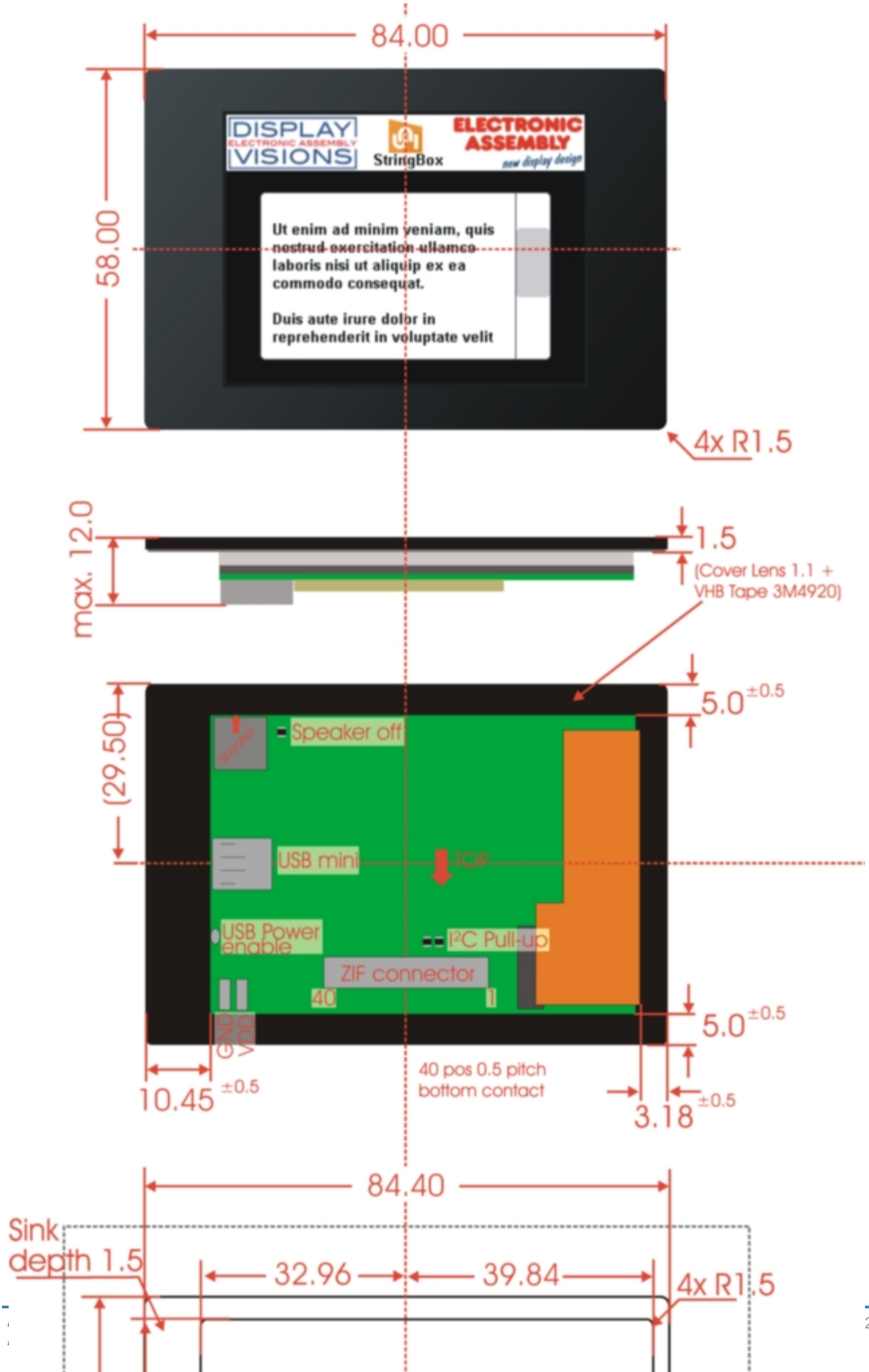
Value	Condition	min.	typ.	max.	Unit
Operating temperature		-20		70	°C
Storage temperature		-30		80	°C
Storage humidity	@ 60°C			90	% RH
Operating voltage		3.1	3.3	3.5	V
Input low voltage (except USB,I/O)		-0,3	0	0.3*VDD	V
Input high voltage (except USB,I/O)		VDD*0.7		VDD+0.3	V
Output low voltage (except 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)
I²C-bus pull-up				10	kΩ

