itronOS+ Software Specification

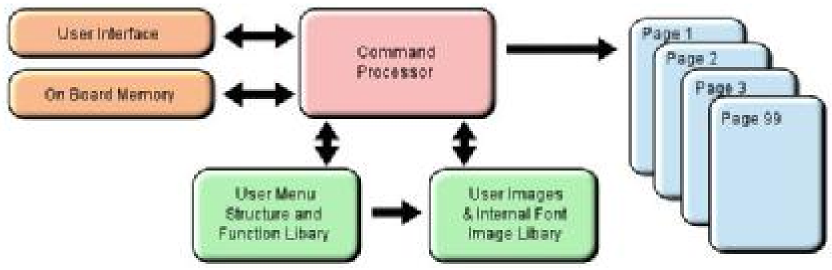
V00.49.55.A.2 14th March 2014

www.itronos.com

Contents

Software Overview

Software
Several customers have asked why we developed our own object oriented programming language rather than provide a product with Linux or an operating system supporting compiled 'C'. If we look back at the original requirements we can see some of the reasons.


User Interface
On Board Memory
Command Processor
Page 1
Page 2
Page 3
Page 99
User Menu Structure and Function Libary
User Images & Internal Font Image Libary

Prime: A combined operating and communication software offering unique capabilities for slave / host applications.
1/ The customer's end user or distributor could write code and insert images to add in their own functionality with a text editor.
2/ The program code could be updated or expanded by the host system using ASCII text over a serial link.
3/ The product should be license free and use simple development tools.
4/ The customer can create his own large images and control them like fonts.
5/ The SD card should be able to stream video and audio with the minimum of user programming.
6/ An existing host software requires only limited changes to upgrade a display from 4X20 LCD to a full colour TFT.
7/ The module has the intelligence to operate as a host and the compact command language to act as a high speed slave.
8/ The number of commands should be minimized by using 'overloading' and provide a higher level of functionality than C functions.
9/ The parameters for interfaces and screen entities should be held in styles similar to HTML.
10/ The application development time should take days or weeks rather than months.
11/ If the software engineer leaves the company, it is relatively easy for the engineering manager to amend the program.

These reasons may not be key to your application, but we believe it does offer new product opportunities.

High Level Object Oriented Commands
The module has an integrated compiler and debugger so that users can write the high level object oriented language commands in a text file or send via an interface to develop their application. Although pictures and fonts can be loaded via an interface, it is best to store these on an SD card or transfer via USB from on a PC. The multi faceted commands are divided into 4 groups as shown below.
You may be thinking how can 25 commands operate a host system, so lets take a look at the LOAD command. It can perform the equivalent language functions of strcpy, strcat, format, inp, outp and a page collation function. Please study our application example code for an understanding of this compact language.

library & system	page & visibility	draw on page	functions
FPROG:Load Menu/Img to Flash	PAGE: Create a page of entities	POSN: Position cursor on page	FUNC: Create a function
LIB: Load Image/Font to RAM	STYLE: Set parameters	TEXT: draw text on page	VAR: Create a variable
INC:Include a sub file	SHOW: Show a page or entity	DRAW: draw box circle line graph	IF ? : Conditional test-true/false
RUN:Call function or user code	HIDE: Hide a page or entity	IMG: draw image on page	LOOP: Repeat commands
RESET:Reset system, NAND	DEL: Delete entity from Library	KEY:create touch or external key	CALC: Calculation and string edit
;; Refresh current page	LOAD: Copy and format pages, strings, interface and data		WAIT: Set delay period
; Terminate command			INT: Set an interrupt

Styles make your Application Consistent
All entities and buffers use parameters stored in a Style similar to HTML web pages. These are extensive and define colours, entity types, buffer size and interface parameters like baud rate, clock edges and data format. Styles can be embedded in parent styles to reduce repetition and simplify changes.

Screen Page Creation and Control
Pages can be smaller than the screen for pop up help menus, status information and lists. Buttons can be varying size, with radio, rectangle or check box style with special types for navigation actions. The cursor position command allows relative or absolute positioning for reduced instructions during page layout. Entities can be updated by incoming host commands and their associated functions can run all the time or only when the entity or it's page is visible. When a text is numeric, it can be compared, incremented or decremented or form part of an equation using the CALC command. Buffers or variables can be created for interfaces, on-board memory, the SD Card, timers, counters and text. Hex code can be included in text variables when prefixed by \\.
When creating your page structures and functions in a file, // prefixes user comments.

Uploading your Menu Structure, Functions and Images
Data received from interfaces or flash memory is processed and stored in RAM libraries for high speed access to create or refresh pages and entities. Every entity has a text name for easy reference by future update commands.
In a similar way to a PC, your software could be permanently retained on an SD card and auto loaded at Power On or saved in internal flash by transferring it from an SD card or uploading it via an interface port. SD cards of 1G size and SDHC cards of 4G, 8G, 16G and 32G size are supported. 2G SD cards are not supported.

If an SD Card is used, the module will look for a file called 'TU480A.MNU' which will reference all other menu or image files. This may be your only menu file with all functions included. It would have a header similar to the example below to copy other files on the SD card to the internal flash memory. See the 'example projects' section

```
RESET(LIBRARY); FPROG;
LIB(BACKIMAGE,"SDHC/backmain.bmp");          //load background picture into the onboard flash library
LIB(STARTIMAGE,"SDHC/startbut.bmp");         //load start button into the onboard flash library
........ FEND;
```

Entities can be changed via the user interfaces by direct reference to there name or style with version v44 firmware.
Examples:

```
LOAD(homestyle.back,BLUE");              change the background colour of the page called homepage to blue
LOAD(rs2.set,"96e");                     change the rs232 baud rate to 9600 baud with even parity
LOAD(GenText.font,"40X56Kata");            change font size of all text using style GenText
```

Command Overview

Command Overview
This page identifies the current and expected operating status of commands and styles

	Command Format, with Parameters / Description
INT	INT(Name,Buffer,Function) If interrupt triggered do function
RESET	RESET(Name) Clear eeprom, delete list, library, or reset system
System	SETUP(System) { startup-bled-wdog-rotate-test-angles-encode-calibrate-clkfreq-ignore tch_loc_tlx-tch_loc_tly-tch_val_tlx-tch_val_tly-tch_loc_trx-tch_loc_try-tch_val_trx-tch_val_try tch_loc_brx-tch_loc_bry-tch_val_brx-tch_val_bry-tch_loc_blx-tch_loc_bly-tch_val_blx-tch_val_bly touchSave }
Touch - Res	SETUP(Touch) { type-enable-samples-debounce-accuracy-inactive-width-height-xnum-ynum-gain-threshold-address-filterlevel-filterlimit-filterenable-switch-invertx-inverty }
Touch - PCT	SETUP(Touch) { type-debounce-enable-inactive-width-height-xnum-ynum-gain-threshold-address-filterlevel-filterlimit-filterenable-switch-invertx-inverty }

Touch - ICT	SETUP(Touch) { type-enable-inactive }
RS2	SETUP(RS2) { baud-data-stop-parity-rxi-proc-procdel-procnum-rxb-txi-txb-encode-flow }
RS4	SETUP(RS4) { baud-data-stop-parity-rxi-proc-procdel-procnum-rxb-txi-txb-encode-flow-duplex }
ASYNC	SETUP(Async) { baud-data-stop-parity-rxi-proc-procdel-procnum-rxb-txi-txb-encode-flow }
SPI	SETUP(SPI) { active-mode-speed-rxi-proc-procdel-procnum-encode-rxb-rxo-rxf-txi-end-dummy-txb-txo-irq }
I2C	SETUP(I2C) { addr-end-active-speed-rxi-proc-procdel-procnum-encode-rxb-txi-txb }
USB	SETUP(USB) { rxi-txi-rxb }
KEYIO	SETUP(KEYIO) { active-inp-trig-edge-keyb-pullup }
ENC	SETUP(ENC) { active-a1-b1-a2-b2-debounce1-debounce2-timeout1-timeout2-mode1-mode2 }
ADC	SETUP(ADC) { active-calib1-calib2-avg1-avg2 }
PWM	SETUP(PWM) { active-pol1-pol2-pol3-cycle1-cycle2-cycle3-duty1-duty2-duty3-delay }
AC97	SETUP(AC97) { active-line_gain_L-line_gain_R-mic_gain_L-mic_gain_R-rec_sel-rec_rate-treble-bass-threeD- master_volume-speaker_volume-headphone_volume-balance-stereo_speaker-pbdonefnc-pbprogfnc-pbmode }
LIB	LIB(Name,Source) Load picture,audio or font into library. BMP/JPG/WAV/FNT
FPROG...FEND	FPROG.....FEND Store menu and image files in onboard flash
INC	INC(FileName) Include the contents of another menu, style or setup file
Page Style	STYLE(Name,Page) { update-sizeX-sizeY-posX-posY-back-image-type }
Text Style	STYLE(Name,Text) { font-size-col-back-opacity-maxLen-maxRows-rotate-justify-yAlign-xtrim-curRel-width-bcol-padding-xSpace-ySpace}
Draw style	STYLE(Name,Draw) { type-maxX-maxY-col-back-opacity-width-rotate-curRel-xOrigin-yOrigin-xScale-yScale-xScroll }
Img Style	STYLE(Name,IMG) { scale-maxX-maxY-rotate-action-step-opacity-curRel }
Key Style	STYLE(Name,Key) { type-debounce-delay-repeat-action-curRel-repeatnum-repeatdec-repeatend }
Del	DEL(Name) Delete a page, entity
Page	PAGE(NAME,STYLE){...}
POSN	POSN(X,Y,Page/Name,Style) Position cursor or re-position named entity
TEXT	TEXT(Name,Value,Style)
	TEXT(Name,Value,Style,PosX,PosY)
DRAW	DRAW(Name,SizeX,SizeY,Style)
	DRAW(Name,SizeX,SizeY,Style,PosX,PosY)
IMG	IMG(Name,Src,Style)
	IMG(Name,Src,Style,PosX,PosY)
KEY	KEY(Name,Func,SizeX,SizeY,Style);
	KEY(Name,downFunc,upFunc,repFunc,SizeX,SizeY,Style);
	KEY(Name,Func,SizeX,SizeY,Style,PosX,PosY);
LOOP	LOOP(Name,Var){...} Loop for a specified number of times
SHOW	SHOW(Name) Show a page, entity
HIDE	HIDE(Name) Hide a page, entity
REFRESH	;; Refresh current page
LOAD	LOAD(Name,N2,N3,N..) Multi function copy pages, variable N2--N.. to Name.
VAR	VAR(Name,Value,Style) Create a variable of a specified type with a default value
Array	VAR(Name,Value,Style,Num) Create an array of variables with size num
File	FILE(
CALC	CALC(Result, Var1, Var2, Act) Quick calculation and text manipulation
RUN	RUN(Func) Run a function or user code
FUNC	FUNC(Name) {...} Declare a set of commands
IF	IF(Var~Var?Func1:Func2) Evaluate condition and do func1 if true, func2 if false
SELECT	SELECT(var) { CASE(n,func);} Evaluate a variable and undertake function
EXIT	EXIT(Name) Use to exit current loop or function
RTC	RTC/D RTCSECS-RTCMINS-RTCHOURS-RTCDAYS-RTCMONTHS-RTCYEARS
RTA	RTA RTASECS-RTAMINS-RTAHOURS-RTADAYS-RTAMONTHS
Runtime Counter	Run Counters CNTMILLI-CNTSECS-CNTMINS-CNTHOURS-CNTDAYS-CNTRUN
IO Counter	IO Counters CNTK00 -- CNTK30
WAIT	WAIT(Time) Wait specified milliseconds before next
LOOP	LOOP(Name,Var){...} Loop for a specified number of times
STRUCTURE	STRUCT(Name,Num) { } Create a multi-dimensional structure of arrays or variables
TERMINATION	; Terminate command
INLINE	[cmd();cmd();....cmd;] Enclose commands as inline function in IF, INT, KEY, RUN

Setup

Command **INT**

Description
If an interrupt occurs for the specified buffer, it will run a function.
An interrupt will occur when a buffer’s style parameters allow activity within the buffer and the appropriate type of interrupt is set.
Serial interfaces can trigger on a byte received, a byte transmitted and a
semi-colon (command separator) received. I/O can trigger on input change.
Use HIDE(Name); to disable an interrupt.

Interrupts are available for counters and timers CNTMILLI...TIMER0. See relative section.

Syntax/Parameters
INT(Name,Buffer,Function,*Priority*);

Name Of Interrupt

Buffer to Interrupt on
> RS2RXC = RS232 Receive Character
> RS4RXC = RS485 Receive Character
> AS1RXC = Async1 Receive Character
> AS2RXC = Async2 Receive Character
> SPIRXC = SPI Recieve Character
> I2CRXC = I2C Receive Character
> USBRXC = USB Recieve Character
> DBGRXC = Debug Receive Character

> TOUCHI = Touch Inactivity Timeout

> ENC = Rotary Encoder 1
> ENC1 = Rotary Encoder 1
> ENC2 = Rotary Encoder 2

> K0 = IO Port K0 or K00
> ....
> K30 = IO Port K30

> TIMER0 = Timer0
> ....
> TIMER19 = Timer19

> CNTMILLI = CNTMILLI timer
> CNTSECS = CNTSECS timer
> CNTMINS = CNTMINS timer
> CNTHOURS = CNTHOURS timer
> CNTDAYS = CNTDAYS timer

Function to call when Buffer read

Priority - Optional

Options
Interrupts can be assigned a priority for faster processing by providing an optional parameter in the INT() command:
INT(name,source,func,priority);
* The default priority will be 0 which will run INT()'s as implemented until now (ie more like pending tasks).
* A value greater than 0 will process the INT() as soon as the interrupt condition has occurred. Interrupts with larger values will have
  a higher priority and can interrupt those with a lower value.
* The range of values are 0 to 15.
* Only one interrupt is permitted per priority level.
* As these interrupts are being processed as true interrupts some commands are not supported. The non-supported commands will
  return an error. These are:
  SHOW(page); // though ;; and SHOW(THIS_PAGE); are supported, see below.
  EXIT();
  FPROG;
  FEND;
* Page refreshing of the currently viewed page is allowed with ";;" or SHOW(THIS_PAGE); BUT changing to another page is not allowed
  from an INT() with priority greater than 0 as this could cause undesired effects. Care should be taken with refreshing a page from an
  INT() as this will lock out other interrupts during this time.
* Interrupt routines should be kept as short as possible to avoid locking out other interrupts and the main processing.

NOTE: <u>The Buffer must be read to clear the interrupt otherwise the Function will keep getting called!</u>

HIDE/SHOW(INT) - v49.03
Pending interrupts are now processed when an interrupt entity is unhidden.

Priority Interrupts - Modified Priority Scheme - v49.37
Modified functionality to always process higher priority interrupts pending rather than process multiple interrupts pending at current level.

Losing Serial Interrupts with "proc" - v49.37
A new INT() processing scheme has been written which has abandoned a "counter" method and instead checks to see if there are any further "packets" waiting to be processed when the INT() command is run.
The old scheme made use of a counter which was incremented in the "hardware" interrupt handler when a packet was received and then decremented when a LOAD(buf,port); was performed from the INT() function. It had been found that the counter can get out of sync with the packets being received and hence packets are left in the receive buffer when the counter has a value of zero.
The new method, sets a task flag in the "hardware" interrupt handler to say a packet has been received. The INT() function then reads the packet when the LOAD(buf,port); command is used and then checks to see if there is another complete packet in the receive buffer. If there is, then the INT() function is called again, and so on, until there are no more complete packets and then the INT() is exited and normal processing resumes.

Nesting of priority INT()s - v49.44
New functionality has been added to support nesting of priority INT()s, ie a priority interrupt can be interrupted by another priority
interrupt with a higher priority (this is now the default behaviour).

A system setup variable has been added to disable this functionality.
SETUP(SYSTEM){ intNest = y | n; } // default = y;

For 'y', priority INT()s can be interrupted by higher priority INT()s
For 'n', priority INT()s run to completion, then the highest pending priority INT() is processed next.

Real Time (Priority) Interrupts - v 49.44
This issue has been resolved. A problem was found with the operating systems' nested interrupt handler. New functionality has been implemented and tested.
Nesting of priority INT()s has also been added - see TFT Improvement

Example
```
PAGE( PageName, PageStyle)
 {
  INT( SerRxInt, RS2RXC, SerRxEvent );
 }
 FUNC( SerRxEvent )
 {
 LOAD( Var, RS2 ); // Must read RS2 to clear interrupt
 LOAD( RS4, Var);  //send out of RS485 interface.
 TEXT ( RecvTxt, Var);; //show received ASCII data on screen and Refresh
 }
```
SHOW/HIDE(INT) Example
```
INT( intRs2, RS2RXC, fncRs2Rx ); // Create RS2 receive interrupt
```

```
HIDE( intRs2 ); // Hide the interrupt
SHOW( intRs2 ); // Show (Enable) interrupt and process pending interrupts
//If you don't want to process any pending interrupts when you show the interrupt then RESET the interrupt first
INT( intTmr0, TIMER0, fncTmr0 ); // Create RS2 receive interrupt
HIDE( intTmr0 ); // Hide the interrupt
RESET( intTmr0 ); // Clear any pending interrupts
SHOW( intTmr0 ); // Show (Enable) interrupt for future interrupts
```

Command	**Reset**
Description	**Clear the contents of the RunTime Counter, Delete List, Library Files or do a System reset.** **System - Reset the System so that it re-boots as at power ON.** **Note: When a RESET(SYSTEM); or hardware reset occurs, the boot software in the module looks to see if a valid start up file type TUxxxA.mnu is present on the SD card, if not, it looks in the internal NAND flash memory. If no TUxxx.mnu file is found, the module initializes the interface RS2 in command mode as 115200,8,1,N and AS1 as 500000,8,N,1,H.** **Runtime - Clears the runtime counter.** **EEPROM - Clears the EEPROM and reload defined variables.** **DELETED - Clears the deleted entity list.** **NAND - Clears the NAND flash memory, also clears both MNU and LIB areas of NAND.** **NANDMNU - Clears just the MNU files in NAND flash.** **NANDLIB -Clears the BMP, FNT, WAV files in NAND flash (LIB area of NAND).** **LIBRARY - Clears the library. Allows new program to load. Interface setup unchanged.** **SDHC - Reinitialises SD card handler (useful after SD card removal/reinsertion).** **FACTORY -Erases all user data from NAND and the whole EEPROM.** **Default values for touch screen are then used. Touch panel calibration will need to be run.** **START - Restarts system without hardware reset and keeping USB/RS232 alive.** **A TUxxxA.MNU file will expected on the SD Card or in NAND.** **Note that RESET(LIBRARY) does the same but doesn't look for a TUxxxA.** **MNU file as the module expects commands to be sent serially to it in this case.** **USB - Added RESET(USB) which causes USB interface to disconnect, reinitialise and reconnect.** **Setting rxi=n and txi=n now disconnects USB interface, then interface is reinitialised when either rxi or txi is enabled.** **RESET(LIBRARY) and RESET(START); Improvements - v49.46** *** This function was intended for use as a command from iDevTFT. However, some customers wish to use the functionality from within their own project code. Therefore, functionality has been modified to process the reset only once the parser has returned to the top level of the scheduler.**
Syntax/Parameters	**RESET(Name);** **Name of item to Reset**

Example	**RESET(SYSTEM);**
	RESET(RUNTIME);
	RESET(EEPROM);
	RESET(DELETED);
	RESET(NAND);
	RESET(NANDMNU);
	RESET(NANDLIB);
	RESET(LIBRARY);
	RESET(SDHC);
	RESET(FACTORY);
	RESET(START);
	RESET(USB);

Command	**SETUP(System)**
Description	**Set up the system. These parameters can be set at initialisation or at any time during operation by specifying the parameter to be changed. Example: SETUP(SYSTEM){ bled=50; }. To change a setting use a dot operator as follows: LOAD(SYSTEM.bled, 50);**

Syntax/Parameters	**Setup(System)
	{
	...
	}**

Options

startup = all; display messages and progress bar at start up using startup=all or none or bar.
bled = 100; set backlight to OFF=0 or ON=100 (1-99 brightness levels available v4 PCB, v32 firmware)
wdog = 1000; set the watchdog time out period in milliseconds.
rotate = 0; set the rotation of the screen with respect to PCB. This is stored in EEPROM for use with boot messages.(0, 90, 180 or 270)
test = hide/showTouchAreas; hide or show touch areas during product development
angles = degrees; select degrees or radians for calc functions
encode = s, w, m; menu text strings can contain single byte ASCII (s), 2 bytes for UNIcode (w) or multibyte for UTF8 (m)
calibrate = y; initialise the internal touch screen calibration screen. This automatically returns to the previous page on completion. If it is necessary to abort then send SETUP(SYSTEM) {calibrate=n;}
clkfreq = 92000000; Main external bus clock is changeable in 2MHz steps from 80MHz to 92MHz (default).
ignore = allErrors; Ignores all errors and continues execution (only recommended in test as can cause undesired results - error messages are still sent from a serial port set to debug, ie rxi=d;).
= invalidJpg; Ignores errors for unsupported JPG formats (eg progressive) and the image is skipped
= imageTooBig; Ignores errors when there's not enough memory to load image and the image is skipped

System Version
The software and hardware versions can be read to a serial port or text variable.

```
   LOAD( RS2, VERS_IBOOT ); returns NAND bootloader version
   LOAD( RS2, VERS_ILOADER ); returns main loader version
   LOAD( RS2, VERS_IAPP ); returns main application version
   LOAD( RS2, VERS_IMODULE ); returns module name and version
```

Startup Information - v42.00
Displaying of messages and progress at start up now configurable using SETUP(SYSTEM){startup=all|none|bar;}
Word 'DEMO' changed to 'PROTO' in version message as this software is still not formally released.