Maßzeichnung EA uniTFTs020-ATC

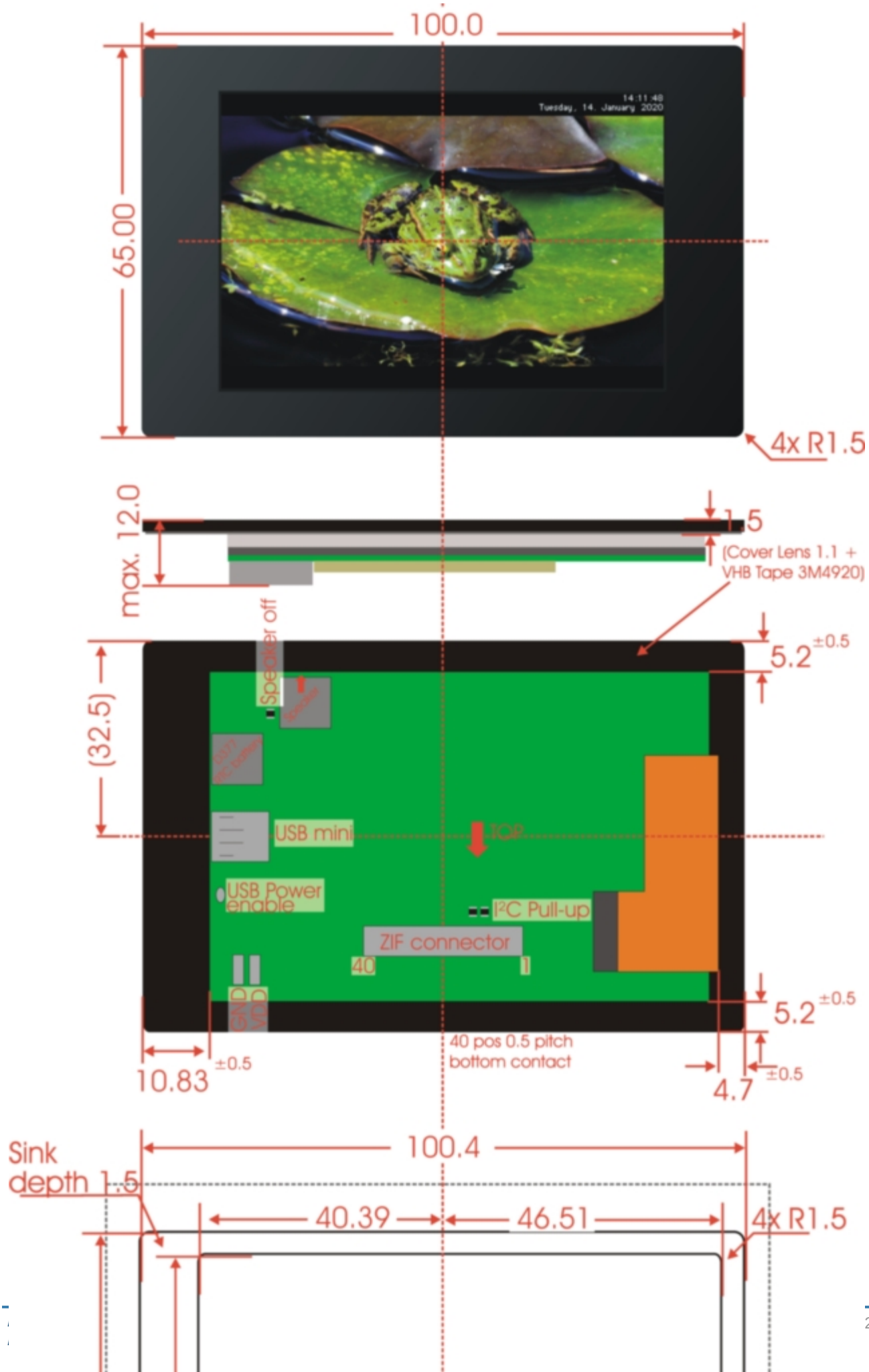

65.00
43.00
DISPLAY VISIONS
ELECTRONIC ASSEMBLY
StringTox File
English
Deutsch
Français
English
Close
Help
4x R1.5
10.95 ±0.4
1.5
(Cover Lens 1.1 +
VHB Tape 3M5915)
3.4 ±0.3
(21.5)
Speaker off
USB mini
TOP
USB Power enable
I²C Pull-up
ZIF connector
40
40 pos 0.5 pitch
bottom contact
3.4
9.3
1.7