Touch Calibration - v47.24
Create your own calibration screen and store parameters in host.

Touch Calibration points can now be set in system setup with user parameters.

```
> To use:
   SETUP(system)
   {
   tch_loc_tlx = num; // Top Left Touch Point Location X
   tch_loc_tly = num; // Top Left Touch Point Location Y
   tch_val_tlx = num; // Top Left Touch Value X
   tch_val_tly = num; // Top Left Touch Value Y
   tch_loc_trx = num; // Top Right Touch Point Location X
   tch_loc_try = num; // Top Right Touch Point Location Y
   tch_val_trx = num; // Top Right Touch Value X
   tch_val_try = num; // Top Right Touch Value Y
   tch_loc_brx = num; // Bot Right Touch Point Location X
   tch_loc_bry = num; // Bot Right Touch Point Location Y
   tch_val_brx = num; // Bot Right Touch Value X
   tch_val_bry = num; // Bot Right Touch Value Y
   tch_loc_blx = num; // Bot Left Touch Point Location X
   tch_loc_bly = num; // Bot Left Touch Point Location Y
   tch_val_blx = num; // Bot Left Touch Value X
   tch_val_bly = num; // Bot Left Touch Value Y
   }
> also supports load(system.tch_loc_tlx,num); etc..
```

Touch Calibration points can also be read
> load(num,tch_loc_tlx); etc.. <- note "system." not used here

The actual touch values are obtained by using TOUCHX and TOUCHY built in variables.

Error Message Suppression - v49.00
Error messages can be ignored/suppressed for certain situations.

```
    SETUP( system )
   {
   ignore=imageTooBig; // Ignores errors when there's not enough memory to load image and the image is skipped
   ignore=invalidJpg; // Ignores errors for unsupported JPG formats (eg progressive) and the image is skipped
   ignore=allErrors; // Ignores all errors and continues execution (only recommended in test as can cause undesired results).
   //No messages are displayed on the screen, though they are transmitted on a serial port set to debug.
   }
```

Memory Usage Output - v49.00
A run-time memory allocation can be output of the diagnostic serial port by setting SETUP(system){test=showMemoryUse;}.

Note, that the transmit buffer should be at least 32KB otherwise transmit data may be lost.
The top location of RAM is 0x23FFFFFF.

Test Flags - v49.04
SETUP(SYSTEM){test=showLoadInfo;} displays loading file / line information on start screen

System Errors - v49.19
All system errors now allocated a fixed error number, which is displayed on the System Error Screen.
Details of the last error can be found from the built-in entities:

```
   SYSERRNUM (S32) Error number
   SYSERRSRC (TXT64) Source of error (menu file name or port name)
   SYSERRPOS (S32) Error line number in source file (if applicable)
   SYSERRMSG (TXT128) Error message for given error number
```

The variables maintain the last error information.

An error function can be specified which is called when a system error occurs:

```
   SETUP(SYSTEM) { errFunc=funcName; }
```

The system error will appear on the screen then the function will be called. The error screen is then cleared by tapping the touch screen. The error function should be kept simple, as functionality is restricted, such as enabling USB port to allow updates.

If SETUP(SYSTEM) { ignore=allErrors; } is set then just the error function is called, no error screen is displayed and no touch is required.
All errors are always sent to a port set to debug (rxi=d;) independent of the ignore or errFunc settings.

Error information can be found on the website click here

Screen Rotation - v49.25
Screen rotations of 0, 90, 180 and 270 degrees fully supported.

Use, for example, SETUP(SYSTEM){rotate=90;} at start of tuXXXa.mnu file. The value is stored in EEPROM and used to orientate boot messages etc on next boot.
Also boot startup messages centralised and rotated.
nandboot.bin V00.27
iuloader.bin V00.38

Touch Screen Calibration - Manual Save & Status - v49.37
* Added option to save manual touch calibration figures to EEPROM, either
```
SETUP(SYSTEM)
{
tch_val_tlx = 123;
...
tch_val_bly = 345;
touchSave = eeprom; // or "none" for no storage
}
```
or,
LOAD(SYSTEM.touchSave,eeprom);
* Added system parameter calFunc which is called when calibration is started and finished. Reading TCH_CAL value will indicate what stage the calibration is at.
```
SETUP(SYSTEM){calFunc=myFunc;}
FUNC(myFunc){LOAD(RS2,TCH_CAL);}
LOAD(SYSTEM.calibrate,y);
```
> myFunc will run when calibration page is shown - TCH_CAL value = 255 (calibrating)
> myFunc will run when calibration finished and previous page shown - TCH_CAL value = 1 (done)
> TCH_CAL value = 0 when touch is not calibrated.
* Added additional retry mechanisms for saving/retrieving values from EEPROM.

Storage of cal figures, screen rotate, and start up text in file: tu480a.cfg - v49.47
* To use the NAND config file for settings, add useCfg=y; to system setup,

```
SETUP(SYSTEM)
{
useCfg=y;
}
```

* File is named tuXXXa.cfg, where XXX is module size 320, 480, 640, 800. The file is stored in NAND in the "LIB" area.
* Typical Config File contents (note parameter values are all numeric within the file):

```
usecfg=1; // Use config file: 0=no, 1=yes
sbled=100; // Start-up back light level: 0 to 100
wdog=-1; // Watchdog: -1 unset, 0 off, 1-16000 time ms
rotate=0; // Screen rotate: 0, 90, 180, 270
startup=3; // Start-up: 0=none, 1=text, 2=bar, 3=text+bar
tch_loc_tlx=19; // Touch - TL x location
tch_loc_tly=19; // Touch - TL y location
tch_loc_trx=459; // Touch - TR x location
tch_loc_try=19; // Touch - TR y location
tch_loc_brx=459; // Touch - BR x location
tch_loc_bry=251; // Touch - BR y location
tch_loc_blx=19; // Touch - BL x location
tch_loc_bly=251; // Touch - BL y location
tch_val_tlx=87; // Touch - TL x value
tch_val_tly=912; // Touch - TL y value
tch_val_trx=928; // Touch - TR x value
tch_val_try=905; // Touch - TR y value
tch_val_brx=912; // Touch - BR x value
tch_val_bry=131; // Touch - BR y value
tch_val_blx=84; // Touch - BL x value
tch_val_bly=132; // Touch - BL y value
```

* Changes made by SETUP(SYSTEM){} and LOAD(SYSTEM.param,val); will modify the file.
* Following touch screen calibration, the calibration parameters will be stored to the config file.
* The new file is compared with the file stored in NAND and only overwritten if different to save NAND write cycles.
* When a RESET(NAND); or RESET(NANDLIB); is issued, the config file is preserved by reading file into memory, erasing NAND, writing the config file back to NAND.
* A RESET(FACTORY); will erase the config file.
* Parameters are read from EEPROM initially and then overwritten by parameters from the config file.
* If an EEPROM is not fitted, then useCfg will default to y.

Start-up backlight level "sbled" - v49.47
* A start-up backlight can be specified.

```
SETUP(SYSTEM)
{
sbled = 100; // Values 0 to 100
}
```

* This value is used as the default level rather than the current 100% value.
* Value is also stored in EEPROM and used for boot-up screens.

New boot code created:
nandboot.bin V00.31
iuloader.bin V00.43

Example
```
SETUP( SYSTEM )
 {
   bled=100;
   wdog=100;
   rotate=0;
   calibrate=n;
   test=showTouchAreas;
   angles=degrees;
   startup=all;
   encode=s;
   clkfreq=92000000;
 }
```

Command **SETUP(TOUCH)**

Description **Setup The Touch settings for Resistive, Projective Capacitive or Immediate Touch**

Projected Capacitive Touch - Suitable with glass (up to 4mm thick) and plastic (up to 1.5mm thick) overlay

Immediate Capacitive Touch - Direct Contact, Not suitable for glass or plastic overlay

Syntax/Parameters **setup(Touch){...}**

Options

Resistive					
Parameter	3.5	4.3	5.7	7.0	Notes
type	res	res	res	res	
enable	y	y	y	y	enable touch keys globally with y or n
samples	-	12	12	22	define the number of touch samples per interrupt.
debounce	-	25	30	25	define the time period between each sampling period.
accuracy	-	50	14	12	define the 0.25 pixel accuracy of the samples.
inactive	1000	1000	1000	1000	time in milliseconds (0=off)

To increase the sensitivity set samples = low number e.g. 3
This may only be suitable for buttons which are larger than 80x80 pixels

	3.5	4.3	5.7	7.0	
Code					simple program to help test touch setup

Projective Capacitive				
Parameter	4.3	5.7	7.0	Notes
type	cap	cap	cap	
width	272	480	480	default to height of display
height	480	640	800	default to width of display
xnum	16	16	16	number of X electrodes
ynum	14	14	14	number of Y electrodes
gain	0	0	0	gain of ADC
threshold	25	25	25	
debounce	3	3	3	
address	75	75	75	I2C address of controller
inactive	1000	1000	1000	time in milliseconds (0=off)
filterlevel	1	1	1	level of filtering applied 0-15
filterlimit	4	4	4	maximum filter limit 0-7
filterenable	Y	Y	Y	enable filtering Y/N
switch	Y	Y	Y	switch X and Y electrodes
invertx	Y	Y	Y	invert the order of the X electrodes
inverty	N	N	N	invert the order of the Y electrodes
enable	Y	Y	Y	enable touch keys globally with y or n

	4.3	5.7	7.0	
Code				simple program to help test touch setup

v49.37 Added ability to specify mxt224 object data using Txx as the parameter name :- ie T9 = "15,00,02,11";
Added separate setup code for all mxt224 variants (mxt224v1,mxt224v2,mxt224Ev1,mxt224Ev2)
Added RESET(TOUCH); for projective capacitive touch

Immediate/Contact Capacitive		
Parameter	7.0	Notes
type	ICT CCT	part number suffix of module
enable	Y	enable touch keys globally with y or n
inactive	1000	time in milliseconds (0=off)

Type= part number suffix of module ie TU800x480C-K612A1U-CCT would use type=cct;

Example

```
setup(touch) // resistive touch
{
type = res; // default is 'res' for resistive touch
enable = y; // enable touch keys globally with y or n
samples = 20; // define the number of touch samples per interrupt. Defaults:4.3" = 12; 5.7" = 12; 7" = 22;
debounce = 10; // define the time period between each sampling period. Defaults: 4.3" = 25; 5.7" = 30; 7" = 25;
accuracy = 20; // define the 0.25 pixel accuracy of the samples. Defaults: 4.3" = 50; 5.7" = 14; 7" = 12;
inactive=1000; // time in milliseconds (0=off)
}

setup(touch)  // projective capacitive touch
{
type = cap; // default is 'res' for resistive touch
enable = y; // enable touch keys globally with y or n
width = 272; // default to height of display
height = 480; // default to width of display
xnum = 16; // number of X electrodes
ynum = 14; // number of Y electrodes
gain = 0; // gain of ADC
threshold = 25;
debounce = 3;
address = 75; // I2C address of controller
inactive=1000; // time in milliseconds (0=off)
filterlevel=1;  //level of filtering applied 0-15,  default=1
filterlimit=4;   //maximum filter limit 0-7; default=4
filterenable=Y;  //enable filtering Y/N
switch=Y;          //switch X and Y electrodes
invertx=Y;        //invert the order of the X electrodes
inverty=N;        //invert the order of the Y electrodes
}

setup(touch) //immediate/contact capacitive touch
{
type = ict;  // other options are cct or cct2 depending on module part number suffix
enable = y; // enable touch keys globally with y or n
inactive=1000; // time in milliseconds (0=off)
}
```

Command **SETUP(RS2)**

Description **Setup the protocol for the RS232 communication**

Syntax/Parameters **setup(RS2){...}**

Options

```
  set="96NC";          //quick set up combination "48, 96, 192, 384, 768,  1150 with parity N, O, E and Command option".

  baud=38450;     //num = 110 to 6,250,000. Any value can be set to allow trimming for deviating clocks i.e. 38450
  data=6;            //num = 5, 6, 7, 8
  stop=15;          //num = 1, 15, 2  - note 15 is 1.5 bits
  parity=N;         //first letter of Odd, Even, None, Mark, Space
  rxi=Y;              //set receive interface as active (Y), a command processing source (C) or disable (N). Default = N
  proc=“;”;         //process on receive termination character. See below
  procDel=Y;       //remove or keep the termination character(s) before processing
  procNum=5;     //interrupt on n bytes received as alternative to proc and procDel.
  rxb=8246;        //set size of receive buffer in bytes. Default = 8192 bytes maximum = 256K bytes.
  txi=Y;              //set transmit interface as active (Y), to echo command processing (E) or disable (N)
  txb=8350;         //set size of transmit buffer in bytes. Default = 8192 bytes
  encode=s;         //set s=ASCII, w=UNICODE, m=UTF8 or use sr specifying raw text bytes and sd for raw data.
  flow=N;            //none (N), hardware RTS/CTS or DTR/DSR (H), software XON XOFF (S).
```

Example

```
setup(RS2)
{
  set="96NC";          //quick set up combination "48, 96, 192, 384, 768,  1150 with parity N, O, E and Command option".
}

setup(RS2)
{
baud=38450;     //num = 110 to 6,250,000. Any value can be set to allow trimming for deviating clocks i.e. 38450
data=6;            //num = 5, 6, 7, 8
stop=15;          //num = 1, 15, 2  - note 15 is 1.5 bits
parity=N;         //first letter of Odd, Even, None, Mark, Space
rxi=Y;              //set receive interface as active (Y), a command processing source (C) or disable (N). Default = N
proc=“;”;         //process on receive termination character. See below
procDel=Y;       //remove or keep the termination character(s) before processing
rxb=8246;        //set size of receive buffer in bytes. Default = 8192 bytes maximum = 256K bytes.
txi=Y;              //set transmit interface as active (Y), to echo command processing (E) or disable (N)
txb=8350;         //set size of transmit buffer in bytes. Default = 8192 bytes
encode=s;         //set s=ASCII, w=UNICODE, m=UTF8 or use sr specifying raw text bytes and sd for raw data.
flow=N;            //none (N), hardware RTS/CTS or DTR/DSR (H), software XON XOFF (S).
}
```

Command **SETUP(RS4)**

Description **Setup the protocol for the RS485 communication**

Syntax/Parameters **setup(RS4){...}**

Options

```
  set="96NC";            //quick set up combination "48, 96, 192, 384, 768,  1150 with parity N, O, E and Command option".

  baud=38450;       //num = 110 to 6,250,000. Any value can be set to allow trimming for deviating clocks i.e. 38450
  data=6;              //num = 5, 6, 7, 8
  stop=15;             //num = 1, 15, 2  - note 15 is 1.5 bits
  parity=N;           //first letter of Odd, Even, None, Mark, Space
  rxi=Y;                //set receive interface as active (Y), a command processing source (C) or disable (N). Default = N
  proc=“;”;            //process on receive termination character(s). See below
  procDel=Y;         //remove or keep the termination character(s) before processing
  procNum=5;     //interrupt on n bytes received as alternative to proc and procDel.
  rxb=8196;           //set size of receive buffer in bytes. Default = 8192 bytes, maximum 256K bytes.
  txi=Y;                //set transmit interface as active (Y), to echo command processing (E) or disable (N)
  txb=8196;           //set size of transmit buffer in bytes. Default = 8192 bytes
  encode=s;          //set s=ASCII, w=UNICODE, m=UTF8 or use sr specifying raw text bytes and sd for raw data.
  flow=N;                 //none (N), software XON XOFF (S).
  duplex=F;              //set Full Duplex (F) or Half Duplex (H).  Half duplex uses connector CN1 pins 1 and 8.
```

Example

```
setup(RS4)
{
set="96NC";            //quick set up combination "48, 96, 192, 384, 768,  1150 with parity N, O, E and Command option".
}

setup(RS4)
{
baud=38450;       //num = 110 to 6,250,000. Any value can be set to allow trimming for deviating clocks i.e. 38450
data=6;              //num = 5, 6, 7, 8
stop=15;             //num = 1, 15, 2  - note 15 is 1.5 bits
parity=N;           //first letter of Odd, Even, None, Mark, Space
rxi=Y;                //set receive interface as active (Y), a command processing source (C) or disable (N). Default = N
proc=“;”;            //process on receive termination character(s). See below
procDel=Y;         //remove or keep the termination character(s) before processing
rxb=8196;           //set size of receive buffer in bytes. Default = 8192 bytes, maximum 256K bytes.
txi=Y;                //set transmit interface as active (Y), to echo command processing (E) or disable (N)
txb=8196;           //set size of transmit buffer in bytes. Default = 8192 bytes
encode=s;          //set s=ASCII, w=UNICODE, m=UTF8 or use sr specifying raw text bytes and sd for raw data.
flow=N;                 //none (N), software XON XOFF (S).
duplex=F;              //set Full Duplex (F) or Half Duplex (H).  Half duplex uses connector CN1 pins 1 and 8.
}
```

Command **SETUP(AS1 / AS2 / DBG)**

Description **Setup the protocol for the Async1 / Async2 / Debug communication**

Syntax/Parameters **setup(AS1){...} / setup(AS2){...} / setup(DBG){...}**

Options

```
  set="96NC";              //quick set up combination "48, 96, 192, 384, 768,  1150 with parity N, O, E and Command option".

  baud=38450;      //num = 110 to 6,250,000. Any value can be set to allow trimming for deviating clocks i.e. 38450
  data=7;          //num = 5, 6, 7, 8
  stop=2;          //num = 1, 15, 2  - note 15 is 1.5 bits
  parity=N;        //first letter of Odd, Even, None, Mark, Space
  rxi=Y;           //set receive buffer interface as active (Y), a command processing source (C) or disable (N). Default = N
  proc=“;”;        //process on receive termination character(s). See below
  procDel=Y;       //remove or keep the termination character(s) before processing
  procNum=5;       //interrupt on n bytes received as alternative to proc and procDel.
  rxb=8246;        //set size of receive buffer in bytes. Default = 8192 bytes, maximum 256K bytes.
  txi=Y;           //set transmit buffer interface as active (Y), to echo command processing (E) or disable (N)
  txb=8246;        //set size of transmit buffer in bytes. Default = 8192 bytes
  encode=s;        //set s=ASCII, w=UNICODE, m=UTF8 or use sr specifying raw text bytes and sd for raw data.
  flow=N;          //none (N), hardware RTS/CTS or DTR/DSR (H), software XON/XOFF (S).
```

Example

```
setup(AS1)
{
  set="96NC";              //quick set up combination "48, 96, 192, 384, 768,  1150 with parity N, O, E and Command option".
}

setup(AS1)
{
baud=38450;      //num = 110 to 6,250,000. Any value can be set to allow trimming for deviating clocks i.e. 38450
data=7;          //num = 5, 6, 7, 8
stop=2;          //num = 1, 15, 2  - note 15 is 1.5 bits
parity=N;        //first letter of Odd, Even, None, Mark, Space
rxi=Y;           //set receive buffer interface as active (Y), a command processing source (C) or disable (N). Default = N
proc=“;”;        //process on receive termination character(s). See below
procDel=Y;       //remove or keep the termination character(s) before processing
rxb=8246;        //set size of receive buffer in bytes. Default = 8192 bytes, maximum 256K bytes.
txi=Y;           //set transmit buffer interface as active (Y), to echo command processing (E) or disable (N)
txb=8246;        //set size of transmit buffer in bytes. Default = 8192 bytes
encode=s;        //set s=ASCII, w=UNICODE, m=UTF8 or use sr specifying raw text bytes and sd for raw data.
flow=N;          //none (N), hardware RTS/CTS or DTR/DSR (H), software XON/XOFF (S).
}
```

Command **SETUP(SPI)**

Description **Setup the protocol for the SPI communication**

Syntax/Parameters **setup(spi){...}**

Options

```
  set="SHRC";              //quick set up as (M)aster/(S)lave, idle (L)ow/(H)igh, edge (R)ising/(F)alling, (C)ommand and speed 350-90000

  active=S;               //set as Master, Slave or None for both transmit and receive. Default = N
  mode=LR LF HF HR;    //set idle state Low or High and Rising or Falling clock edge. Default = LR
  speed=100;          //set transmit speed value in kilobits/sec from 350 to 90000 for master mode. Default = 1000
  rxi=Y;              //set receive buffer interface as active (Y), a command processing source (C) or disable (N). Default = N
  proc=“;”;           //process on receive termination character(s). See below.
  procDel=Y;         //remove or keep the termination character(s) before processing
  procNum=5;       //interrupt on n bytes received as alternative to proc and procDel.
  encode=s;        //set s=ASCII, w=UNICODE, m=UTF8 or use sr specifying raw text bytes and sd for raw data.
  rxb= 8264;       //set size of receive buffer in bytes. Default = 8192 bytes
  rxo=M;           //set receive data order as most significant bit (M) or least significant bit (L). Default = M
  rxf= N;             //use none or hardware MB to signify receive buffer full. Default = N
  txi=Y;              //set transmit buffer interface as active (Y), to echo command processing (E) or disable (N)
  end="\\nn";        //byte sent to host when no data left in display's spi transmit buffer.
  dummy="\\nn";  //dummy byte sent to module which is ignored so that data can be received by the host.
  txb=8244;        //set size of transmit buffer in bytes. Default = 8192 bytes
  txo=M;           //set transmit data order as most significant bit (M) or least significant bit (L). Default = M
  irq=L;           //use irq in slave mode. N=no, L=low; H=high; B=buffered using input/output switch. CN3 pin 7
```

Example

```
setup(spi)
{
set="SHRC";
}

setup(spi)
{
active=S;               //set as Master, Slave or None for both transmit and receive. Default = N
mode=LR LF HF HR;    //set idle state Low or High and Rising or Falling clock edge. Default = LR
speed=100;          //set transmit speed value in kilobits/sec from 350 to 90000 for master mode. Default = 1000
rxi=Y;              //set receive buffer interface as active (Y), a command processing source (C) or disable (N). Default = N 0
proc=“;”;           //process on receive termination character(s). See below.
procDel=Y;         //remove or keep the termination character(s) before processing
encode=s;        //set s=ASCII, w=UNICODE, m=UTF8 or use sr specifying raw text bytes and sd for raw data.
rxb= 8264;       //set size of receive buffer in bytes. Default = 8192 bytes
rxo=M;           //set receive data order as most significant bit (M) or least significant bit (L). Default = M
rxf= N;             //use none or hardware MB to signify receive buffer full. Default = N
txi=Y;              //set transmit buffer interface as active (Y), to echo command processing (E) or disable (N)
end="\\nn";        //byte sent to host when no data left in display's spi transmit buffer.
dummy="\\nn";  //dummy byte sent to module which is ignored so that data can be received by the host.
txb=8244;        //set size of transmit buffer in bytes. Default = 8192 bytes
txo=M;           //set transmit data order as most significant bit (M) or least significant bit (L). Default = M
irq=L;           //use irq in slave mode. N=no, L=low; H=high; B=buffered using input/output switch. CN3 pin 7
}
```

Command **SETUP(I2C)**

Description **Setup the protocol for the I2C communication**

Syntax/Parameters **setup(i2c){...}**

Options

```
   set = "C3E";            //quick set up of I2C - Slave with Command and Address

   addr="\\3E";          //set the address pair of the TFT module (Slave mode) in HEX code, range \\01-\\7F
   end="\\00";           //byte returned when no data left in display's i2c transmit buffer
   active=S;             //set as Master (M) or Slave (S) or disabled (N). Default = N
   speed=100;            //set transmit speed value in kilobits/sec from 20 to 400 for master mode. Default = 100
   rxi=Y;                //set receive buffer interface as active (Y), a command processing source (C) or disable (N). Default = N
   proc=“;”;             //process on receive termination character(s)
   procDel=Y;            //remove or keep the termination character(s) before processing
   procNum=5;    //interrupt on n bytes received as alternative to proc and procDel.
   encode=s;             //set s=ASCII, w=UNICODE, m=UTF8 or use sr specifying raw text bytes and sd for raw data.
   rxb=8192;             //set size of receive buffer in bytes. Default = 8192 bytes
   txi=Y;                //set transmit buffer interface as active (Y), to echo command processing (E) or disable (N)
   txb=8192;             //set size of transmit buffer in bytes. Default = 8192 bytes
```

Example

```
setup(i2c)
{
set = "C3E";            //quick set up of I2C - Slave with Command and Address
}

setup(i2c)
{
addr="\\3E";          //set the address pair of the TFT module (Slave mode) in HEX code, range \\01-\\7F
end="\\00";           //byte returned when no data left in display's i2c transmit buffer
active=S;             //set as Master (M) or Slave (S) or disabled (N). Default = N
speed=100;            //set transmit speed value in kilobits/sec from 20 to 400 for master mode. Default = 100
rxi=Y;                //set receive buffer interface as active (Y), a command processing source (C) or disable (N). Default = N
proc=“;”;             //process on receive termination character(s)
procDel=Y;            //remove or keep the termination character(s) before processing
encode=s;             //set s=ASCII, w=UNICODE, m=UTF8 or use sr specifying raw text bytes and sd for raw data.
rxb=8192;             //set size of receive buffer in bytes. Default = 8192 bytes
txi=Y;                //set transmit buffer interface as active (Y), to echo command processing (E) or disable (N)
txb=8192;             //set size of transmit buffer in bytes. Default = 8192 bytes
}
```

Command **SETUP(USB)**

Description **Setup the protocol for the USB communication**

Syntax/Parameters **SETUP(USB){...}**

Options

```
rxi=C;
txi=Y;
rxb=1250000;
```

Example

```
SETUP(USB) {rxi=C; txi=Y; rxb=1250000;}
```

Command **SETUP(KEYIO)**

Description **Setup the protocol for the Key io either for external keyboard or inputs and outputs**

Syntax/Parameters **setup(keyio){...}**

Options

```
   active=\\000000FF;            //high is active “\\00000000” >“\\7FFFFFFF”, default is inactive
   inp=\\0000000C;               //high is input, low is output “\\00000000” >“\\7FFFFFFF”
   trig=\\00000001;              //high is trigger interrupt “\\00000000” >“\\7FFFFFFF” as defined by edge. 1=trigger .
   edge=\\00000000;              //high is rising edge, low is falling edge “\\00000000” >“\\7FFFFFFF”
   keyb=\\00000FF0;              //high is scanned keyboard connection “\\00000000”>"\\7FFFFFFF”
   pullup=\\7FFFFFFF;            //introduced in v49.06 to allow input pull up resistors to be set ON=1 and OFF = 0
```

Example

```
setup(keyio)
{
active=\\000000FF;            //high is active “\\00000000” >“\\7FFFFFFF”, default is inactive
inp=\\0000000C;               //high is input, low is output “\\00000000” >“\\7FFFFFFF”
trig=\\00000001;              //high is trigger interrupt “\\00000000” >“\\7FFFFFFF” as defined by edge. 1=trigger .
edge=\\00000000;              //high is rising edge, low is falling edge “\\00000000” >“\\7FFFFFFF”
keyb=\\00000FF0;              //high is scanned keyboard connection “\\00000000”>"\\7FFFFFFF”
pullup=\\7FFFFFFF;            //introduced in v49.06 to allow input pull up resistors to be set ON=1 and OFF = 0
}
```

Command **SETUP(ENC)**

Description **Setup the protocol for the Rotary Encoder communication**

Syntax/Parameters **SETUP(ENC){...}**

Options

```
active = 1/2/12; // N=none active, 1=enc1 active, 2 = enc2 active, 12 = both active
a1 = \\xx; // port number for enc1 A channel
b1 = \\xx; // port number for enc1 B channel
a2 = \\xx; // port number for enc2 A channel
b2 = \\xx; // port number for enc2 B channel
debounce1 = n; // debounce time in ms for enc1 (1 - 100ms)
debounce2 = n; // debounce time in ms for enc2 (1 - 100ms)
timeout1 = n; // timeout period in ms for enc1 (1 - 1000ms)
timeout2 = n; // timeout period in ms for enc2 (1 - 1000ms)
mode1 = n; // encoder type (1 or 2) for enc1
mode2 = n; // encoder type (1 or 2) for enc2
```

Example

```
SETUP(ENC)
{
active = 12; // N=none active, 1=enc1 active, 2 = enc2 active, 12 = both active
a1 = \\09; // port number for enc1 A channel
b1 = \\03; // port number for enc1 B channel
a2 = \\00; // port number for enc2 A channel
b2 = \\06; // port number for enc2 B channel
debounce1 = 50; // debounce time in ms for enc1 (1 - 100ms)
debounce2 = 50; // debounce time in ms for enc2 (1 - 100ms)
timeout1 = 500; // timeout period in ms for enc1 (1 - 1000ms)
timeout2 = 500; // timeout period in ms for enc2 (1 - 1000ms)
mode1 = 1; // encoder type (1 or 2) for enc1
mode2 = 1; // encoder type (1 or 2) for enc2
}
```

Command **SETUP(ADC)**

Description **Setup the protocol for the Analogue to Digital Convertors**

Syntax/Parameters **setup(adc){...}**

Options

```
   active=num;          //set none, ADC1, ADC2 or both
   calib1=num;          //set value to use for calibration/scaling of ADC1
   calib2=num;          //set value to use for calibration/scaling of ADC2
   avg1=num;            //number of samples read and then averaged for ADC1
   avg2=num;             //number of samples read and then averaged for ADC2
```

Example

```
setup( adc )
{
active=12;          //set none, ADC1, ADC2 or both
calib1=0.4;         //set value to use for calibration/scaling of ADC1
calib2=0.2;         //set value to use for calibration/scaling of ADC2
avg1=16;            //number of samples read and then averaged for ADC1
avg2=16;             //number of samples read and then averaged for ADC2
}
```

Command **SETUP(PWM)**

Description **Setup the protocol for the Pulse width modulation**

Syntax/Parameters **setup(pwm){...}**

Options

```
   active=123;                  //to enable the required combination of PWM's use 1,2,3 and N=none eg 1, 12, 13, 23, 3
   OR
   active1=Y/N;   active2=Y/N;  active3=Y/N;

   pol1=H;                      //polarity = High or Low on first phase of PWM1
   pol2=H;                      //polarity = High or Low on first phase of PWM2
   pol3=H;
   cycle1="200";                //cycle time in microseconds of PWM1. Range 160Hz to 1MHz
   cycle2 = "300";              //cycle time in microseconds of PWM2. Range 160Hz to 1MHz
   cycle3 = "300";              //cycle time in microseconds of PWM3. Range 160Hz to 1MHz
   duty1= "44";                 //value of first phase as a  percentage for PWM1 = 1-99
   duty2= "56";                 //value of first phase as a  percentage for PWM2 = 1-99
   duty3= "56";                 //value of first phase as a  percentage for PWM3 = 1-99
   delay= "50";                 //delay between first phase of PWM1 and first phase of PWM2 in microseconds
```

PWM parameters cycle1, duty1, cycle2, duty2, cycle2, duty3 now accept float parameters giving greater PWM resolution. - v49.37

Example

```
setup(pwm)
{
active=123;                  //to enable the required combination of PWM's use 1,2,3 and N=none eg 1, 12, 13, 23, 3
pol1=H;                      //polarity = High or Low on first phase of PWM1
pol2=H;                      //polarity = High or Low on first phase of PWM2
pol3=H;
cycle1="200";                //cycle time in microseconds of PWM1. Range 160Hz to 1MHz
cycle2 = "300";              //cycle time in microseconds of PWM2. Range 160Hz to 1MHz
cycle3 = "300";              //cycle time in microseconds of PWM3. Range 160Hz to 1MHz
duty1= "44";                 //value of first phase as a  percentage for PWM1 = 1-99
duty2= "56";                 //value of first phase as a  percentage for PWM2 = 1-99
duty3= "56";                 //value of first phase as a  percentage for PWM3 = 1-99
delay= "50";                 //delay between first phase of PWM1 and first phase of PWM2 in microseconds
}
```

Command **SETUP(AC97)**

Description **Setup the protocol for the AC97 audio boards**

Syntax/Parameters **setup(ac97){...}**

Options

```
active = Y; // enabled or not
line_gain_L = 31; // 0 - 63
line_gain_R = 31; // 0 - 63
mic_gain_L = 31; // 0 - 63
mic_gain_R = 31; // 0 - 63
rec_sel = MIC; // MIC / LINE
rec_rate = 44100; // 0 - 48000
treble = 0; // 0 - 15
bass = 7; // 0 - 15
threeD = 0; // 0 - 15
master_volume = 100;    //0 low - 100 high
speaker_volume = 100;  //0 low - 100 high
headphone_volume = 100; //0 low - 100 high
balance = 50;   //0-Left   50-Center   100-Right
stereo_speaker = Y;    //Y when using module with stereo amp, N when using single speaker
pbdonefnc = EndFunc; // function that is called when playback is finished
pbprogfnc = UpdateVal;  // function that is called every 100ms during playback
pbmode = fore; // halts further execution of itronSmart code until playback finishes
         = back; // playback occurs in the background and the pbdonefnc is called when playback finishes (default)
```

Example

```
SETUP(AC97)
{
active = Y; // enabled or not
rec_sel = MIC; // MIC / LINE
rec_rate = 44100; // 0 - 48000
treble = 0; // 0 - 15
bass = 7; // 0 - 15
threeD = 0; // 0 - 15
master_volume = 100;    //0 low - 100 high
speaker_volume = 100;  //0 low - 100 high
headphone_volume = 100; //0 low - 100 high
balance = 50;   //0-Left   50-Center   100-Right
stereo_speaker = Y;    //Y when using module with stereo amp, N when using single speaker
pbdonefnc = EndFunc; // function that is called when playback is finished
pbprogfnc = UpdateVal;  // function that is called every 100ms during playback
pbmode = fore; // halts further execution of itronSmart code until playback finishes
}
```

Command **LIB**

Description **Load image, font, user font or user code file into the library.**
Image and Fonts from an SD/SDHC Card or NAND
Image and Font files can be BMP, JPG and FNT formats. Use iDevTFT to auto convert GIF and PNG.
Since BMP and JPG format does not contain transparency information, a colour can be specified after the file name. The rotation and scaling of an image can also be specified as in the IMG command. By default, BMP images are loaded in and converted to 32-bits-per-pixel images, whereas JPG images are converted to 16-bits-per-pixel images, saving memory usage.

Syntax/Parameters **LIB(Name,Source);**

Options **LIB() Image**
LIB(name, "SDHC/file.ext?scale=x&width=w&height=h&back=c&col=c&rotate=r&bits=b");
> 'scale'
number => percentage 1%, 2%, ..., 100%, ..., 120%, ... etc in 1% steps.
fit => non-proportional fit into the width and height specified (else panel size is assumed)
min => proportional fit into width and height specified (minimum fit - gaps top/bottom or left/right) (default, if not specified = 100%)
> 'col' = colour If specified, the destination area not containing the image is filled with the colour (height and/or width required).
> 'back' = colour If specified, then this is the transparent colour in the image.
> 'width' and 'height'
These are the destination sizes before any rotation is performed.
If width and height are not specified then the result transformation are used or if not possible then the screen size is used.
> 'rotate' = 0, 90, 180, 270 in degrees (default, if not specified = 0 degrees)
> 'bits' = 16, 32 Specifies the destination image pixel depth, either 16 bpp or 32 bpp. Jpegs default to 16 bpp (1555 ARGB), bitmaps to 32 bit (8888 ARGB).

Faster Loading
Fast start can be achieved by only loading the main menu images at start up then loading other images on demand and setting a flag to indicate they have been loaded.

Image and Font loaded from a Serial Link TBD
Where the image or font is sent over a serial interface use the following command structure.

User Code TBD
User code is submitted in 'C' and compiled by our firmware engineers subject to quotation and agreement. The resultant file is of type .BIN. The user code can then be used with the RUN(Name) command.

Text variable - v39.00
LIB command accepts text variable (or pointer to text variable) for file name.

Reset(library) - v42.00
Cears all entities from memory (apart from built in entities).
Users uploading their program via RS232/RS485/AS1 can then restart without system reset.

Image scaling for LIB() - v48.24
Added image scaling for LIB() command using bilinear scaling algorithm.
Rotation of 0, 90, 180, 270 degrees corrected in LIB() command.
Images correctly redrawn when page has been set up for update=changed.

Note: Progressive JPEG's are not supported - error message now provides more accurate information.

Example **LIB(libImg1, "SDHC/backimg.bmp?back=\\000007"); // colour for tranparency RRGGBB=\\000007**
LIB(libImg2, "NAND/backimg.bmp?back=\\000007&rotate=180&scale=75");
LIB(fntAsc16x16, "SDHC/asc16B.fnt?start=\\0020");
LIB(libImg3, "NAND/image.jpg");
JPG Image Handling
NOTE: Use of JPEG files instead of bitmaps can significantly decrease load time (though it can take longer to convert the file once loaded). However the lossy nature of JPEG may not provide accurate transparency capability and is therefore most suitable for backgrounds.

User Code
LIB(prog1, "SDHC/ourprog.bin");
LIB(prog2, "RS2/bin?size=36574");

Loaded from Serial Link
LIB(libImg1, "RS2/bmp?size=2438&back=\\FFFFFF&rotate=180&scale=75");
LIB(libImg2, "RS4/bmp?size=1076&back=\\FFFFFF");
LIB(fntAsc1, "SPI/fnt?size=23765&start=\\0000");

Command **FPROG**
FEND

Description FPROG and FEND are used to program subsequent commands into internal flash memory. Use the RESET(NAND) command after FPROG if the existing files are to be replaced, otherwise the files are appended to NAND. Subsequent LIB commands then load images and files from NAND. When the module starts up, it checks for the correct TUxxxA.mnu start file in NAND, loads it into memory but skips the code between FPROG and FEND
Each file is copied into a buffer before writing to NAND.