65.40
Sink depth 1.5
25.7
30.7
4x R1.5
43.40
39.40
Cutout
4x R1.5
Frontpanel

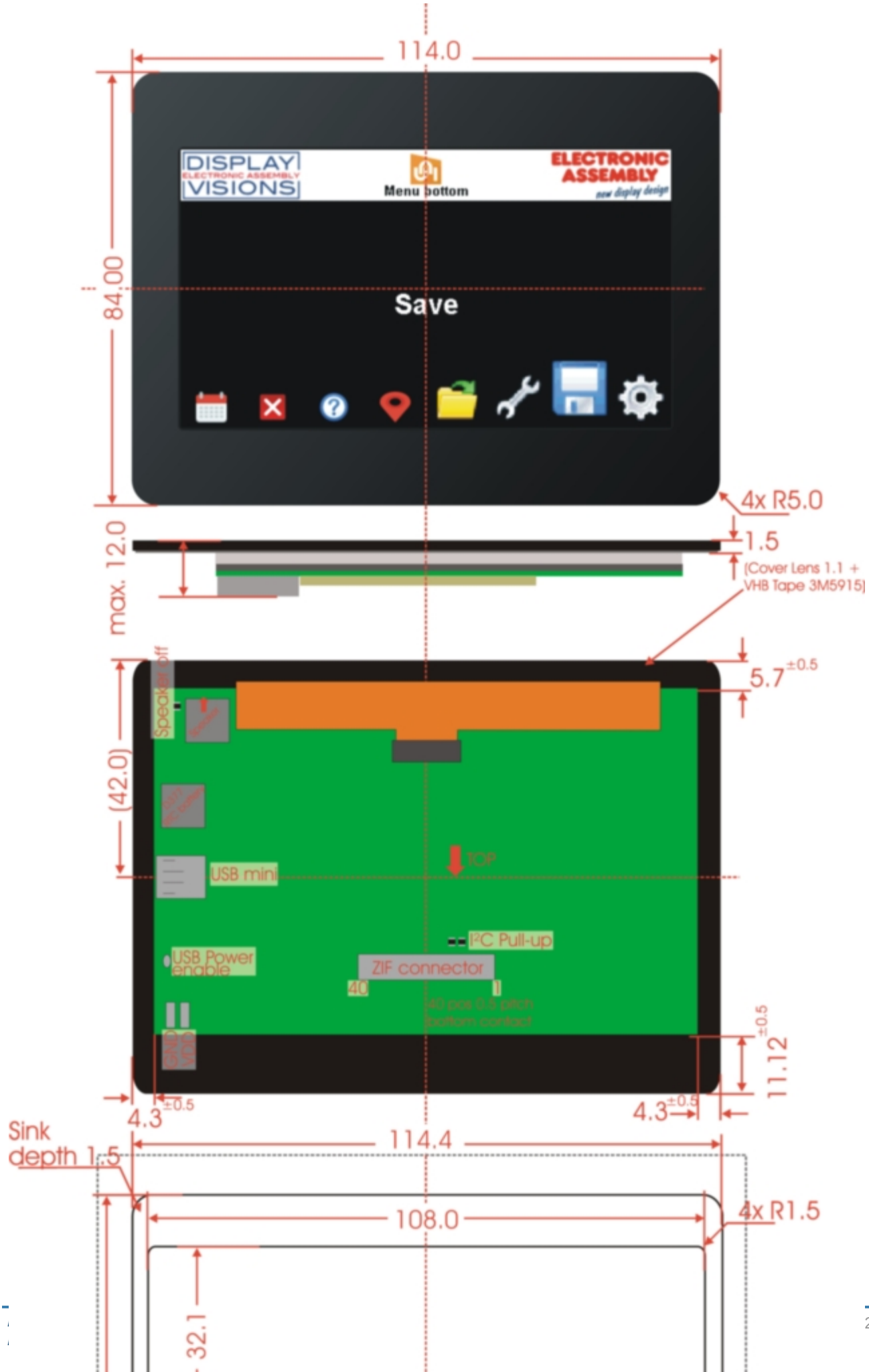
Maßzeichnung EA uniTFTs028-ATC


84.00
58.00
DISPLAY VISIONS
StringBox
ELECTRONIC ASSEMBLY
new display design
Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
Duis aute irure dolor in reprehenderit in voluptate velit
4x R1.5
max. 12.0
1.5
(Cover Lens 1.1 + VHB Tape 3M4920)
5.0 ±0.5
(29.50)
Speaker off
USB mini
TOP
USB Power enable
I²C Pull-up
ZIF connector
40
1
GND
VCC
5.0 ±0.5
10.45 ±0.5
40 pos 0.5 pitch bottom contact
3.18 ±0.5
84.40
Sink depth 1.5
32.96
39.84
4x R1.5

Maßzeichnung EA uniTFTs035-ATC


100.0
65.00
14:11:48
Tuesday, 14. January 2020
4x R1.5
max. 12.0
1.5
(Cover Lens 1.1 +
VHB Tape 3M4920)
(32.5)
5.2 ±0.5
Speaker off
USB mini
TOP
USB Power enable
I²C Pull-up
ZIF connector
40
1
GND
5V
5.2 ±0.5
10.83 ±0.5
40 pos 0.5 pitch
bottom contact
4.7
±0.5
Sink depth 1.5
100.4
40.39
46.51
4x R1.5

Maßzeichnung EA uniTFTs043-ATC


114.0
84.00
DISPLAY VISIONS
Menu bottom
ELECTRONIC ASSEMBLY
Save
4x R5.0
1.5
(Cover Lens 1.1 + VHB Tape 3M5915)
max. 12.0
5.7 ±0.5
(42.0)
Speaker off
USB mini
TOP
I²C Pull-up
USB Power enable
ZIF connector
40
40 pos 0.5 pitch bottom contact
11.12 ±0.5
4.3 ±0.5
4.3 ±0.5
Sink depth 1.5
114.4
108.0
4x R1.5
32.1

uniTFTDESIGNER - DESIGNSOFTWARE

Mit der Windows Designsoftware uniTFTDesigner lassen sich kinderleicht Bildschirmlayouts erstellen (WYSIWYG) und mit Hilfe des Makro-Editors können Funktionsabläufe definiert werden. Die Eigenschaften von Objekten (Postion, Größe, Winkel) sind einfach einstellbar. Auch die Touchfunktionalität wird durch den uniTFTDesigner voll unterstützt, so können Radiogroups, Schieberegeler, Bargraphen und einfache Touchbuttons erstellt werden.
Bei Betätigung eines Touchbuttons kann auf eine andere Seite umgeschaltet oder ein Makro gestartet werden.
Ein integrierter Simulator zeigt sofort die reale Ansicht samt Funktionalität. Auch die digitalen und analogen Ein- und Ausgänge können darüber direkt simuliert werden.
Eine umfassende Debug-Funktion und die integrierte Hilfe runden dieses Tool ab.