Syntax/Parameters
```
FPROG;
....
FEND;
```

Options A file called FPROG.MNU can be created by accessing the FILES >> Create Project Flash File in the iDevTFT development software. This can be included into the main TUxxxA.mnu file using the INC command as follows:

```
FPROG;
INC( "SDHC/fprog.mnu" );
FEND;
```

Example Example content of TU480A.mnu file on SDHC card

```
  FPROG;
  RESET( NAND );
  LOAD( NAND, "SDHC/TU480A.mnu" ); //copies itself
  LOAD( NAND, "SDHC/imgfile1.bmp" );
  LOAD( NAND, "SDHC/imgfile2.bmp" );
  FEND;

  LIB( libImg1, "NAND/imgfile1.bmp" );
  LIB( libImg2, "NAND/imgfile2.bmp" );
  LIB( libImg3, "SDHC/img3.bmp" ); //loaded from SDHC each boot up.
  etc
```

Command **INC**

Description Include another menu, style or setup file in the current file. 7 levels of include are possible.
This command can be used to reference a file containing styles and commands on the SDHC card so that it's contents are included at that point in the command process.
This enables modular design of the menu system.

The system does not recognize directory structures in the SDHC card.
Please put all active files in the root. All file names are 8 characters maximum length.

Syntax/Parameters INC(Source);

Options **Text variable - v39.00**
INC command accepts text variable (or pointer to text variable) for file name.

Maximum line length - v49.02
Maximum line length increased from 512 bytes to 8K bytes for include files.

Menu File Replacement Macro INC("SDHC/file.mnu?find1=repl1&find2=repl2&..."); - v49.46
* Implemented functionality to include search/replace in included "mnu" files.
* INC("SDHC/file.mnu?find1=repl1&find2=repl2&...");
* Code looks for $find1$ (case insensitive) and replaces with repl1 etc.

If test.mnu contains:
```
VAR(var$name1$1Txt,"",TXT);
VAR(var$name1$2Txt,"$val2$",TXT);
VAR(var$name1$3Txt,"$invalid1$",TXT);
VAR(var$NAME1$4Txt,"$invalid2",TXT);
PAGE(pg$NaMe1$Main,stPage)
{
POSN($x$,$y$); TEXT(txt$name1$1,var$name1$3Txt,stTxt);
POSN(+0,+32); TEXT(txt$name1$2,var$name1$4Txt,stTxt);
}
```

then INC("SDHC/test.mnu?name1=Alpha&val2=Beta&x=99&y=99"); creates:

```
VAR(varAlpha1Txt,"",TXT);
VAR(varAlpha2Txt,"Beta",TXT);
VAR(varAlpha3Txt,"$invalid1$",TXT);
VAR(varAlpha4Txt,"$invalid2",TXT);
PAGE(pgAlphaMain,stPage)
{
POSN(99,99); TEXT(txtAlpha1,varAlpha3Txt,stTxt);
POSN(+0,+32); TEXT(txtAlpha2,varAlpha4Txt,stTxt);
}
```

and calling it again with INC("SDHC/test.mnu?name1=Delta&val2=Hello&x=199&y=149"); creates:

```
VAR(varDelta1Txt,"",TXT);
VAR(varDelta2Txt,"Hello",TXT);
VAR(varDelta3Txt,"$invalid1$",TXT);
VAR(varDelta4Txt,"$invalid2",TXT);
PAGE(pgDeltaMain,stPage)
{
POSN(199,149); TEXT(txtDelta1,varDelta3Txt,stTxt);
POSN(+0,+32); TEXT(txtDelta2,varDelta4Txt,stTxt);
}
```

Example
```
INC( "SDHC/submenu.mnu" ); // specifies the file path on the SDcard.
INC( "NAND/menu2.mnu" ); // specifies the file path in the NAND.
INC( File1, File2, File3, ..., FileN ); // multiple files are possible
```

Command **Style**

Description Styles enable you to maintain a common theme throughout your application and reduce the number of parameters required to be passed in the Page, text, draw, image and key commands. A style is only used during the creation of an entity. When updating a text or an image, the style is omitted from the command.

Options **Updating**
Style parameters can be updated using the dot operator except sizes and watchdog values.
LOAD(ADC1.calib1,0.75);
changes the calibration value for the analogue input ADC1.

Inherited Styles - 47.24
Style inheritance using previously defined style
  > style(styleA,text){...}
  > style(styleB,styleA){...} <- firstly copy style from styleA then apply new style parameters

Style with Floats - v49.02
Style parameters now except floats (rather than throwing an error).
Floats are rounded to ints where applicable.

Centre/Center - v49.16
Added support for accepting both "centre" and "center" in parameters.

STYLE Handling Improvements - v49.37
* Maximum style parameter value length now 128 chars (was 64).
* Maximum number of settable params in one STYLE() command is now 17 (was 16).
* Improved style creation algorithm.

Colours of "none" or "transparent" - v49.37
* The value of "none" or "transparent" can now be used for all colours in the styles.

Image Rotation (90 degree steps) in STYLE - v49.42
* Images can now be rotated using the style parameter:
STYLE(stImg,IMAGE){rotate=90;} and
LOAD(stImg.rotate,90);
* Supported angles: 0, 90, 180, 270. Image is rotated clockwise.
* Transparency is set with the back=colour property.

Image Scaling in STYLE - v49.42
* Images can now be scaled using the style parameter:
STYLE(stImg,IMAGE){scale=75;} and
LOAD(stImg.scale,75);
* Supported scaling in 1% steps.
* PNG Alpha Transparency is only supported with 100% scaled images.
* Transparency is set with the back=colour property.

Style switching LOAD(text.style,newstyle); - v49.46
* Added functionality to allow the style of an entity to be changed to another.
LOAD(entName.style,newStyle);

eg
STYLE(stText1,TEXT){ font=Ascii16; col=blue; back=dimgrey; }
STYLE(stText2,TEXT){ font=Ascii8; col=green; back=mistyrose; }
...
TEXT(txtTest,"This is a test",stText1);
...
LOAD(txtTest.style,stText2);;

* Works with STYLES: TEXT, DRAW, IMAGE, PAGE, KEY.

Built-In TEXT, PAGE, DRAW, and IMAGE Styles - v49.51
A larger range of "default" styles has been added.
The built-in styles are prefixed by four characters as follows:
* Text Styles are prefixed by "DST_"
* Page Styles are prefixed by "DSP_"
* Draw Styles are prefixed by "DSD_"
* Image Styles are prefixed by "DSI_"
The styles are not created at start up. The style is only created the first time it is referenced from the users' project.

The style are created from the parameters within the style name itself, ie the colour black is taken from the actual name DSP_BLACK, so DSP_PINK would create a colour pink. These colours are from the colour table on the website.

The only limitations to this method is that the style name needs to fit within the 18 character entity name limit.

Colours
Where the styles take a colour for a parameter, then the value can be represented in one of three ways.
* The name of the colour can be specified as in the colour chart on the website, eg BLACK, BLUE, GOLD etc
* A 6 digit hexadecimal number can be used, eg 123ABC which converts to \\123ABC
* A 3 digit hexadecimal number can be used, eg FD0 which converts to \\FFDD00

Fonts
Where the style takes a font for a parameter, then one of two options are assumed.
* If the value is either 8, 16, or 32 then the built-in fonts Ascii8, Ascii16 or Ascii32 are used
* Otherwise the value is assumed to be the name of a font loaded by the LIB command, eg fnt64 from LIB(fnt64,"SDHC/name.fnt");

The rules/examples for each can be found in the appropriate sections below

Built-In KEY Style - v49.52
Added built-in touch key style:
DSK_act_deb_del_rep_cur
Where
act = 'action'
deb = 'debounce' in milliseconds
del = 'delay' in milliseconds
rep = 'repeat' in milliseconds
cur = 'curRel'
Examples
KEY(key1,func,100,20,DSK_D_9_500_200_CC); // Create Key, action = down; debounce = 9; delay = 500; repeat = 200; curRel = CC
KEY(key2,func,100,100,DSK_C____TL); // Create Key, action = change; curRel = TL
Built-In Styles - Parameters Are Optional - v49.52
Modified "built-in" styles to make parameters optional. Simply miss out the parameter between the underscores.

TEXT(txt4,"",DST___128__CC); // Create Text with only maxLen and curRel changed from defaults
PAGE(pg3,DSP__libBg){...} // Create Page with only image changed from defaults;
DRAW(dr4,40,40,DSD_B__RED__BR); // Create Draw, type = box; only specifying some parameters
KEY(key2,func,100,100,DSK_C____TL); // Create Key, action = change; curRel = TL

Command **VAR Data Styles - VAR(Name,Value,Style)**

Description Specify your own style for integer, float, pointer or text or use a built in style name.

Syntax/Parameters STYLE(stVar, Data)

```
 {
 type = U8;     // U8, U16, U32 - unsigned 8, 16 and 32 bit integer
                // S8, S16, S32 - signed 8, 16, 32 bit integer
                // TEXT for text strings
                // FLOAT for higher resolution calculation
                // POINTER for use with images
 length=64;     // For text, specify the length from 1 to 8192, default =32
 decimal=3;     // Specify the number of decimal places when type is float. Range 0 to 7, default=2
 format="dd mm YY";   //Specify RTC format. see RTC page for format character types
 location=SDRAM;      //Specify the data location as SDRAM (default) or EEPROM
 }
```

Options

Built In Styles (Add E for EEPROM types Example FLT4E)
The following pre defined 'built in' style names are available
U8/U8E - type = U8, U16/U16E - type = U16, U32/U32E - type = U32
S8/S8E - type = S8, S16/S16E - type = S16, S32/S32E - type = S32
PTR/PTRE - type = pointer, TXT/TXTE - type = TEXT, length=32
FLT1/FLT1E - type = float, decimal = 1, FLT2/FLT2E - type = float, decimal = 2
FLT3/FLT3E - type = float, decimal = 3, FLT4/FLT4E - type = float, decimal = 4

Read-Only VARs (Constants) - v49.32
Variables can be designated read-only (constants) by specifying readonly=y; in the data style:
> STYLE(U8C,data){type=u8;readonly=y;}
A system error will occur if modification to a read-only variable is requested.
Checking is performed in LOAD(), VAR() and CALC() commands only.

Built-in "Constant" styles created:
U8C, U16C, U32C, S8C, S16C, S32C, PTRC, FLT1C, FLT2C, FLT3C, FLT4C, TXT4C, TXT16C, TXTC, TXT64C, TXT128C.

Extra Built-In Data Styles TXT and FLT - v49.51
* Added TXTn, TXTnE and TXTnC data styles of type text where n is the maxLen value allocated for the data between 1 and 8192. If the name ends in an 'E' then it stored in EEPROM. If the name ends in 'C' then it is a constant (ie read-only). Examples TXT42, TXT27E, TXT32C.
* Added FLTn, FLTnE and FLTnC data styles of type float where n is the number of displayed decimal places between 0 and 17. If the name ends in an 'E' then it stored in EEPROM. If the name ends in 'C' then it is a constant (ie read-only). Examples FLT5, FLT7E, FLT4C.

```
VAR(v1,"",TXT1);
STYLE(TXT1,DATA){type=text;maxLen=1;}
VAR(v2,"",TXT8192);
STYLE(TXT8192,DATA){type=text;maxLen=8192;}

VAR(v4,"",TXT128C);
STYLE(TXT128C,DATA){type=text;maxLen=128;readonly=y;}

VAR(v5,"",TXT48E);
STYLE(TXT48E,DATA){type=text;maxLen=48;locate=eeprom;}

VAR(v6,3.141,FLT7);
STYLE(FLT7,DATA){type=float;decimal=7;}

VAR(v7,1.23,FLT8C);
STYLE(FLT8C,DATA){type=float;decimal=8;readonly=y;}

VAR(v8,2.34,FLT12E);
STYLE(FLT12E,DATA){type=float;decimal=12;locate=eeprom;}
```

Command

Page Styles - PAGE(Name,Style) {.......}

Description

The style defines the page size, position and background.

Syntax/Parameters

```
STYLE(stPage,Page) //create a style name and define as type Page
 {
 sizeX=480;      //specify width of page 1 to 3* LCD width
 sizeY=272;      //specify height of page 1 to 3* LCD height
 posX=0;         //specify the absolute X position of page on screen. -4 * LCD width to 4 * LCD width
 posY=0;         //specify the absolute Y position of page on screen. -4 * LCD height to 4 * LCD height
 back=black;     //specify background colour of page as hex \\000000 to \\FFFFFF or colour name
 image=pageimg;  //specify background image of page as SDHC path or entity name using LIB.
 }
```

Options

Pop-Up Pages - v49.03
Page style parameter "type=popUp;" or "type=p;" added to define a page as a pop-up.
Default is "type=normal;" or "type=n;" for non-pop-up page.
This allows pages the same size as the screen to be defined as a pop-up.
Pages smaller than the screen size will still always be declared as pop-ups; the "type" parameter is ignored in this case.

Built-In PAGE Styles - v49.51
Add a default PAGE style
DSP_
DSP_col
DSP_col_img
DSP_col_img_cur
Where
col = 'col' name or 3-digit hex or 6-digit hex (see "Colours" below)
img = 'image' image name
Examples
PAGE(pg1,DSP_BLACK){...} // Create Page, back = black;
PAGE(pg2,DSP_AB0056_libBg){...} // Create Page, back = \\AB0056, image = libBg;

Command **Text Styles - TEXT(Name,Text,Style)**

Description
Fonts are available using single byte, 2 byte and UTF8 multi-byte coding.
Built in ASCII fonts have the reserved names Ascii8, Ascii16, Ascii32 (case sensitive).
Other library fonts are uploaded using the LIB command and have file type .FNT
These are available for download from the character fonts web page at www.itrontft.com.

Syntax/Parameters
```
STYLE(Txt32ASC16,TEXT) //assign a name for the style like Txt32ASC16
 {
 font="ASC16B,16THAI";    //define fonts using built in or preloaded .FNT files via LIB command
 size=2;                  //a 24x24 font is expanded to a 48x48 font. default=1
 col=white;               //"\\000000" to "\\FFFFFF" or reserved words from the colour chart.
 maxLen=64;               //maximum length of text. default =32, maximum=512
 maxRows=4;               //maximum number of rows=32 where new line code \\0D\\0A is used.
 rotate=90;               //rotation relative to screen 0, 90, 180, 270. default=0
 curRel=CC;               //specify placement relative to cursor. CC Centre Centre , TC Top Centre,
 }                        //BC Bottom Centre, LC Left Centre, RC Right Centre, TL Top Left,
                          //BL Bottom Left, TR Top Right, BR Bottom Right
```

Options
Unique Font Overlay
It is possible to overlay one font over another to enable single byte operation with ASCII from 20H to 7FH and Cyrillic, Greek, Hebrew, Bengali, Tamil, Thai or Katakana from 80H to FFH. The LIB command is used to load the extended font at 0080H instead of it's normal UNICODE location. The style for a text can then specify font="MyASCII,MyThai"; causing the Thai to overlap the ASCII from 80H to FFH.

Text Alignment - v49.00
To support "monospaced" fonts (ie those that have the same x-advance), the text style parameter xtrim=Y|N has been added.
The default is Y which makes text boxes fit to the width of the visible text. However, for monospaced fonts (eg numbers) this can cause problems with the numbers 'shifting' left and right (eg number 1 is narrower than 2). To stop this, set xtrim=N;

Opacity - v49.00
Added opacity to text.
```
   STYLE( st, text )
   {   ...
   opacity = n; // n = 0..100 where 0=transparent..100=opaque (default=100)
   ...   }
   LOAD( st.opacity, num ); // Dot operator also supported.
```

Text justification - v49.14
Added text justification style property for displaying multiple lines in text box:
```
   justify = left; (or L) // Justify left (default)
   justify = right; (or R) // Justify right
   justify = centre; (or C) // Justify centre
Example1: STYLE( stText, text ) { font=Ascii16; col=blue; justify=right; }
Example2: LOAD( stText.justify, centre );
```

Built-In TEXT styles- v49.51
DST_
DST_col
DST_col_fnt
DST_col_fnt_len
DST_col_fnt_len_row
DST_col_fnt_len_row_cur
Where
col = 'col' name or 3-digit hex or 6-digit hex (see "Colours" below)
fnt = 'font' 8, 16, 32 for Ascii8, Ascii16, Ascii32 or font name (see "Fonts" below)
len = 'maxLen'
row = 'maxRows'
cur = 'curRel'
Examples
```
TEXT(txt1,"Hello",DST_BLACK_16); // Create Text, colour = black; font = built-in Ascii16;
TEXT(txt2,"",DST_F30_fnt32_8_4); // Create Text, colour = \\FF3300, font = fnt32; maxLen = 8; maxRows = 4
TEXT(txt3,"",DST_RED_F32_8_4_TL); // Create Text, colour = \\FF3300, font = F32; maxLen = 12; maxRows = 4; curRel = TL
```

Command **Draw Styles - DRAW(Name,X,Y,Style)**

Description
Draw or update a Line, Box or Circle of size X,Y or Pixel at X,Y. The entities can be an outline or filled.

Syntax/Parameters
```
STYLE(stCircleRed,DRAW)
 {
 type=B;           //Specify the type of shape to draw. type = B or Box , C or Circle, L or Line, G or Graph
 col=red;          //Specify the border colour of the shape. Use hex, colour name + alpha
 back=\\00FF66; //Specify the fill colour of the shape. Use hex, colour name + alpha
 maxX=160;         //Declare the maximum width allowing for rotation
 maxY=40;          //Declare the maximum height allowing for rotation
 rotate=0;         //Specify the rotation of the shape with respect to the screen. 0,90,180,270
 curRel=CC;        //specify placement relative to cursor. CC Centre Centre , TC Top Centre
 }                 //BC Bottom Centre, LC Left Centre, RC Right Centre, TL Top Left,
                  // BL Bottom Left, TR Top Right, BR Bottom Right
```

Options
Alpha blending
It is possible to specify transparency values with colours if the colour is entered as a 32-bit hex number the top 8 bits specify the alpha blending level.
col = \\aarrggbb; back = \\aarrggbb; where aa = alpha level.
For example, col = \\80FFFF00; gives 50% transparent yellow.

Graphs - 47.00
A number of graph styles now exist as draw types:
```
type=p; type=pixel; // Pixel Scatter - places a point at x,y
type=t; type=trace; // Trace/Line - joins the dots between current point and previous point.
type=y; type=yBar; // Bar Y - draws vertical line from 0 to y and clears from y+1 to ymax
type=x; type=xBar; // Bar X - draws horizontal line from 0 to x and clears from x+1 to xmax
```
The origin on the graph can be changed
```
xOrigin=val; // (default=0)
yOrigin=val; // (default=0)
```
The scaling of pixels can be set:
```
xScale=val; // (default=100.0) [val can be float and is a percentage]
yScale=val; // (default=100.0) [val can be float and is a percentage]
```
Note to draw graph with 0,0 at top and n,n at bottom, use yScale=-100;

The graph can be made to scroll (currently right-to-left only supported)
```
xScroll=val; // where val=0 (default - no scroll); val=n (scroll left n pixels before each plot
   STYLE(gstyle,DRAW) {
   type=trace;
   maxX=100;
   maxY=100;
   col=green;
   back=black;
   width=3;
   curRel=cc;
   xOrigin=50;
   yOrigin=50;
   xScale=200;
   yScale=200;
   xScroll=1;
   }
```

Opacity - v49.00

Added opacity to drawing objects.

```
   STYLE( st, draw)
   {   ...
   opacity = n; // n = 0..100 where 0=transparent..100=opaque (default=100)
   ...   }
   LOAD( st.opacity, num ); // Dot operator also supported.
```

Built-In DRAW styles- v49.51

DSD_
DSD_typ
DSD_typ_wid
DSD_typ_wid_col
DSD_typ_wid_col_fil
DSD_typ_wid_col_fil_cur

Where

typ = 'type' of shape (best to use the single letter here)
wid = 'width' of border
col = 'col' colour of border (see "Colours" below)
fil = 'back' colour of fill (see "Colours" below)
cur = 'curRel'

Examples

```
DRAW(dr1,100,20,DSD_BOX); // Create Draw, type = box;
DRAW(dr2,120,DSD_C_2_RED); // Create Draw, type = circle; width = 2; border colour = red;
DRAW(dr3,60,40,DSD_B_1_RED_FD0_TL,20,40); // Create Draw, type = box; width = 1; border = red; back = \\FFDD00; curRel = TL;
```

Command **Image Styles - IMG(Name,Source,Style)**

Description The image may be larger than the size specified so it is necessary to define how it will be scaled.

Syntax/Parameters
```
STYLE(MyImage,Image)
 {
 scale=100;     //The image is scaled down or up by a percentage.
                //Supports 5% steps below 100 and 100% steps above 100.
 maxX=160;      //Declare the maximum width allowing for rotation
 maxY=40;       //Declare the maximum height allowing for rotation
 rotate=0;      //Specify the rotation of the shape with respect to the screen. 0,90,180,270
 curRel=CC;    //specify placement relative to cursor. CC Centre Centre , TC Top Centre,
                //BC Bottom Centre, LC Left Centre,RC Right Centre, TL Top Left,
                //BL Bottom Left, TR Top Right, BR Bottom Right
 }
```

Options

Image Action - 47.12
The way in which an image is displayed can be changed for slideshows.
```
   STYLE(imgSt,Image){ action=type; step=pixels; }
```
> action type options are:
- i or instant = Instant (default);
- u or up = Move Up;
- d or down = Move Down;
- l or left = Move Left;
- r or right = Move Right;
- ur or ru or upright = Move Diagonal Up-Right
- dr or rd or downright = Move Diagonal Down-Right
- ul or lu or upleft = Move Diagonal Up-Left
- dl or ld or downleft = Move Diagonal Down-Left
- a or all = Sequence through all (except instant);

> step pixels defaults to 20 pixels (value 1 to minimum of lcd width or lcd height)

Opacity - v49.00
Added opacity to images.
```
   STYLE( st, image)
   {
   ...
   opacity = n; // n = 0..100 where 0=transparent..100=opaque (default=100)
   ...
   }
   LOAD( st.opacity, num ); // Dot operator also supported.
```
Restrictions: Not supported for library images with bits=16.

Built-In IMG styles- v49.51
DSI_
DSI_cur
Where
cur = 'curRel'
Examples
```
IMG(img1,libImg1,DSI_TL); // Create Image, curRel = TL;
```

Command **Key Styles - KEY(Name,Function,X,Y,Style)**

Description Specify the source of key data. Touch keys are dependent on certain SYSTEM parameters.
If you require a dual action, specify 2 keys at the same location, one with action D and one with U.

Syntax/Parameters
```
STYLE(myTouch,key)
 {
 type=touch;      //specify 'touch' screen or external 'keyio'
 debounce=250; //Specify the time delay to allow a key press to stabilise. Value in milliseconds.
 delay=1000;      //Specify the time delay before auto repeat occurs. Value in milliseconds. 0=off.
 repeat=500;      //Specify the repeat period if the key is held down. Value in milliseconds
 action = D;        // Specify D or Down and U or Up. Specify the up or down action for the key.
 curRel=CC;       //specify touch key placement relative to cursor. CC Centre Centre,
 }                      //BC Bottom Centre, LC Left Centre, RC Right Centre, TL Top Left,
                        // BL Bottom Left, TR Top Right, BR Bottom Right, TC Top Centre.
```

Options

KEY style event handler - v49.27
Added an additional parameter to the key style. evfunc allows a function to be called when any key using that style is pressed.
This function is run before the function specified in the KEY entity.
A typical use for this would be to provide a method to add a key beep.

```
   STYLE(myTouch,key)
   {
   .
   evfunc = fncBeep;
   .
   }

   FUNC(fncBeep)
   {
   LOAD(BUZZ, 100);
   }
```

Built-In KEY Style - v49.52
Added built-in touch key style:
DSK_act_deb_del_rep_cur
Where
act = 'action'
deb = 'debounce' in milliseconds
del = 'delay' in milliseconds
rep = 'repeat' in milliseconds
cur = 'curRel'
Examples
```
KEY(key1,func,100,20,DSK_D_9_500_200_CC); // Create Key, action = down; debounce = 9; delay = 500; repeat = 200; curRel = CC
KEY(key2,func,100,100,DSK_C____TL); // Create Key, action = change; curRel = TL
```

Page

Command **PAGE**

Description Create a Page or Group of entities. Pages contain entities to be shown on the display plus functions that will run as a background task only on that page. Entities are listed so that they are layered from back to front. Create the style and declare the page before using the SHOW(PageName); command.

Syntax/Parameters

```
PAGE(Name,Style)
{
..,//Page Contents
}
```

Style The style defines the page size, position and background.

```
STYLE(stPage,Page)   //create a style name and define as type Page
 {
 update=all; //define page refresh method with ;; options: 'all' or 'changed'
 sizeX=480; //specify width of page 1 to 3* LCD width
 sizeY=272; //specify height of page 1 to 3* LCD height
 posX=0;     //specify the absolute X position of page on screen. -4 * LCD width to 4 * LCD width
 posY=0;     //specify the absolute Y position of page on screen. -4 * LCD height to 4 * LCD height
 back=black;  //specify background colour of page as hex \\000000 to \\FFFFFF or colour name
 image=pageimg; //specify background image for the page using the entity name used in the LIB command to store the image .
 }
```

Options **Page Refreshing v47.00**
A new mode has been added for pages which allows for either:
 - all entities to be redrawn on a page when a double semicolon (;;) refresh is encountered (default),
 - only the entities that have been modified since last refresh to be redrawn when a double semicolon (;;) is encountered.

A SHOW(page); command will always redraw all the entities on a page.

```
STYLE(name,PAGE) { update=all; } // default: refreshes all entities on a ;;
STYLE(name,PAGE) { update=changed; } // refreshes only changed entities on a ;;
```

Page Merging - v37.00
LOAD(Page1,Page2,Page3,...); will copy all entities from Page2 into Page1, then Page3 into Page 1, then Page4 into Page 1.

Pop-Up Pages - v49.03
Page style parameter "type=popUp;" or "type=p;" added to define a page as a pop-up.
Default is "type=normal;" or "type=n;" for non-pop-up page.
This allows pages the same size as the screen to be defined as a pop-up.
Pages smaller than the screen size will still always be declared as pop-ups; the "type" parameter is ignored in this case.

Support for Pages Larger Than Screen Size - v49.43
* Added functionality to support pages defined greater than page size. The posX and posY in the page style can be used to scroll the page around.

```
STYLE(stPg,PAGE){sizeX=960;}
LOAD(stPg.posX,-480);; // Show right half of page
```

* Modified page style so that only sizeX or sizeY needs to be defined (the other size defaults to the screen size). Previously both sizeX and sizeY needed to be defined to override the default page size.

Order Page Layers PAGE(name){ent1;ent2;...} - v49.46
* Implemented Page re-ordering functionality.
* Use the PAGE(name) command with no style specified.
* The parameters are:
entName; // Specify the entity on the page
entName,action; // Specify the entity plus the action to perform.
* The format and parameters are:

```
PAGE(pgName)
{
ent0,u; // up - move entity one place forward
ent1,d; // down - move entity one place backward
ent2,f; // first - move entity to the first position (the front)
ent3,l; // last - move entity to the last position (the back)
ent4,b; // below - move entity behind previous entity in list (ent3)
ent5,a; // above - move entity in front of previous entity in list (ent4)
}
```

The second 'action' parameter is optional, and defaults to 'b' - below with the first one in the list defaulting to 'f', the top.

```
PAGE(pgName)
{
ent0; // 'f'
ent1; // 'b'
ent2; // 'b'
ent3; // 'b'
}
```

In the above, ent0 will appear at the top with ent1 behind ent0, ent2 behind ent1, and ent3 behind ent2. Any other entities on the page will then be shown behind these with their order unchanged.

PAGE(){} reordering parameters now use '=' separator - v49.47
For consistency with STYLE() and SETUP(), the PAGE(){} reordering parameters now support '=' between the parameter name and parameter type.

```
PAGE(name)
{
name=f;
name,f;
}
```

Screen Capture LOAD(SDHC, PAGE); - v49.51
* Added TFT screen capture to Bitmap file on SD Card.
* Use LOAD(SDHC,PAGE);
* Files are stored in the "SDHC/page/" directory. If it does not exist then it is created.
* Files are stored as "SDHC/page/pagename_num.bmp" where pagename is the name of the page being displayed on the screen and num is an incrementing number. eg if the page "pgTest" is being displayed then the file will be "SDHC/page/pgTest_1.bmp". A second capture would create "SDHC/page/pgTest_2.bmp".
* The stored bitmap takes into account the rotation of the screen.
* If there is enough free memory then a buffer is created and the display memory is copied instantly to the buffer and this is saved to SD card as a bitmap. This prevents screen updates modifying the required screen capture. If there is not enough free memory then the actual display memory is used and copied to SD card as a bitmap.
* Note, due to the large file sizes involved, writing the bitmap to the SD card will take typically 5-7 secs on 3.5", 9-11 secs on 4.3", 23-26 secs 5.7" and 28-32 secs on 7" displays.

Using update=changed
To gain the faster refreshing, a few rules apply.
1/ Only the screen area where the changed entity is located is redrawn.
2/ The entity is redrawn on top of any existing pixels being displayed in that area.
3/ Entities with transparent backgrounds will show all previous rendering at that location in the transparent area.
4/ Hiding an entity will not produce any visible difference until a full page refresh is performed.

To support the “update=changed;” method.
1/ Do not use images with transparent backgrounds

2/ Specify the “back” colour in the style for text.
3/ To hide an entity, a “masking” image will need to be placed over the entity.
4/ To refresh only the entities on a page that have been modified, use the double semi-colon “;;” refresh method after the last update object, e.g. IMG (imgt1,myimg);;.
5/ To refresh the whole page, use the SHOW(page); method.

Example

```
PAGE(MainPage,stPage)
 {
   POSN( 200, 208 );  KEY( StopKey, StopEvent, 95, 95, TOUCH );     //call function StopEvent
   POSN( 76, 252 ); KEY( SaveKey, SaveEvent, 62, 24, TOUCH );        //call function SaveEvent
   POSN( +80, +0 ); KEY( CalibKey, CalibEvent, 62, 24, TOUCH );        /call function CalibEvent
   POSN( +80, +0 ); KEY( ClockKey, [Show(Clock);], 62, 24, TOUCH );   //inline code to show clock
 }
```

Command **POSN**

Description Position Cursor X,Y or +X or –X or X, Y, Name/Page.
The cursor can be positioned on the display using absolute co-ordinates or moved in relation to it's current position by using +/- offset values. The origin is located at the top left of the screen.

Re-position a previously placed entity by specifying the new coo-ordinates and it's name.
This can be useful for indicator bars, simple movement animations and moving text.

It may be necessary to place the cursor on another page to create a new entity.

Syntax/Parameters POSN(X,Y,Name/Page)

Options Move multiple entities at the same time - v47.12
Multiple entities can be moved at the same time POSN(x0,y0,Ent1,Ent2,Ent3,Ent4,...);
POSN(+10,+10,Img1,keyup,keydn); moves both image and keys 10 pixels in X and Y direction
This is useful for slider bars where the bar image, key up and key down objects move in sync.

Move with update=changed - v49.19
Movement of entities POSN(x,y,ent); when update=changed supported.

POSN in DRAW, IMG, TEXT and KEY Commands-vv49.37
* Added initial positioning to commands:
a) DRAW(name,width,style,x,y);
DRAW(name,width,height,style,x,y);
DRAW(name,width,height,start,arc,style,x,y);
b) IMG(name,src,style,x,y);
c) TEXT(name,text,style,x,y);
d) KEY(name,func,width,height,style,x,y);
KEY(name,downFunc,upFunc,width,height,style,x,y);
KEY(name,downFunc,upFunc,repFunc,width,height,style,x,y);

Example POSN(+25,+0); moves the cursor 25 pixels to the right.
POSN(236,48); absolute position of x=236, y=48.
POSN(24,56,CalcPage); position cursor on calc page at x=24, y=56.
POSN(VarX,Vary); use variables with absolute values to control position of cursor
POSN(VarX,Vary,VertBar); use variables to move an entity - vertical bar
POSN(TOUCHX,TOUCHY,MyRectCursor); move a cursor to the contact point on the screen.

Example
```
STYLE(stPage, PAGE) { back=black; update=changed; }
STYLE(stCircFg, DRAW) { type=Circle; col=white; }
STYLE(stCircBg, stCircFg) { col=black; }

VAR(x,0,S16);
VAR(y,0,S16);

PAGE(pgMain, stPage)
{
POSN(x,y);
DRAW(circBg, 60, 60, stCircBg);
DRAW(circFg, 60, 60, stCircFg);
LOOP(lpMain, FOREVER)
{
POSN(x,y,circBg);
LOAD(x,TOUCHX);
LOAD(y,TOUCHY);
POSN(x,y,circFg);;
}
}
SHOW(pgMain);
```

Note: It is advised not to move entities unless necessary as the rendering times are greatly increase

Command **TEXT**

Description Create or update Text.

Use Carriage Return and/or Line Feed for multi line entry "\\0D\\0A". The font and colour are defined in the style. If the cursor relative position is 'CC' (Centre Centre) it is easy to locate text in the centre of images like buttons.

Text areas can overlap other text areas when for example a 'drop shadow' is required. Text can include embedded hex codes to access Unicode fonts and a cursor.
Faster display updates occur if text uses a solid background colour (ie no alpha blending).

Syntax/Parameters

```
TEXT(Name,Text,Style)
TExt(Name,Text,Style,PosX,PosY);
```

Style

```
STYLE(Txt32ASC16,TEXT)      //assign a name for the style like Txt32ASC16
    {
    font="ASC16B,16THAI";    //define fonts using built in or preloaded .FNT files via LIB command
    size=2;                  //a 24x24 font is expanded to a 48x48 font. default=1
    col=white;               //"\\000000"  to "\\FFFFFF" or reserved words from the colour chart.
    back=black;              //ONLY USE where a page has a style updated=changed;
    opacity = n;             // n = 0..100 where 0=transparent..100=opaque (default=100)
    maxLen=64;               //maximum length of text. default =32, maximum=512
    maxRows=4;               //maximum number of rows=32 where new line code \\0D\\0A is used.
    rotate=90;               //rotation relative to screen 0, 90, 180, 270. default=0
    justify = cur;           // Default. Justification is determined by curRel. (TL,CL,BL=left; TC,CC,BC=centre; TR,CR,BR=right)
    yAlign = cur;            // Default. Vertical alignment is determined by curRel. (TL,TC,TR=top; CL,CC,CR=centre; BL,BC,BR=bottom)
    xtrim=y;                 //trim text box to width of text. Set to N for numbers to prevent them moving according to width.
    curRel=CC;               //specify placement relative to cursor. CC Centre Centre , TC Top Centre,
    }                        //BC Bottom Centre, LC Left Centre, RC Right Centre, TL Top Left,
                             // BL Bottom Left, TR Top Right, BR Bottom Right
```

Options

Editable Text and Visible Cursor - v37.00
A text can contain single byte hex of the form \\00 to \\FF
A text can contain hidden codes for use in password and editable fields.
\\01 defines the text as a PASSWORD so that only ***** are shown.
\\02 defines a hidden cursor and \\03 a hidden cursor with insert ON
\\04 defines an underline cursor and \\05 an underline cursor with insert ON
\\06 defines a block cursor and \\07 a vertical cursor with insert ON
Always place the cursor before the applicable character.
When a page or text is hidden, the cursor remains at its current location.
The CALC() command can then be used to manipulate the text and cursor in EditBox.
Example Editable Text:
TEXT(EditBox,"Hello\\04World",8ptTextRed); this places an underline cursor at W

Fonts
Fonts are available using single byte, 2 byte and UTF8 multi-byte coding.
Built in ASCII fonts have the reserved names Ascii8, Ascii16, Ascii32 (case sensitive).
Other library fonts are uploaded using the LIB() command and have file type .FNT
These are available for download from the character fonts web page at www.itrontft.com.

Unique Font Overlay
It is possible to overlay one font over another to enable single byte operation with ASCII from \\20 to \\7F and Cyrillic, Greek, Hebrew, Bengali, Tamil, Thai or Katakana from \\80 to \\FF. The LIB() command is used to load the extended font at \\0080 instead of it's normal UNICODE location. The style for a text can then specify font="MyASCII,MyThai"; causing the Thai to overlap the ASCII from \\80 to FF.

Text Alignment - v49.00
To support "monospaced" fonts (ie those that have the same x-advance), the text style parameter xtrim=Y|N is used.
The default is Y which makes text boxes fit to the width of the visible text. However, for monospaced fonts (eg numbers) this can cause problems with the numbers 'shifting' left and right (eg number 1 is narrower than 2). To stop this, set xtrim=N;

Text Justification - v49.14
Text justification style property for displaying multiple lines in text box:
justify = left; (or L) // Justify left (default)
justify = right; (or R) // Justify right
justify = centre; (or C) // Justify centre
Example1: STYLE(stText, text) { font=Ascii16; col=blue; justify=right; }
Example2: LOAD(stText.justify, centre);;

TEXT Styles
Support for accepting both "centre" and "center" in parameters - v49.16.
Support for opacity - v49.00.

Text rotation supported - v38.00.

New Text Parameters - width, bcol, padding, xSpace, ySpace [work in progress] - v49.37
* Added parameters to create text box to text style:
width - width of border (default = 0)
bcol - colour of border (default = white)
padding - number of pixels to add between text area and border (default = 0)
xSpace - extra spacing between chars (default = 0)
ySpace - extra spacing between lines (default = 0)

Structure support in TEXT() commands - v49.44
* Structure variables can now be used for the source in the TEXT() command.

Support for immediate ints and floats in TEXT(); - v49.46
Changed TEXT(name,text,...) to support text = int, flt as well as existing var, "txt".

```
TEXT( txt1, "String", stTxt );
TEXT( txt2, 100, stTxt ); // Now supported
TEXT( txt3, 3.142, stTxt ); // Now supported
TEXT( txt4, varTxt, stTxt );
TEXT( txt5, varU8, stTxt );
TEXT( txt6, varFlt, stTxt );
```

Text Alignment not correct with rotated text - v49.43
* Default text alignment fixed for rotated text and multi-line text.

Text Wrapping & Text Boxes - v49.49 View text wrapping editor application
* Can now pass width/height in TEXT command, similar to DRAW, so that a text box/area can be created.
* This gives the following combinations:
- Create Text:
TEXT(name, "Text", style);
TEXT(name, "Text", style, posX, posY);
TEXT(name, "Text", sizeX, sizeY, style);
TEXT(name, "Text", sizeX, sizeY, style, posX, posY);
- Update Text:
TEXT(name, "Text");
TEXT(name, sizeX, sizeY);
TEXT(name, "Text", sizeX, sizeY);

* The text area can be one of the following types:
type=a | auto; (default)

if no solid back colour specified, text area resizes to fit text
if solid back colour is specified, text area keeps keeps maximum x and y of text
the maxLen x maxRows properties are only used for initial allocation of memory
type=r | resize;
text area resizes to fit current text
if sizeX and/or sizeY specified then minimum box size is sizeX/sizeY
type=m | max;
text area remains maximum text width/height of text box
if sizeX and/or sizeY specified then minimum box size is sizeX/sizeY
this is used primarily for page update=changed to wipe any previous text
type=f | fixed;
text a rea size is fixed to sizeX x sizeY

* New Text styles available:
width = border width
bcol = border colour
xSpace = character spacing
ySpace = row spacing
padding = space between border and text
maxX = maximum box width
maxY = maximum box height
wrap = text wrap (see below)

* Text wrapping options:
wrap = n; // no wrap (default)
wrap = cc; // char cols = wrap text at character at maxLen (ie num cols)
wrap = wc; // word cols = wrap text at word within maxLen (ie num cols)
wrap = cb; // char box = wrap text at character at box width
wrap = wb; // word box = wrap text at word within box width

View text wrapping editor application

Using PAGE STYLE Update=Changed
To gain the faster refreshing, a few rules apply.
1/ Only the screen area where the changed entity is located is redrawn.
2/ The entity is redrawn on top of any existing pixels being displayed in that area.
3/ Entities with transparent backgrounds will show all previous rendering at that location in the transparent area.
4/ Hiding an entity will not produce any visible difference until a full page refresh is performed.
To support the "update=changed;" method.
1/ Do not use images with transparent backgrounds
2/ Specify the "back" colour in the style for text.
3/ To hide an entity, a “masking” image will need to be placed over the entity.
4/ To refresh only the entities on a page that have been modified, use the double semi-colon ";;" refresh method after the last entity, e.g. TEXT(txt1, "Hello");;.
5/ To refresh the whole page, use the SHOW(page); method.