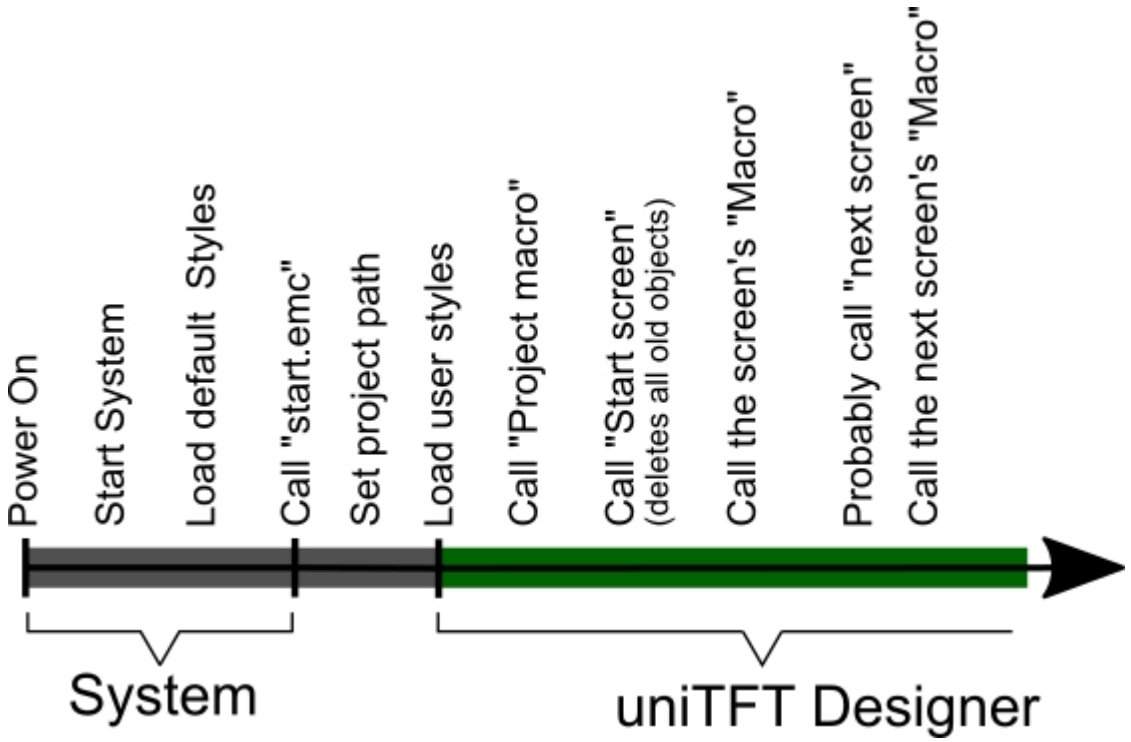
Die Oberfläche

Hilfe und Erkläuterungen

Unter dem Menüpunkt "Help" bzw. F1 finden Sie Informationen über den Versionsstand (About-Dialog), sowie dieses Hilfe-File.
Im Macro Editor gelangen Sie zur spezifischen Hilfe des jeweiligen Befehls durch Drücken der Tastenkombination F1.
Eine große Auswahl an Beispielprojekten finden Sie auf dem Home bzw. Willkommen-Screen.

Abarbeitungsreihenfolge: Makros, Screen

Die Abarbeitung der Makros und Screens folgt immer diesem Muster:



Hinweis: uniTFTDesigner löscht normalerweise alle Definitionen und vorhandenen Objekte (#ODI 0) bevor ein neuer Screen aufgebaut wird, außer dies wird in "Delete behaviour" in den Screen-Eigenschaften abgeschaltet oder auf die Objekte begrenzt.

Tastenkürzel

Um schnelleres Arbeiten im uniTFTDesigner zu ermöglichen sind im folgenden Tastenkürzel angegeben:

Globale short cuts

Hilfe öffnen	F1
Befehlshilfe öffnen	F1
Programm schließen	Ctrl + Q / Alt + F4
Neues Projekt	Ctrl + N
Projekt öffnen	Ctrl + O
Projekt schließen	Ctrl + F4
Projekt speichern	Ctrl + S
Projekt speichern unter	Ctrl + Shift + S
Backup-Zip erstellen	Ctrl + Shift + Z
Backup-Zip laden	Ctrl + Shift + B
Recovery-Point laden	Ctrl + Shift + R
Kopieren (Objekte, Screens, Styles usw.)	Ctrl + C
Einfügen	Ctrl + V
Alles Auswählen	Ctrl + A
Zoom in (Screen)	Ctrl + +
Zoom out (Screen)	Ctrl + -
Undo	Ctrl + Z
Re-do	Ctrl + Y
Editieren	F2
Deploy zur Hardware (EA uniTFTs-Serie)	F5
Deploy zum Simulator	F6
Schnittstelleneinstellung / Simulatorpfad	Ctrl + Alt + P
Terminal öffnen	Alt + T
Default View (Ansicht Wiederherstellen)	Ctrl + 0
Start uniEXPLORER	Ctrl + Alt + E
Start Hardcopytool	Ctrl + Alt + H
Start Debugger	Ctrl + Alt + D