Example

```
TEXT(EditBox,"Hello World",st8Red12);  //creates Edit Box with user defined style st8Red12
TEXT(EditBox,"Hello People");           //modifies content of EditBox
TEXT(EditBox,TextVar);                  //modifies content of EditBox with content of variable
TEXT(EditBox,"Hello\\w0020World"); // example of unicode embedded character (see fonts page)
```

Command **DRAW**

Description Draw or update a Line, Box, Circle or Graph of size X,Y. The entities can be an outline or filled.
The colour can be enhanced using alpha blending within the draw style.
Graphs of a different colour can be superimposed on top of each other.
Faster display updates occur if draw uses a solid background colour (ie no alpha blending).
DRAW accepts VARs, signed/unsigned integers
    (U8, U16, U32, S8, S16, S32), floats (FLT) and pointers (PTR)
DRAW(PTR, VAR|INT|FLT|PTR, VAR|INT|FLT|PTR, Style);
    Note PTR refers to the entity being pointed to by PTR and not
    the PTR itself. Use LOAD(PTR > "Name"); to set a pointer.

Syntax/Parameters

```
DRAW(Name,X,Y,Style)
DRAW(Name,X,Y,Style,PosX,PosY);
DRAW(name,length,angle,stVector);
```

Style It is possible to specify transparency values with colours if the colour is entered as a 32-bit hex number the top 8 bits specify the alpha blending level.
col = \\aarrggbb;   back = \\aarrggbb;   where aa = alpha level.
For example, col = \\80FFFF00; gives 50% transparent yellow.
Support for opacity - v49.00.

```
STYLE(gstyle,DRAW)  {
    type=trace; //The shape to draw. type = B/Box; C/Circle;  L/Line;, T/Trace; P/Pixel; y/yBar; x/xBar;  v/Vector;
    maxX=100;  // Not required except for dynamic rotation where the maximum width is declared
    maxY=100;  // Not required except for dynamic rotation where the maximum height is declared
    col=green;  //Specify the line border colour of the shape. Use hex, colour name + alpha
    back=black; //Specify the fill colour of the shape. Use hex, colour name + alpha
    opacity = n;  // n = 0..100 where 0=transparent..100=opaque (default=100)
    width=3;    //Specify the line border width of the shape default = 1
    rotate=0;     //  Specify the rotation of the shape with respect to the screen. 0,90,180,270
    curRel=cc;   //specify placement relative to cursor. CC Centre Centre , TC Top Centre etc.
    xOrigin=50;   //specify graph x origin wit respect to declared graph
    yOrigin=50;  //specify graph y origin wit respect to declared graph
    xScale=200; //scale the value automatically to fit the graph
    yScale=200; //scale the value automatically to fit the graph
    xScroll=1;    //define scroll direction and increment 1=left to right one pixel, 0=no scroll
    }
```

Options **Graphs - v47.00**
A number of graph styles now exist as draw types:

```
  type=p; type=pixel;      // Pixel Scatter - places a point at x,y
  type=t; type=trace;      // Trace/Line - joins the dots between current point and previous point.
  type=y; type=yBar;       // Bar Y - draws vertical line from 0 to y and clears from y+1 to ymax
  type=x; type=xBar;       // Bar X - draws horizontal line from 0 to x and clears from x+1 to xmax
```

The origin on the graph can be changed

```
   xOrigin=val; // (default=0)
   yOrigin=val; // (default=0)
```

The scaling of pixels can be set:

```
   xScale=val; // (default=100.0) [val can be float and is a percentage]
   yScale=val; // (default=100.0) [val can be float and is a percentage]
```

Note to draw graph with 0,0 at top and n,n at bottom, use yScale=-100;

The graph can be made to scroll (currently right-to-left only supported)

```
   xScroll=val; // where val=0 (default - no scroll); val=n (scroll left n pixels before each plot
```

Please refer to the ADC analogue input section for a graph application example.

Graph Displaying Problems v49.37
* Added bounds check to prevent drawing outside of graph area.
* Fixed line wrapping within graph area.

DRAWing of Graphs - Trace Line Should Not be Wrapping - v49.42
* Improved graph trace drawing.
* Solved problem with trace line wrapping when rendering a graph.
* Extended graphics area to encompass line width and modified page rendering to deal with windowed pixels.

DRAWing of Ellipses and more DRAW parameter options - v49.37
* Added ellipse drawing, specify type=ellipse or e in the STYLE
* Added more DRAW command options

```
DRAW( name, x ); // Circle or Square modification: width=x; height=x;
DRAW( name, x, style ); // Circle or Square definition: width=x; height=x;
DRAW( name, x, y ); // All Shapes modification: width=x; height=y;
DRAW( name, x, y, style ); // All Shapes definition: width=x; height=y;
DRAW( name, x, y, s, a ); // Ellipse Arc modification: width=x; height=y; startAngle=s; arcAngle=a;
DRAW( name, x, y, s, a, style ); // Ellipse Arc definition: width=x; height=y; startAngle=s; arcAngle=a;
```

* Notes:
- 0 degrees is at 12'o'clock and positive angles are clockwise.
- The style for an arc must be defined to encompass the size of the shape if drawn from 0 to 360 degrees.
- Further work required on the fill algorithm for ellipses.
* Added option for start angle and arc angle to be in radians or degrees depending on SYSTEM.angles setting. v49.48

Graphs to support alpha blending and opacity - v49.42
* New features of alpha blending and opacity with graphs
* Support for graphs when the display is rotated 90, 180 and 270 degrees

Drawing Lines - v49.48
* New algorithm for line drawing implemented.
* Lines are created with anti-aliased edges.
* Vertical and horizontal lines are a special case and are rapidly created (as no anti-aliasing required).
* Line ends are now created perpendicular to the line rather than vertical or horizontal.

Drawing Vectors (Lines with length and angle) - v49.48
* Added new line type - vector - which is defined with a length and an angle.

```
STYLE( stVector, DRAW ) { type = v or vector; ... }
DRAW( name, length, angle, stVector );
```

where
'length' is line length,
'angle' is angle clockwise from 12'o'clock position. 'angle' is in degrees or radians depending on SYSTEM.angles setting.

Example tu800a.mnu

```
STYLE(stPage,PAGE){}
STYLE(stLine1,DRAW){type=vector;col=white;width=1;curRel=cc;}
STYLE(stLine2,DRAW){type=vector;col=white;width=2;curRel=cc;}
STYLE(stLine3,DRAW){type=vector;col=white;width=3;curRel=cc;}
STYLE(stLine4,DRAW){type=v;col=white;width=9;curRel=cc;}
STYLE(stLine5,DRAW){type=v;col=white;width=20;curRel=cc;}
STYLE(stLine6,DRAW){type=v;col=white;width=21;curRel=cc;}
STYLE(stCirc,DRAW){type=circle;width=1;col=red;curRel=cc;}
```

```
VAR(varAngle,0.0,FLT4);

PAGE(pgMain,stPage)
{
POSN(149,119); DRAW(drLine1,200,45,stLine1); DRAW(drCir1,150,stCirc); DRAW(drPt1,7,stCirc);
POSN(399,119); DRAW(drLine2,200,45,stLine2); DRAW(drCir2,150,stCirc); DRAW(drPt2,7,stCirc);
POSN(649,119); DRAW(drLine3,200,45,stLine3); DRAW(drCir3,150,stCirc); DRAW(drPt3,7,stCirc);
POSN(149,359); DRAW(drLine4,200,45,stLine4); DRAW(drCir4,150,stCirc); DRAW(drPt4,7,stCirc);
POSN(399,359); DRAW(drLine5,200,45,stLine5); DRAW(drCir5,150,stCirc); DRAW(drPt5,7,stCirc);
POSN(649,359); DRAW(drLine6,200,45,stLine6); DRAW(drCir6,150,stCirc); DRAW(drPt6,7,stCirc);
LOOP(lpMain,FOREVER)
{
DRAW(drLine1,150,varAngle);
DRAW(drLine2,150,varAngle);
DRAW(drLine3,150,varAngle);
DRAW(drLine4,150,varAngle);
DRAW(drLine5,150,varAngle);
DRAW(drLine6,150,varAngle);;
CALC(varAngle,varAngle,0.5,"+");
}
}
SHOW(pgMain);
```

New Line Draw CurRel options plus offset - v49.48
* Additional parameter can now be passed to DRAW() command for Line and Vector drawing which allow for the line to be offset relative to the cursor:

```
DRAW( name, length, angle, offset, stVect );
DRAW( name, length, angle, offset );
DRAW( name, length, angle, offset, stVect, xPos, yPos );
DRAW( name, width, height, offset, stLine );
DRAW( name, width, height, offset );
DRAW( name, width, height, offset, stLine, xPos, yPos );
```

* The offset parameter is ideal for the pointer on gauges and clocks where the point of rotation is a certain distance along the line
* New curRel options allow for better line placement.

```
SA, SB, SC // Start of line. SA = Edge A; SB = Edge B; SC = Centre
EA, EB, EC // End of line. EA = Edge A; EB = Edge B; EC = Centre
MA, MB, MC // Mid-point of line. MA = Edge A; MB = Edge B; MC = Centre
OA, OB, OC // Offset along line. OA = Edge A; OB = Edge Bl OC = Centre
```

* Example: Vector length = 100, angle = 125 deg, width = 5 (not to scale)

Draw() with structures - v49.48
* DRAW() command now accepts structure parameters for width/height/angles/offset.

Anti-Aliased Line Drawing - v49.48
* New algorithm for line drawing implemented.
* Lines are created with anti-aliased edges.
* Vertical and horizontal lines are a special case and are rapidly created (as no anti-aliasing required).
* Line ends are now created perpendicular to the line rather than vertical or horizontal.

Example

Circle Example

```
DRAW(MyCircle, 32, 32, DrawCircle);
DRAW(MyCircle, 64, 64);          //modified circle is double diameter.
DRAW(MyCircle,VarX,VarY);          //modified circle using variables. Should not exceed MaxX,maxY.
```

Box Example

```
DRAW(MyBox, 32, 32, DrawBox);
DRAW(MyBox, 64, 64);          //modified circle is double diameter.
DRAW(MyBox,VarX,VarY);          //modified circle using variables. Should not exceed MaxX,maxY.
```

Line Example

```
DRAW(MyLine,10,10,lineStyle); //draws line 45 degrees top left to bottom right.
DRAW(MyLine2,10,-10,lineStyle); //draws line 45 degrees bottom left to top right.
```

Graph Example

```
DRAW(MyGraph,100,100,GraphStyle); //draws a graph window of 100x100 pixels.
DRAW(MyGraph,20,30); //draws a pixel on the graph at 20,30 relative to the origin.
DRAW(MyGraph,varX,varY); //use variables to plot a pixel on the graph.
RESET(MyGraph); //clears the graph
```

Command **Image**

Description Draw or update an Image. Source has several techniques.
If an image is pre-stored in the library, it's entity name is used for Source.
If it is to be directly loaded from the SDHC card or NAND flash, the path is the Source.
Scaling and rotation can also be specified in the LIB command.
The system does not recognize directory structures in the SDHC card.
Please put all active files in the root. All file names are 8 characters maximum length.
LIB can be used with BMP and JPG although due to the lossy nature of jpeg, it is used for non transparency images like backgrounds

Syntax/Parameters

```
IMG(Name,Source,Style)
IMG(Name,Source,Style,PosX,PosY);
```

Style **Image Styles**
The image may be larger than the size specified so it is necessary to define how it will be scaled.
Support for accepting both "centre" and "center" in parameters - v49.16.

```
STYLE(MyImage,Image)
   {
   scale=100;        // The image is scaled down or up by a percentage.
                     //Supports 5% steps below 100 and 100% steps above 100.
   maxX=160;         // Not required except for dynamic rotation where the maximum width is declared
   maxY=40;          // Not required except for dynamic rotation where the maximum height is declared
   rotate=0;         // Specify the rotation of the shape with respect to the screen. 0,90,180,270
   action =i;        // defines the way in which an image is presented on screen
   step=20;          //sets the number of pixels an image moves when the action is a moving. 1-minimum of TFT screen's x or y.
   opacity = n;      // n = 0..100 where 0=transparent..100=opaque (default=100)
   curRel=CC;        // specify placement relative to cursor. CC Centre Centre , TC Top Centre,
   }                 // BC Bottom Centre, LC Left Centre, RC Right Centre, TL Top Left,
                     // BL Bottom Left, TR Top Right, BR Bottom Right
```

If maxX and maxY are the same size as the loaded file or unspecified, the library image is used rather than a copy created which saves memory space.
24 bit images are stored as 32 bit data. 16 bit images are stored as 16 bit and only expanded to 32 bit during page refresh so optimizing memory usage.

Options **Actions - v47.12**
The way in which an image is displayed can be changed for slideshows.
STYLE(imgSt,Image){ action=type; step=pixels; }
> action type options are:
- i or instant = Instant (default);
- u or up = Move Up;
- d or down = Move Down;
- l or left = Move Left;
- r or right = Move Right;
- ur or ru or upright = Move Diagonal Up-Right
- dr or rd or downright = Move Diagonal Down-Right
- ul or lu or upleft = Move Diagonal Up-Left
- dl or ld or downleft = Move Diagonal Down-Left
- a or all = Sequence through all (except instant);

Image scaling - v48.24 and rotation - v38.00 for LIB()
Support for image scalling for LIB() command using bilinear scaling algorithm.
LIB(name, "SDHC/file.ext?scale=x&width=w&height=h&back=c&col=c&rotate=r&bits=b");
> 'scale'
number => percentage 1%, 2%, ..., 100%, ..., 120%, ... etc in 1% steps.
fit => non-proportional fit into the width and height specified (else panel size is assumed)
min => proportional fit into width and height specified (minimum fit - gaps top/bottom or left/right) (default, if not specified = 100%)
> 'col' = colour If specified, the destination area not containing the image is filled with the colour (height and/or width required).
> 'back' = colour If specified, then this is the transparent colour in the image.
> 'width' and 'height'
These are the destination sizes before any rotation is performed.
If width and height are not specified then the result transformation are used or if not possible then the screen size is used.
> 'rotate' = 0, 90, 180, 270 in degrees (default, if not specified = 0 degrees)
> 'bits' = 16, 32 Specifies the destination image pixel depth, either 16 bpp or 32 bpp. Jpegs default to 16 bpp (1555 ARGB), bitmaps to 32 bit (8888 ARGB).

Raw Image Files - v49.16
Support for new TFT raw image file (*.tri) format.
A converted library BMP/JPG can be stored as .tri format using FILE("SAVE") command.
Raw image files are loaded using LIB() command directly into the library entities memory, thus greatly speeding up loading.
File Format:
Offset Size Description
\\00 8 bytes Identifier (IUK_TRIF)
\\08 4 bytes size of image (N) in bytes (LSB first)
\\0C 4 bytes width of image in pixels (LSB first)
\\10 4 bytes height of image in pixels (LSB first)
\\14 1 byte number of bytes per pixel
\\15 3 bytes unused

Displaying .PNG image files - v49.42
* PNG images can now be displayed on the TFT module.
* The following formats are supported: 24-bit RGB, 48-bit RGB, 32-bit RGBA, 64-bit RGBA, 8-bit Greyscale, 8-bit Greyscale with 8-bit Alpha
* The less common formats of interlaced images, paletted images, and fixed-transparency are not supported.

IMG() Source Filename in a Text Var - v49.42
* IMG now supports source image filename in a text var.
eg

```
LOAD( varFilePath, "SDHC/dir1/", varFileName );
IMG( imN, varFilePath, stImg );
```

Using PAGE STYLE Update=Changed
To gain the faster refreshing, a few rules apply.
1/ Only the screen area where the changed entity is located is redrawn.
2/ The entity is redrawn on top of any existing pixels being displayed in that area.
3/ Entities with transparent backgrounds will show all previous rendering at that location in the transparent area.
4/ Hiding an entity will not produce any visible difference until a full page refresh is performed.

To support the "update=changed;" method.
1/ Do not use images with transparent backgrounds
2/ Specify the "back" colour in the style for text.
3/ To hide an entity, a "masking" image will need to be placed over the entity.
4/ To refresh only the entities on a page that have been modified, use the double semi-colon ";;" refresh method after the last update object, e.g. IMG (imgt1,myimg);;.
5/ To refresh the whole page, use the SHOW(page); method.

Example

```
IMG(MyPic,TopBtn,MyImage);          //previously stored as TopBtn using LIB command
IMG(MyPic,"sdhc/TopBtn.bmp",90,60,MyImage);  //stored on SDHC card
```

Command **KEY**

Description Create a Touch Area of size X,Y or define a Key on the external keyboard.

The touch area can have a One Touch function by using the built in style TOUCH or TOUCHR (repeat)
More sophisticated function is available on touch change with TOUCHC
Both these built in styles process when the key is depressed.
For processing at press and release, create 2 keys at the same location with different styles, one with action=DOWN; and the other with action=UP;.

When specifying an external key action, the values for X and Y indicate the contact points on the key board matrix where K0 is \\00 through to K23 which is \\17 .
This method allows dual key press capability as in SHIFT key operation.
Key scan uses ports K0-K23 which can be configured as shown in the I/O section.
Switches connected to 0V should use the I/O interrupt command INT(...);

The last touch co-ordinates are stored in predefined variables TOUCHX and TOUCHY

The touch screen can be calibrated using the command SETUP(system) { calibrate=y; }
The position of touch keys can be temporarily viewed as a grey area using
SETUP(system) { test=showTouchAreas; } and hidden again using test=hideTouchAreas.
See the SYSTEM command for global touch screen debounce, sampling and accuracy parameters.

KEY(name,func,width,height,style); now accepts ints and vars for width and height v47.00.

Restriction: If processing a function called from a KEY() command then further key presses will be ignored. Each touch key press function must be processed to completion before another can be processed. Please refer to the project example 'keyboard' for the technique to process keys.

To view Hardware Touch page for Resistive and capacitive touch setup click here

Syntax/Parameters
```
KEY(Name,Function,X,Y,Style)
KEY(Name,Function,X,Y,Style,PosX,PosY);
KEY(name,downFunc,upFunc,X,Y,style,Posx,Posy);
KEY(name,downFunc,upFunc,repFunc,X,Y,style,Posx,Posy);
```

Style
```
STYLE(myTouch,key)
     {
      type=touch;       //specify 'touch' screen  or external 'keyio'
      debounce=250;  //Specify the time delay to allow external key press to stabilise in milliseconds.
      delay=1000;       //Specify the time delay before key auto repeat occurs in milliseconds. 0=off.
      repeat=500;       //Specify the repeat period if the key is held down in milliseconds
      action = D;        //Specify D or Down and U or Up and C for change. See note below
      curRel=CC;       //specify touch key placement relative to cursor. CC Centre Centre , TC Top Centre,
      }                     //BC Bottom Centre, LC Left Centre, RC Right Centre, TL Top Left,
                           // BL Bottom Left, TR Top Right, BR Bottom Right
```

Specify the source of key data. Touch debounce and sampling is setup globally in SYSTEM or in SETUP(TOUCH) for resistive touch, projective capacitive touch and Immediate capacitive touch.
If you require a dual action, specify 2 keys at the same location, one with action D and one with U.