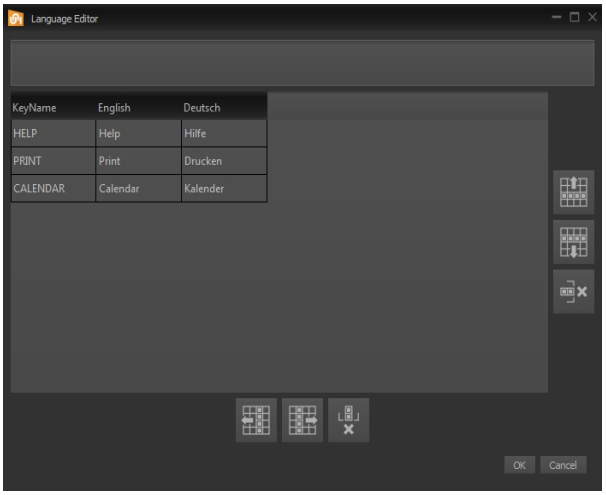
WYSIWYG - Grafische short cuts

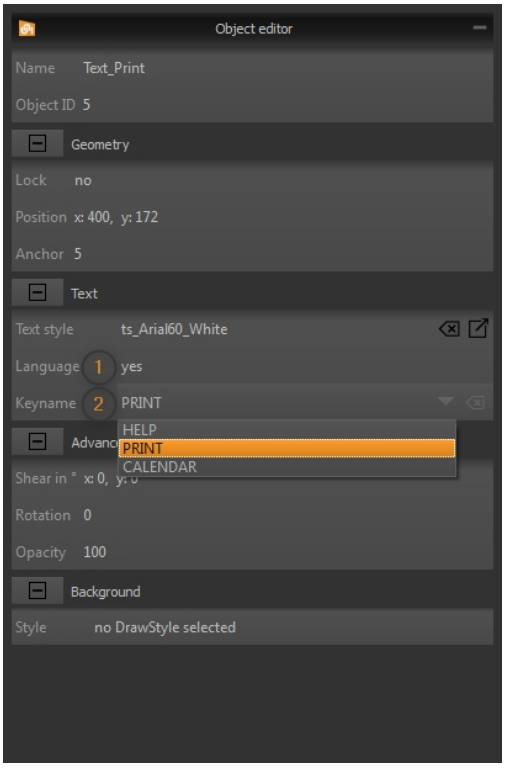
select / deselect multiple items	Shift + Linke Maustaste

select / deselect übereinander liegender Objekte	Ctrl + Linke Maustaste
select / deselect Objekte innerhalb einer Gruppe	Alt + Linke Maustaste
Layer: nach oben	Ctrl + Pfeil hoch
Layer: nach unten	Ctrl + Pfeil runter
Layer: ganz nach oben	Ctrl + Shift + Pfeil hoch
Layer: ganz nach unten	Ctrl + Shift + Pfeil runter
Gruppe erstellen	Ctrl + G
Gruppe auflösen	Ctrl + U
Objekt um 1 Pixel bewegen	Pfeiltasten
Objekt um Grid size bewegen	Shift + Pfeiltasten
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

Der uniTFTDesigner unterstützt die Mehrsprachigkeit der EA uniTFTs-Serie. Im Language Editor (**Project -> Language Editor**) können mehrere Sprachen und KeyNames definiert und die jeweilige Übersetzung eingetragen werden. Die Übersetzungsdatei (Language.csv) befindet sich im Data-Ordner des Projektes und kann an Übersetzungsbüros weitergegeben werden

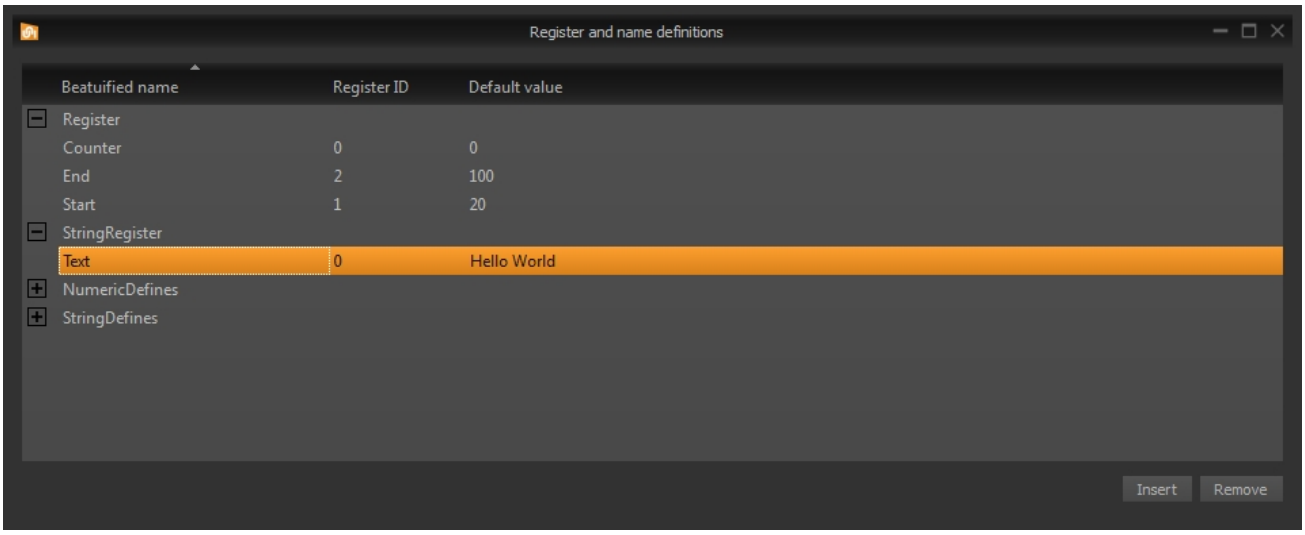




1. Um bei Objekten (z.B. Text, Button, SpinBox, …) die Mehrsprachigkeit nutzen zu können muss dies in den Objekteinstellungen aktiviert werden.
2. Nun kann mit dem KeyName die erforderliche Übersetzung ausgewählt werden und wird zur Laufzeit in die Eingestellte Sprache übersetzt.

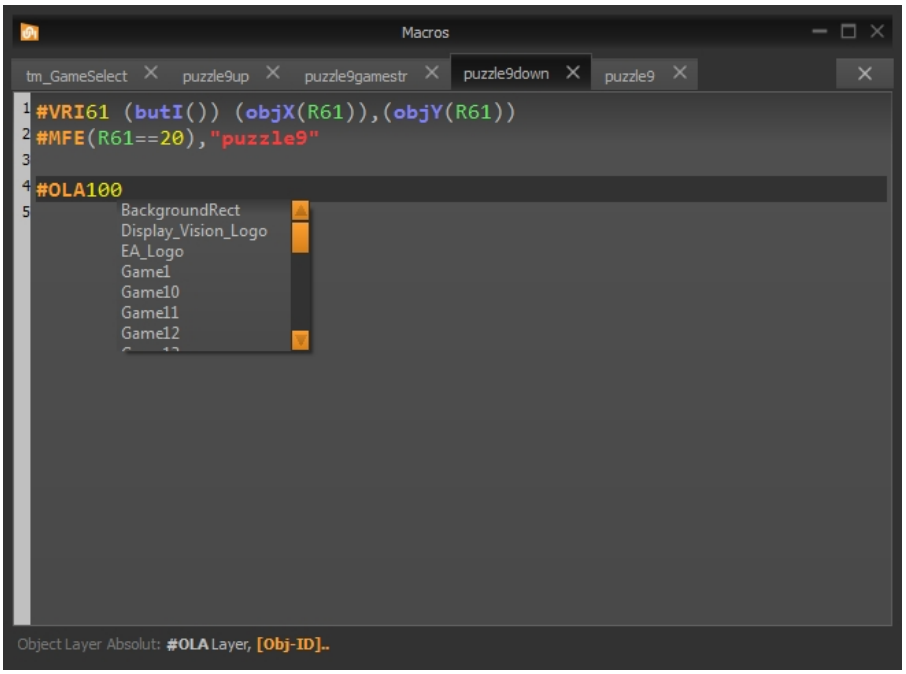
Register Editor

Im Register Editor (**View -> Workspace Panels -> Register and name definitions**) können Register und Stringregister Beautified names zugewiesen werden. Auch Default-Werte können hier eingestellt werden. Zusätzlich gibt es die Möglichkeit numerische und String – Defines zu vergeben, also eine Art Defines zu erstellen. Im Makro Editor können die definierten Beautified names anstelle der IDs verwendet werden.



Makro Editor

Im Makro Editor (**View -> Workspace Panels -> Macros**) werden Befehlsfolgen, als Funktionsgruppen den sogenannten Makros, geschrieben. Es bietet sich an hier alle nicht-grafischen Befehle oder zur Laufzeit zu berechnenden Objekte zu editieren und zu definieren.
Vorteilhaft ist das Syntaxhighlighting um Befehle und Parameter klar strukturiert zu erkennen. Auch Kommentare (beginnend mit **/**) können eingefügt werden.
Alle im Projekt verfügbaren Objekt-, Makro-, Registernamen und auch die eingebauten Kalkulationen werden passend zum Parameter vorgeschlagen (**Ctrl + Leertaste**).
Als Hilfe eignet sich die kurze Befehlshilfe in der Statuszeile. Mit dem Tastenkürzel **F1** wird automatisch die Hilfe zu dem jeweiligen Befehl angezeigt.

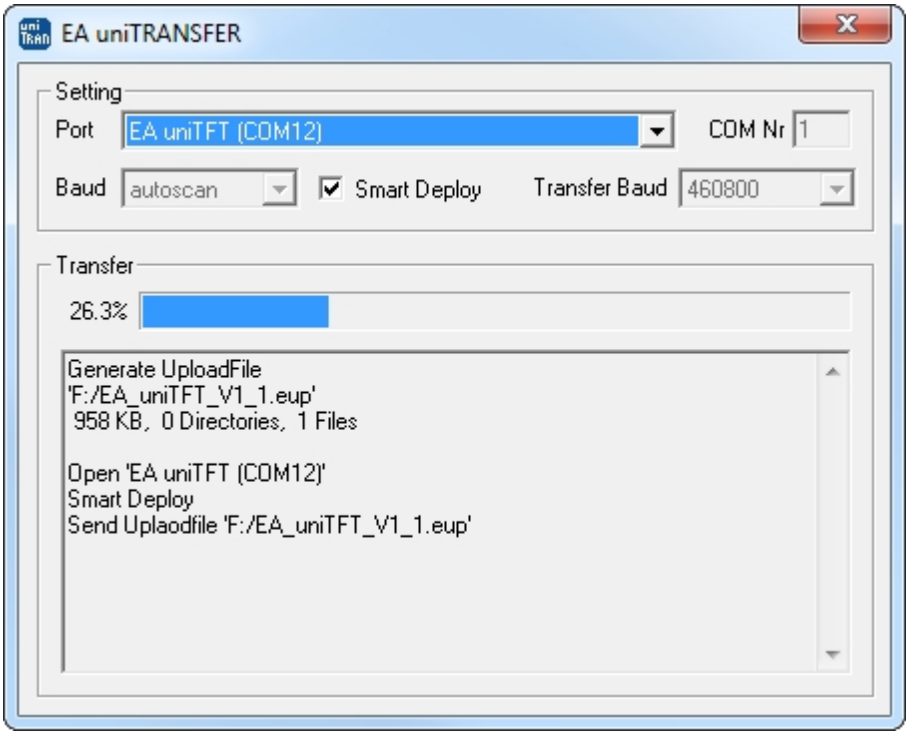


TOOLS FÜR WINDOWS

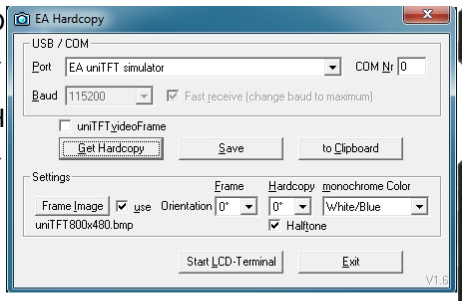
Neben der Designsoftware uniSKETCH sind eine Reihe weitere Windows-Tools vorhanden. Darunter das Tool EA uniTRANSFER welches Projekte und Files auf das EA uniTFTs-Serie übertragen kann. Zu Dokumentationszwecken ist es sehr Hilfreich Bildschirminhalte aus der jeweiligen Situation anzufertigen. Hier kann das Tool EA Hardcopy hilfreiche Dienste anbieten. Das mächtisgste Tool ist der EA uniTFT simulator, welcher die echte Hardware auf dem PC Simuliert.