Options **Extended touch key repeat function - 49.16**
Extended touch key repeat function. Supported repeatnum, repeatdec and repeatend parameters to KEY style.

If none of these parameters are specified or any of them are 0 the repeat function is as before. If they are specified, after the initial delay the value in 'repeat' is used 'repeatnum' times. Then the repeat value is reduced by the value specified in repeatdec. The key repeats 'repeatnum times again and then the time is again reduced. This cycle repeats until the repeat value reaches the value in repeatend. The key then repeats with this interval until it is released.

Example, initial delay of 1s, 3x330, 3x280, 3x230, 3x180, 3x130, 100 ...
```
delay = 1000
repeat = 330
repeatnum = 3
repeatdec = 50
repeatend = 100
```

KEY Style event handler - evfunc - 49.27
An Additional parameter in the key style. evfunc allows a function to be called when any key using that style is pressed.
This function is run before the function specified in the KEY entity.
A typical use for this would be to provide a method to add a key beep.
```
STYLE(myTouch,key)
   {
   . .
   evfunc = fncBeep;
   . .
   }
FUNC(fncBeep)
   {
   LOAD(BUZZ, 100);
   }
```

Last Key Pressed - LAST_KEYIO_NAME and LAST_TOUCH_NAME - v49.51
* Added two built-in text vars that give the entity names of the last external and touch keys pressed
LAST_KEYIO_NAME
LAST_TOUCH_NAME
eg press touch key named "key1". LOAD(RS2,LAST_TOUCH_NAME); // Outputs "key1"

Action Types
Styles for touch keys action=u|d|c; (up|down|change) - where change detects key down and key up
Built in touch styles
- TOUCH with type=touch; debounce=50; repeat1=0; repeat2=0; action=D;
- TOUCHR with type=touch; debounce=50; repeat1=1000; repeat2=200; action=D;
- TOUCHC with type=touch; debounce=50; repeat1=1000; repeat2=200; action=C;

a) KEY(name,func,width,height,style);
- supports existing implementation plus must be used for external keys
- 'func' is called for key down, up and repeat, depending on key style action
b) KEY(name,downFunc,upFunc,width,height,style);
- 'downFunc' called when key down detected and for key repeat, depending on key style action
- 'upFunc' called when key up detected, depending on key style action
- either 'downFunc' and/or 'upFunc' can be omitted if no function call required
c) KEY(name,[downFunc],[upFunc],[repFunc],width,height,style);
- 'downFunc' called when key down detected, depending on key style action
- 'repFunc' called when key up detected, depending on key style action
- 'upFunc' called when key up detected, depending on key style action
- either 'downFunc' and/or 'upFunc' and/or 'repFunc' can be omitted if no function call required

Note external keys still only support actions of up and down and command KEY(name,func,x,y,style);

Example
```
KEY(TopKey,TopFnc,90,50,MyTouch);  a touch area 90x50 pixels. Create your own style MyTouch
KEY(ExtKey,ExFunc,\\07,\\10,MyIOK); This external key operates when K7 and K16 connect. Create your own style MyIOK {type=keyio}
KEY(ExtKey,ExFunc,K07,K16,MyIOK);  This external key operates when K7 and K16 connect. Create your own style MyIOK {type=keyio}
KEY(TKey,[HIDE(SPage);SHOW(TPage);],50,50,TOUCH); Inline commands instead of function
```

Plan: KEY(ExtKey,ExFunc,K07,K16,PushKey); This external key operates when K7 and K16 connect.

Examples

```
KEY(key1,[LOAD(rs2,"a");],90,84,TOUCH);   - 'a' is output on key down only
KEY(key2,[LOAD(rs2,"b");],90,84,TOUCHR);  - 'b' are output on key down and key repeat
KEY(key3,[LOAD(rs2,"c");],90,84,TOUCHC);  - 'c' are output on key down, key repeat and key up

KEY(key4,[LOAD(rs2,"d");],[LOAD(rs2,"e");],90,84,TOUCHC);- 'd' are output on key down and key repeat, 'e' is output on key up
KEY(key5,[LOAD(rs2,"f");],[LOAD(rs2,"g");],[LOAD(rs2,"h");],90,84,TOUCHC);- 'f' is output on key down, 'h' on key repeat, 'g' on key up
KEY(key6,,[LOAD(rs2,"i");],[LOAD(rs2,"j");],90,84,TOUCHC);- 'j' are output on key repeat, 'i' on key up

KEY(key7,,,[LOAD(rs2,"k");],90,84,TOUCHC);- 'k' are output on key repeat only
KEY(key8,,[LOAD(rs2,"l");],,90,84,TOUCHC);- 'l' is output on key up only
```

Command **Loop**

Description **Repeats the specified actions a number of times in a PAGE then continue. Max 28 nested loops or functions - v49.32. The value for Var1 can be a number from 1-65000 or the text FOREVER. You can exit a LOOP using the command EXIT(Name); Loops can be nested and used in PAGES or FUNCTIONS.**

Syntax/Parameters **LOOP(Name,Var1){...........}**

Options

EXIT(Name) - end loops - v47.24
```
> EXIT(name); // exit nested loops up to and including loop with name
Examples:
> LOOP(lp1,FOREVER){ CALC(x,y,z,"+"); IF(x=5?[EXIT(lp1);]); } // exit loop when x=5
```
Note, if the name provided in the EXIT(name); command does not exist in the current loop nesting, then all loops and functions are exited up to the top level. It is not possible to exit the page loop in this way.

Precautions when using LOOP() including "Array Error - Subscript Out Of Range" message
At the start of each pass through a loop, a check is performed to see if a touch screen key is being pressed and, if it is, then the associated touch key function is called. Caution must be observed with the touch key function to not modify variables that are being used within the loop otherwise undesired results can occur which can be difficult to spot or result in an error message.

Good coding practice
> Make sure variables used in a loop are not modified from a touch key function (unless this is a desired action)
> If a variable does need to be changed then set a 'flag' in the key function and test the flag in the page loop and make the change there instead.

Example
```
LOOP(MyLoop,12){SHOW(Page1);WAIT(100);SHOW(page2);WAIT(100);}  //repeat 12 times
LOOP(MyLoop,FOREVER) {SHOW(Page1);WAIT(100);SHOW(page2);WAIT(100);}

Loop Example 1
   FUNC(fn1)
     {
     VAR(ii,0,U8);
     VAR(jj,0,U8);
     VAR(kk,0,U8);
     LOOP(lp0,10)
       {
       LOOP(lp1,10)
         {
         LOOP(lp2,10)
           {
           LOAD(RS2,ii,",",jj,",",kk,"\\0d");
           CALC(kk,kk,1,"+");
           }
         CALC(jj,jj,1,"+");
         }
      CALC(ii,ii,1,"+");
       }
    LOAD(RS2,"\\0d");
    }
     RS2 Outputs: 0,0,0\\0d0,0,1\\0d\\0,0,2\\0d...9,9,9\\0d<\\0d

Loop Example 2
    KEY(k0,[LOOP(klp,10){LOAD(RS2,"*");}LOAD(rs2,"\\0d\\0a");],480,136,TOUCH);
    RS2 outputs on key press: **********\\0d\\0a

    Restriction: If the LOOP() command is within a function called from a KEY() command then further key presses will be ignored.
    Each touch key press function must be processed to completion before another can be processed.
    Please refer to the project example 'keyboard' for the technique to process keys.

Example 1 - Variables:
  VAR( varX, 0, U8 );
  // We have a simple function...
  FUNC( fnTest1 )
  {
    LOAD( varX, 0 );
    LOOP( lpTest1, 10 )
    {
      // [Touch Keys are effectively tested here]
      LOAD( RS2, varX );
      CALC( varX, varX, 1, "+" );
    }
  }
  // In a page we have...
  KEY( kyTest1, [ LOAD( varX, 0 ); ], 100, 100, TOUCH );
  // Normally we would get 0123456789 sent out of the RS2 port each time fnTest1 is run
  // If however the key kyTest1 is pressed when the loop is being run then the output may be changed to 0123012345!

Example 2 - Arrays:
  VAR( varArr, 0, U8, 5 );
  VAR( varY, 0, U8 );
  // We have another simple function...
  FUNC( fnTest2 )
  {
    LOAD( varY, 0 );
    LOOP( lpTest2, 5 )
    {
     // [Touch Keys are effectively tested here]
      LOAD( RS2, varArr.varY );
      CALC( varY, varY, 1, "+" );
    }
  }
  // In a page we have...
  KEY( kyTest2, [ LOAD( varY, 0 ); ], 100, 100, TOUCH );
  KEY( kyTest3, [ LOAD( varY, 3 ); ], 100, 100, TOUCH );

  // Normally we would get the contents of varArr.0 varArr.1 varArr.2 varArr.3 varArr.4 sent out of the RS2 port each time fnTest2 is run
  // If however the key kyTest2 is pressed when the loop is being run then the output may be changed to varArr.0 varArr.1 varArr.0 varArr.1 varArr.2!
  // Or, an error when kyTest3 is pressed giving varArr.0 varArr.1 varArr.3 varArr.4 ** Array Error - Subscript Out Of Range ** (ie varArr.5 doesn't exist!)
```

Command	**SHOW**
Description	Show a Page on the Display or reveal a hidden Group or Entity This puts the selected page on the top layer of the screen. If the HIDE() command has previously been used for an entity, it will now appear on a page when the page is shown on the display. SHOW(Page) can also be used to refresh a page if entities have changed. Reserved names provide relative navigation when the name of a page may not be known.. SHOW(PREV_PAGE); Show the page which launched the current page. SHOW(THIS_PAGE); Refresh the current page SHOW(Entity1, Entity2, Entity3...);; multiple show entities then refresh current page
Syntax/Parameters	SHOW(Name)
Options	**Smaller Pages / Pop Ups** Support for popup pages - ie smaller than 480 * 272. When smaller page is displayed SHOW(SmallPage); it is overlayed onto the existing page which is then ‘frozen’. Use SHOW(PREV_PAGE); to return. **SHOW(INT) - v49.03** Pending interrupts are now processed when an interrupt entity is unhidden. All of the interrupts shown on the Int page are handled **SHOW() Groups of Entities - v49.51** Added array of pointer support to SHOW() commands. This allows for groups of entities to be shown on a page by passing the single name for the array of pointers. VAR(group>"",PTR,3); LOAD(group.0>"img1"); LOAD(group.1>"text1"); LOAD(group.2>"shape1"); SHOW(group);; // Show img1, text1, shape1

Example	INT(intRs2, RS2RXC, fncRs2Rx); // Create RS2 receive interrupt
	HIDE(intRs2); // Hide the interrupt
	SHOW(intRs2); // Show (Enable) interrupt and process pending interrupts
	
	If you don't want to process any pending interrupts when you show the interrupt then RESET the interrupt first
	INT(intTmr0, TIMER0, fncTmr0); // Create RS2 receive interrupt
	HIDE(intTmr0); // Hide the interrupt
	RESET(intTmr0); // Clear any pending interrupts
	SHOW(intTmr0); // Show (Enable) interrupt for future interrupts

Command	**HIDE**
Description	Hide a Page, Group or Entity. If the page on which a small sized page, group or entity is placed is showing on the screen and the page refreshed, the named page, group or entity will disappear from view. Touch, external keys are disabled. HIDE(Entity1, Entity2, Entity3...);; multiple hide entities then refresh current page
Syntax/Parameters	HIDE(Name)
Options	**HIDE(INT) - v49.03** Pending interrupts are now processed when an interrupt entity is unhidden. All of the interrupts shown on the Int page are handled **HIDE() Groups of Entities - v49.51** Added array of pointer support to HIDE() commands. This allows for groups of entities to be hidden on a page by passing the single name for the array of pointers. VAR(group>"",PTR,3); LOAD(group.0>"img1"); LOAD(group.1>"text1"); LOAD(group.2>"shape1"); HIDE(group);; // Hide img1, text1, shape1

Example	INT(intRs2, RS2RXC, fncRs2Rx); // Create RS2 receive interrupt
	HIDE(intRs2); // Hide the interrupt
	SHOW(intRs2); // Show (Enable) interrupt and process pending interrupts
	
	If you don't want to process any pending interrupts when you show the interrupt then RESET the interrupt first
	INT(intTmr0, TIMER0, fncTmr0); // Create RS2 receive interrupt
	HIDE(intTmr0); // Hide the interrupt
	RESET(intTmr0); // Clear any pending interrupts
	SHOW(intTmr0); // Show (Enable) interrupt for future interrupts

Command	**;; - Page Refresh**
Description	Refresh the current page. Can be used for refreshing a page after a series of entity updates without knowing which page is showing. LOAD(VOLTS,"34");LOAD(AMPS,"100");;
Syntax/Parameters	;; = SHOW(this_page);

Options	**Page Refreshing - v47.00**
	A new mode has been added for pages which allows for either
	1/ All entities to be redrawn on a page when a double semicolon (;;) refresh is encountered (default), or
	2/ Only the entities that have been modified since last refresh to be redrawn when a double semicolon (;;) is encountered.
	A SHOW(page); command will always redraw all the entities on a page.
	STYLE(name,PAGE) { update=all; } // default: refreshes all entities on a ;;
	STYLE(name,PAGE) { update=changed; } // refreshes only changed entities on a ;;

Example	LOAD(VOLTS,"34");LOAD(AMPS,"100");;
	
	FUNC(TestFunc)
	{
	SHOW(But1);
	LOAD(AMPS,"50");
	;;
	}

Data

Command **LOAD**

Description Copy Pages and Groups into a previously defined Page or Group. The background and page attributes for 'Dest' apply to the result so only entities are copied from previous pages. This allows simple templates to be merged to form a complex page.

Combine Variables, Buffers and Text and copy the result to a Variable or Buffer. This allows absolute text and variables to be joined together and sent to an interface.

Syntax/Parameters Load(Dest,Name,Name,...);

Options **Text to Integer/Float**

```
  LOAD(MyInt,MyText); //The text string is parsed until a non-valid numeric value.
  LOAD(MyInt,"1","2","3"); //MyInt = 123
```
If the string does not start with a number or +/- then the result is 0.

Example Pointers
To set/change which entity the entity pointer is pointing to you use '>' instead of ','.
```
  LOAD( EntPtr1>"Var1" ); // Set EntPtr1 to point to Var1
  LOAD( EntPtr1>"Var1",num,"3" ); // Set EntPtr1 to point to Var123 (very powerful, not found in C)
```

To put data or an entity name into the entity pointed to by the entity pointer use quotes.
```
  LOAD( EntPtr1, "ABC" );   // Load the Entity pointed to by EntPtr1 with "ABC"
```

Change Setup Parameters
To change setup parameters use the dot operator. Do not change size and watchdog parameters.
This operator works for: RS2, RS4, AS1, AS2, DBG, I2C, SPI, USB, PWM, ADC, KEYIO, SYSTEM
```
  LOAD( SYSTEM.bled, 50 );  // set backlight to 50% brightness
  LOAD( RS2.baud, 9600); // change data rate
  LOAD( RS2.baud, baudvar);  //use a variable
```

Change Style Parameters
To change style parameters use the dot operator. Some parameters, such as maximum lengths, are not changeable.
```
  LOAD( stText.col, blue );  // change text colour to blue
  LOAD( stImg.curRel, CC );; // change cursor relative position (and update display ';;')
  LOAD( stDraw.width, 2 );  // change border width to 2
```

Write files to NAND
Files can also be transferred from SDHC or serial port to NAND flash

```
  LOAD(NAND,"SDHC/TU480A.mnu"); //use in conjunction with FPROG

  LOAD(NAND,"EXT/filename.ext?size=xxx&timedate=yyy&usechecksum=0&useack=1");<CR>zzzzzzzzzzzzzzz
```
Reads xxx bytes of zzzzzz data from the current serial port and saves the data to the specified file in NAND, setting the file date to yyy
yyy is the file date/time string "YYYY:MM:DD:hh:mm:ss" as used by the RTC.
If usechecksum = 1 the file data must be followed by a 16 bit checksum of the file data (2 bytes, MSB first). The file is only written to NAND if the sent checksum matches the received data checksum.
If useack = 1 a single byte is returned after the file is written to indicate success / failure. 06h (ACK) is returned on success, 15h (NAK) is returned on failure.
Important, the data must follow immediately after the carriage return <CR> (0Dh). Unlike other commands, the optional line feed (0Ah) must not be sent as this will be interpreted as the first byte of data.

Firmware Updating - v49.00
Support has been added to update the module firmware via a serial port / USB.
```
> LOAD( NAND, "EXT/xxxxxxxx.tft?size=xxxx&usechecksum=xx" );
> LOAD( NAND, "EXT/nandboot.bin?size=xxxx&usechecksum=xx" );
> LOAD( NAND, "EXT/iuloader.bin?size=xxxx&usechecksum=xx" );
```
Note: The checksum parameter should be used to help detect problems with data corruption.
Note: It is important that power is not removed during the updates.

Packed Project File (PPF) - v49.08
PPF file can now be loaded via serial port, eg LOAD(NAND, "EXT/file.ppf?size=xxxx");.

UART Enable / Disable - v49.16
Added active=Y/N parameter to RS2/RS4/AS1/AS2/DBG/USB setup. Default is Y for backward compatibility. Allows the associated interface port pins to be disabled by using LOAD(port.active, N);

LOAD from NAND - v49.18
Added support for LOAD(NAND, "NAND/file.ppf"); which allows a Packed Project File to be loaded and files extracted.
This also allows for LOAD(SDHC, "NAND/file.ext"); which copies file from NAND to SDHC, though FILE("COPY"...) should be used in preference.

LOAD from serial port - v49.18
Files can now be loaded from serial ports using LOAD(NAND, "RS2/file.ext?size=1234");
This allows file data to be read when not operating from command mode, ie can use RUN("LOAD(NAND, \\22RS2/file.ext?size=1234\\22);"); then receive data on RS232 port.

NAND/NANDMNU/NANDLIB - v49.25
Load source of NAND/NANDMNU/NANDLIB now supported LOAD(SDHC,"NAND/file.ext");

Write files to NANN form VAR - v49.25
LOAD(NAND,"VAR/file.ext?size=xxxx",varBuf); added where the file contents are stored in a text variable.
If the file has the .ppf extension then it is processed and the containing files stored.

Text VAR Source - v49.32
Added ability to use same source text variable as the destination text variable:
LOAD(txtVarA, txtVarB, txtVarA);

Text entities - v49.32
Text entities now supported in LOAD() commands:
LOAD(txtVar, txtEnt, ...); and LOAD(txtEnt, txtVar, ...); and LOAD(RS2, txtEnt);

Support for LOAD(BUZZ, ON); and LOAD(BUZZ, OFF); - v49.42
* LOAD(BUZZ, ON); and LOAD(BUZZ, OFF); (ie ON and OFF without quotes) now supported.

Example
```
  LOAD(num,2);   //load variable num with value 2
  LOAD(EditText,EditText,"D");  //Concatonate contents of EditText with D
  LOAD(RS2, "DATE=", DTIME, "; TEMP=", ACTVAL, ";\\0D\\0A");  //send concatenated data to RS232
  LOAD(NumImg,"Image",num,".bmp"); //Create a name like Image2.bmp
  LOAD(BasePage,BaseBack,BaseEnglish); //Create page from template pages
```

ARRAYS
```
LOAD( C, Var );
LOAD( C.1, Var );
LOAD( C.1.1, Var );

LOAD( Var1, C );
LOAD( Var1, C.0.1 );
```

Command **VAR**

Description Create a variable having a certain style and a default value.
A variable contains text or numbers which can be amended and be referred to as a single name in an equation or to show information on the display.
Variable names must start with a letter or _.
Variables can be pointers to other variables and entities and use the '>' operator.
Non volatile parameter storage is also handled by VAR which initially loads the default value, then at subsequent power ON reloads the last stored value which was saved using LOAD(varname,newval);

A range of 'built in' styles exist like U8,U16,U32,S8,S16,S32,FLT1,FLT2,FLT3,FLT4,TXT as shown in VAR styles below.
These can be appended with E for storage in 'non volatile' EEPROM as described below.