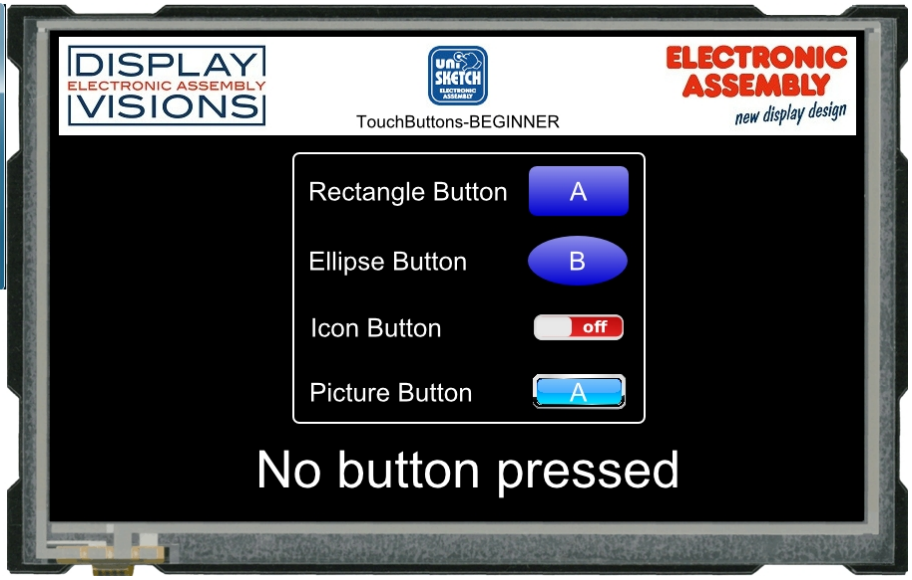
EA uniTRANSFER

Nach dem die Port-Einstellungen korrekt eingegeben sind kann uniTRANSFER beliebige Files auf das displayinterne FLASH kopieren. Um Projekte aufzuspielen ist es ausreichend, den Projektordner per Drag'n'Drop auf das Fenster zu ziehen. Ein Fortschrittsbalken im Programm gibt Auskunft über den Stand der Übertragung. Auf dem Display selbst sind weiter gehende Informationen Sichtbar.
Die Check-box "Smart Deploy" kann aktiviert werden um Dateien und Projekte möglichst schnell zu Übertragen. Er vergleicht Erstellzeit und Dateigröße zwischen Modul und Datenquelle. Sollten diese unterschiedlich sein wird die Datei ersetzt ansonsten bleibt sie bestehen und wird nicht kopiert. Das spart bei großen Dateien, wie Fonts oder Bildern sehr viel Zeit.
EA uniTRANSFER legt ein *.eup-File an. Dieses File beinhaltet alle Übertragungsdaten sowie Befehle zur Programmierung des FLASH Speichers. Sie können das erstellte Uploadfile *.eup auch unter einem beliebigen anderen System zum EA uniTFT übertragen. Dazu übertragen Sie den Inhalt der *.eup Datei 1:1 (mit Protokoll in Paketen), weitere Kommandos sind nicht notwendig.



EA Hardcopy



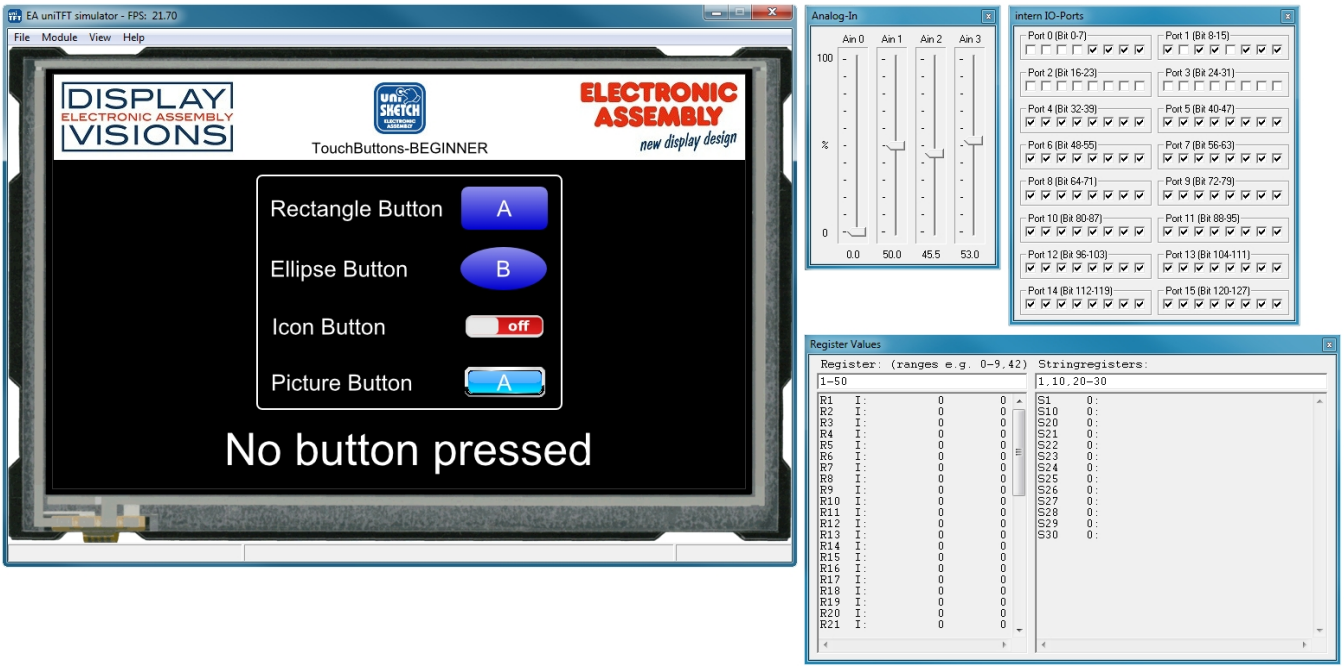


Das Hardcopy-Tool eignet sich hume in ne Ausssage kräftige Dokument

iondererAnwendungzuerstellen.

EA uniTFT Simulator

DerSimulatorkanndirektausdem



uniTFTDesigneraufgerufenwerdenundbildetdasProjektHardware-

nahab. Neben Eingabemöglichkeiten wie Ports und Analogeingängen k

anz. B. die Computereigene RS232 Schnittstelle ist Master RS232 od

er
RS232
Schnittstelle verwendet werden. Eine Debugfunktion und On

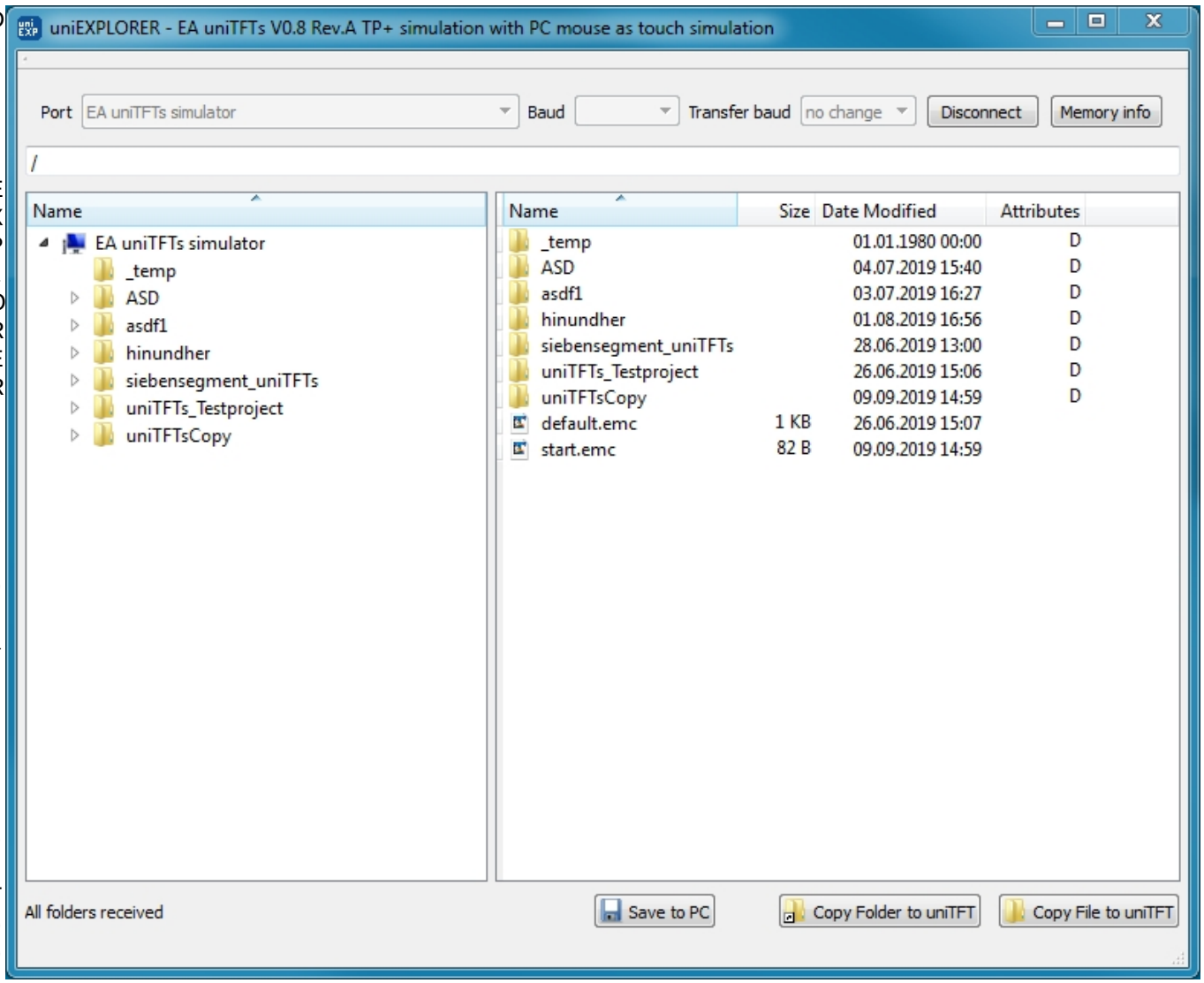
lineanzeige der Register erleichtert die Entwicklung eigener Makr

ofiles.Ebensoistesmöglichmit Breakpointszuarbeiten,oderauch

einzelne Zeile nach unten ausführen

EA uniEXPLORER

Der uniEXPLORER ist ein praktisches Tool um die auf dem Modul vorh



anden enDateienausz ulesen.

REVISION HISTORY

EA uniTFTs-Serie Firmware

Date	Version	Info
	1.0	First release

uniTFTs-Simulator

Date	Version	Info
	1.0	First release

uniTFT Designer - Designsoftware

Date	Version	Info
	1.0	First release

Helpfile

Date	Version	Info
	1.0	First release