Syntax/Parameters VAR(Name,Value,Style)
+ pointer usage
+ non volatile parameter storage

Arrays are defined using an extension to the VAR() command.
Each required dimension is passed as an additional parameter to the command.
VAR(name,init,type,size0); One-dimensional (or single dimension) array
VAR(name,init,type,size0,size1); Two-dimensional array
VAR(name,init,type,size0,size1,size2); Three-dimensional array
VAR(name,init,type,size0,size1,size2,size3); Four-dimensional array

Style **VAR Data Styles**
Specify your own style for integer, float, pointer or text or use a built in style name

```
STYLE(stVar, Data)
  {
   type = U8;     // U8, U16, U32 - unsigned 8, 16 and 32 bit integer
                  // S8, S16, S32 - signed 8, 16, 32 bit integer
                  // TEXT for text strings
                  // FLOAT for higher resolution calculation up to 17 decimal places
                  // POINTER for use with images
  length=64;    // For text, specify the length from 1 to 8192, default =32
  decimal=3;    // Specify the number of decimal places when type is float. Range 0 to 17, default=2
  format="dd mm YY";      //Specify RTC format. see RTC page for format character types
  location=SDRAM;           //Specify the data location as SDRAM (default) or EEPROM
  }
```

Built In Styles
The following pre defined 'built in' style names are available. Add E for EEPROM types Example FLT4E.

```
  U8/U8E      - type = U8,   U16/U16E - type = U16,  U32/U32E    - type = U32
  S8/S8E      - type = S8,   S16/S16E  - type = S16,  S32/S32E    - type = S32
  PTR/PTRE    - type = pointer,  TXT/TXTE - type = TEXT, length=32
  FLT1/FLT1E  - type = float, decimal = 1,  FLT2/FLT2E  - type = float, decimal = 2
  FLT3/FLT3E  - type = float, decimal = 3,  FLT4/FLT4E  - type = float, decimal = 4
```

Options **Example Numbers**

```
VAR(lowval,32.4,FLT1); define lowval as a single decimal float and default value 32.4
VAR(lowval,22.4,FLT1E); define lowval as a single decimal float and default value 22.7
                              or load EEPROM value if already exists.
                           Use RESET(EEPROM); to clear and reload only current values.
```

Example Pointers
Create a pointer which is defaulted to null using the '>' symbol.
VAR(EntPtr1>"",PTR);

To set/change which entity the entity pointer is pointing to you use '>' instead of ','.
  LOAD(EntPtr1>"Var1"); // Set EntPtr1 to point to Var1

To put data into the entity pointed to by the entity pointer, enclose data / source entity in quotes.
  LOAD(EntPtr1, "ABC");   // Load the Entity pointed to by EntPtr1 with ABC

The following commands now support entity pointers where (| means 'or this')

```
  > LOAD(name | ptr | "ptr", | > num | "txt" | var | ptr,...);
  > CALC(var | ptr, var | ptr, num | var | ptr,"op");
  > TEXT(name | ptr, "txt" | var | ptr,...);
  > IF(var | ptr op num | "txt" | var | ptr ? func | func_ptr : func | func_ptr);
  > KEY(name, func | func_ptr,...);
  > INT(name, buf, func | func_ptr,...);
  > SHOW(name | ptr,...);
  > HIDE(name | ptr,...);
  > RUN(name | func_ptr,...);
  > IMG(name | img_ptr, lib | img_ptr,...);
```

Floats - v47.12
Increased maximum decimal places to 17 (was 7).

Pointers - v48.24
Support added for pointers to pointers.

```
   VAR( vU16Ent, 1234, U16 );

   VAR( vPtrEnt1 > "vU16Ent", PTR );
   VAR( vPtrEnt2 > "vPtrEnt1", PTR );
   VAR( vPtrEnt3 > "vPtrEnt2", PTR );

   TEXT( txtEnt, vPtrEnt3 ); effectively puts 1234 in text box 'txtEnt'
   LOAD( vPtrEnt3, 5678 ); effectively loads vU16Ent with 5678
```

VARs - v49.02
VARs can now be initialised with other VARs, as well as immediate values.

Touch - v49.14
Added built-in variables TOUCH0, TOUCH1, TOUCH2, TOUCH3, TOUCHP to aid with touchscreen diagnotics.

ADC - v49.16
Added system variables ADC1VAL, ADC2VAL (U16) to allow direct use of conversion results.

Pointer Copy - v49.18
Pointers can be copied using CALC(dst,src,"PCOPY");

```
   VAR( v1 > "v3", PTR );
   VAR( v2 > "v4", PTR );
   VAR( v3, "alpha", TXT );
   VAR( v4, "beta", TXT );
   gives: v1>v3; v2>v4; v1=alpha; v2=beta; v3=alpha; v4=beta
   CALC( v1, v2, "PCOPY" );
   gives: v1>v4; v2>v4; v1=beta; v2=beta; v3=alpha; v4=beta
```

Text VAR Source - v49.32
Added ability to use same source text variable as the destination text variable:
LOAD(txtVarA, txtVarB, txtVarA);

Read-Only VARs (Constants) - v49.32

Variables can be designated read-only (constants) by specifying readonly=y; in the data style:
> STYLE(U8C,data){type=u8;readonly=y;}
A system error will occur if modification to a read-only variable is requested.
Checking is performed in LOAD(), VAR() and CALC() commands only.

Built-in "Constant" styles - v49.32
U8C, U16C, U32C, S8C, S16C, S32C, PTRC, FLT1C, FLT2C, FLT3C, FLT4C, TXT4C, TXT16C, TXTC, TXT64C, TXT128C.

Arrays Definitions
Array: A data structure consisting of a collection of elements (values), each identified by at least one index.
Index: A non-negative integer used to index a value in an array. Indices can also be called subscripts.
Element: A location in the array data structure which is used to store a value.
Dimension: The dimension of an array is the number of indices needed to select an element.

Indexing Arrays
Arrays use zero-based indexing, i.e. the first element of the array is indexed by 0.
For example, we define a 20 element array as:
VAR(A, 0, U8, 20);
Then the elements of the array are indexed A.0 through to A.19.
Array elements are accessed by separating the indices with a dot.
Single dimension with A.0 through to 4 dimensions with name.idx0.idx1.idx2.idx3

Example

```
VAR(lowval,32.4,FLT1); //define lowval as a single decimal float and default value 32.4
VAR(lowval,22.4,FLT1E); //define lowval as a single decimal float and default value 22.7
                                // or load EEPROM value if already exists.
                              //Use RESET(EEPROM); to clear and reload only current values.

VAR(Txt1,"Hello World",TXT);
```

Arrays

```
VAR(A,0,U8,3); //One-dimensional (or single dimension) array
VAR(B,1,U8,2,20); //Two-dimensional array
VAR(C,2,U8,2,5,20); //Three-dimensional array
VAR(D,3,U8,2,5,4,15); //Four-dimensional array
```

Command **FILE**

Description File handling commands for SD/SDHC Card and NAND Flash - v49.16.

These commands support the file input/output operations, such as file open, file read, file write, file close, file delete, etc. File and path names can be supplied either as immediate strings or via text variable entities. More complicated file names can be constructed with concatenation of text, strings, pointers, numbers etc using the file "MKFN" command. Details on directory name and file name construction can be found below in File and Directory Names.

Also the use of the file object, used to maintain an association with an open "stream" when reading from and writing to the SD/SDHC card can be found below in File Object Variable. Data is read from and written to files in multiples of bytes. The order the data bytes are read/written and the conversion of the bytes (eg ASCII or binary) can be specified during setting up of the File Object Variable.

To allow the user to manage file error conditions without causing a system error, every FILE() command returns an optional file result (fileRes) of the file action, eg "File Not Found" or "Access Denied". This is returned as an error number or a configurable text string. See more about fileRes below.

Each command is explained along with a simple example. More in-depth examples using combinations of FILE() commands can be found below in File Examples.

File and Directory Names

SD/SDHC card
Supports long file names and unlimited directories with the following limitations:
  Maximum file name length is 256 characters,
  Maximum path name length is 8191 characters (including the SDHC/),
  Directory depth is unlimited but must fit in pathname length.
  The pound '£' symbol is not supported.
File names take the format:
  "SDHC/dir1/dir2/longFileName.ext"

NAND
Supports 8.3 file names only. Subdirectories are not supported in the NAND.
File names take the format:
  "NAND/filename.ext" for automatic placement in NAND (menu files in NANDMNU, others in NANDLIB).
  "NANDLIB/filename.ext" for placement in NANDLIB area (where files are expected to be rarely updated).
  "NANDMNU/filename.ext" for placement in NANDMNU area (where files are expected to be frequently updated).

File Object Variable
When a file is to opened, read or modified in stages and then closed, the file system associates these actions with a "stream" that can identified in future actions by a file object. The user must create a unique file object variable for each open file, though the variable can be reused once a file has been closed.
You can create a file object variable with built in data styles FILEASC or FILEBIN.

```
  VAR( name, 0, FILEASC ); // name is a file object variable with data read/write as ASCII
  VAR( name, 0, FILEBIN ); // name is a file object variable with data read/write as binary (raw)
```

You can also create your own file styles to alter the way data bytes are read/written. The encode property determines how this data is managed. Refer to the data styles for the VAR() command for the options. The predefined styles are:

```
  STYLE( FILEASC, data ) { type=file; encode=sr; } // This will store a varU8 = 12 as two bytes \\31\\32
  STYLE( FILEBIN, data ) { type=file; encode=sd; } // This will store a varU8 = 12 as one byte \\0C
```

The visibility property can be checked to see if the file object variable is currently assigned to a file stream. An assigned variable is "visible", and unassigned variable is not "visible".

```
  CALC( res, varFileObj, "EVIS" ); // res is 1 if varFileObj is in use.
```

Syntax/Parameters **FILE(....)**

Options *Click "Function" to see more details on the selected file function*

"APPEND" - Append data to 'filename'.
"CLOSE" - Close an open file on SD/SDHC.
"COPY" - Copy a file.
"DATE" - Get file time and date.
"DELETE" - Delete a file.
"EXISTS" - Check whether a file or directory exists.
"GETPOS" - Get current read/write position in the file.
"MKDIR" - Make a directory on SD/SDHC.
"MKFN" - Make a file name from a list of entities and strings.
"OPEN" - Open a file on SD/SDHC for read/write/append/overwrite.
"READ" - Read data from an open file on SD/SDHC.
"READALL" - Read data from 'filename'.
"RENAME" - Rename/Move a file.
"SAVE" - Save media (eg image) to a file.
"SETPOS" - Set new read/write position in the file.
"SIZE" - Get file size.
"WRITE" - Write data to an open file on SD/SDHC.
"WRITEALL" - Write data to 'filename'.

File Result (fileRes)

Every FILE() command returns an optional file result (fileRes) of the file action. This allows the user to manage file error conditions without causing a system error. System errors will still occur for syntax errors and file system unrelated errors.
If fileRes is specified and is a numeric variable then an error number is returned. If fileRes is a text entity or text variable entity then a text string for the error is returned. These strings are preconfigured in text variable entities but can be changed by the user (eg to support different languages). The following table summarises the file results.

Number	Default Text String	Entity Name	File Result Description
0	"OK"	FILERES0	OK. No error - the function succeeded
1	"Disk I/O Layer Error"	FILERES1	Disk Error. An unrecoverable error occurred in the lower layer (disk I/O functions)
2	"Assertion Failed"	FILERES2	Internal Error. Assertion failed
3	"Physical Drive Error"	FILERES3	Not Ready. The physical drive cannot work
4	"File Not Found"	FILERES4	File Not Found. Could not find the file
5	"Path Not Found"	FILERES5	Path Not Found. Could not find the path
6	"Invalid Path Name"	FILERES6	Filename Error. The path name format is invalid
7	"Access Denied"	FILERES7	Access Denied. Access denied due to prohibited access or directory is full
8	"File Already Exists"	FILERES8	File Exists. Any object that has the same name is already existing
9	"Invalid File Object"	FILERES9	Not Object. The file/directory object is invalid
10	"Write Protected"	FILERES10	Write Protected. The physical drive is write protected
11	"Invalid Drive"	FILERES11	Invalid Drive. The logical drive number is invalid
12	"No Work Area"	FILERES12	Not Enabled. The volume has no work area
13	"No FAT Filesystem"	FILERES13	Not FAT. There is no valid FAT volume
14	"Make Filesystem Failed"	FILERES14	MKFS Aborted. (Not supported)
15	"Timeout"	FILERES15	Timeout. Could not get a grant to access the volume within defined period
16	"Locked Out"	FILERES16	Locked. The operation is rejected according to the file sharing policy

17	"LFN Buffer Overflow"	FILERES17	No Buffer. LFN working buffer could not be allocated
18	"Too Many Open Files"	FILERES18	Too Many Open Files. Number of open files > 5
19	"Invalid Parameter"	FILERES19	Invalid Parameter. Given parameter is invalid

The error strings in FILERES0 to FILERES19 are variables of type TXT and have a maximum length of 32 characters. They can be accessed and changed, as any other text variable. For example:

```
  LOAD( FILERES4, "File was not found!" ); // Change default "File Not Found" message
  TEXT( txtName, FILERES1 );; // Display the Disk Error message on the screen
```

Maximum simultaneous open files - v49.19
Increased maximum simultaneous open files to 8 (this includes menu files, images, as well as those opened with FILE() command).

File Tests - v49.32
Tests added to check filename length when reading/writing to NAND. Note maximum length is 15 characters.

Potential Future Commands - Not Yet Supported
These commands are not implemented but could be made available in future development depending on customer demand.
FILE("DIR", *[fileRes]*, dst, *[format]*, pathName); // Formatted directory listing
FILE("APPENDLN", *[fileRes]*, fileObj, fileName, *[numWritten]*, data *[, data [, ...]]*); // Open, Append Line, Close File (given fileName)
FILE("READLN", *[fileRes]*, fileObj, *[numRead]*, data *[, numToRead]*); // Read Line (given open fileObj)
FILE("STATUS", *[fileRes]*, fileName); // Get File Status
FILE("WRITELN", *[fileRes]*, fileObj, *[numWritten]*, data *[, data [, ...]]*); // Write Line (given open fileObj)
FILE("WRITELN", *[fileRes]*, fileObj, fileName, *[numWritten]*, data *[, data [, ...]]*); // Open, Write Line, Close File (given fileName)

Example **File Examples**

Example 1 - Simple Logging

```
  VAR( varFileObj, 0, FILEASC ); // Create a file object variable
  FUNC( fncLogStatus )
  {
     FILE( "APPEND", varRes, varFileObj, "SDHC/logs/log1.txt",,, varTime, ": ", varTemperature1, "\\0d\\0a" );
     IF( varRes != 0 ? fncReportError );
  }
```

Example 2 - Running a Log with Dated File Name

```
  VAR( varFileObj, 0, FILEASC ); // Create a file object variable
  VAR( varDay, -1, S16 ); // Variable to store time last log was made
  VAR( varS32Tmp, 0, S32 ); // Variable for temporary storage
  VAR( varTxtRes, "", TXT ); // Text variable for file result
  VAR( varTxtTmp, "", TXT ); // Text variable for temporary storage
  FUNC( fncTimerExpired )
  {
     LOAD( varS32Tmp, RTCHOURS );
     IF( varDay != varS32Tmp ? fncOpenLogFile );
     FILE( "WRITE", varTxtRes, varFileObj,,,varLogData1, varLogData2, "\\0a" );
     IF( varTxtRes != "OK" ? [ LOAD( RS2, "Log File Write Error: ", varTxtRes, "\\0d\\0a" ); ] );
  }
  FUNC( fncOpenLogFile )
  {
     LOAD( varDay, RTCHOURS );
     CALC( varS32Tmp, varFileObj, "EVIS" );
     IF( varS32Tmp == 1 ? [ FILE( "CLOSE", , varFileObj ); ] );
     FILE( "MKFN", varTxtRes, varTxtTmp, "SDHC/logs/log", varDay, ".txt" );
     IF( varTxtRes != "OK" ? [ LOAD( RS2, "Log File Name Error: ", varTxtRes, "\\0d\\0a" ); ] );
     FILE( "OPEN+W", varTxtRes, varFileObj, varTxtTmp );
     IF( varTxtRes != "OK" ? [ LOAD( RS2, "Log File Open Error: ", varTxtRes, "\\0d\\0a" ); ] );
  }
  INT( intTmr0, TIMER0, fncTimerExpired ); // Call function every minute
  LOAD( TIMER0, 60000, 0 ); // Create a 1 minute repetitive timer
```

Command **FILE - "APPEND"**

Description Append data to a file on SD/SDHC. Opens file, appends data to the end of the file, closes the file. Use "WRITE" if the file is already open.

Syntax/Parameters FILE("APPEND", *[fileRes]*, fileObj, fileName, *[numWritten]*, *[numToWrite]*, data *[, data [, ...]]*);

Options "APPEND" - Append data to file. - Immediate string.
fileRes - Result of file operation. See File Result. (optional) - Entity Name of numeric variable (U8, U16 etc), text variable, text.
fileObj - File object variable. See File Object Variable. - Entity name of file object variable.
fileName - File name. See File and Directory Names. -Immediate string or text variable of file name.
numWritten - Number of bytes written to file. (optional) - Entity Name of numeric variable (U32 etc).
numToWrite - Maximum number of bytes to write to file. (optional) - Entity Name of numeric variable (U32 etc).
data - One of more sets of data to write to file. - Immediate strings or numbers plus entity names of text or numeric variables or pointers to variables.

Example

```
  FILE( "APPEND", txtVarRes, varFileObj, "SDHC/logs/mylog1.txt", , , "Hello", " World: ", varCount, "\\0a" );
  FILE( "APPEND", , varFileObj, varFileName, varNumBytes, 1024, varData );
```

Command **FILE - "CLOSE"**

Description Close an open file on SD/SDHC.

Syntax/Parameters FILE("CLOSE", *[fileRes]*, fileObj);

Options "CLOSE" -Close file. - Immediate string.
fileRes - Result of file operation. See File Result. (optional) - Entity Name of numeric variable (U8, U16 etc), text variable, text.
fileObj - File object variable. See File Object Variable. - Entity name of file object variable.

Example

```
  FILE( "CLOSE", txtVarRes, varFileObj1 );
  FILE( "CLOSE", , varFileObj4 );
```

Command **FILE - "COPY"**

Description Copy a file from SD/SDHC to SD/SDHC or between SD/SDHC and NAND.

Syntax/Parameters FILE("COPY", *[fileRes]*, dstFileName, srcFileName); // Copy File - doesn't copy and reports error if file exists
FILE("COPY+O", *[fileRes]*, dstFileName, srcFileName); // Copy File - overwrites file if it exists

Options "COPY" - Copy source file to destination file if destination file does not exist. - Immediate string.
"COPY+O" - Copy source file to destination file and overwrite if destination file already exists. - Immediate string.
fileRes - Result of file operation. See File Result. (optional) - Entity Name of numeric variable (U8, U16 etc), text variable, text.
dstFileName - Destination file name. See File and Directory Names. - Immediate string or text variable of file name.
srcFileName - Source file name. See File and Directory Names. - Immediate string or text variable of file name.

Example	FILE("COPY", varRes, "NAND/tree.bmp", "SDHC/images/A Tree Somewhere.bmp");
	FILE("COPY+O", , varDstName, varSrcName);

Command	FILE - "DATE"
Description	Get file date and time.
Syntax/Parameters	FILE("DATE", *[fileRes]*, dst, *[format]*, fileName);
Options	"DATE" - Get formatted file time and date. - Immediate string. fileRes - Result of file operation. See File Result. (optional) - Entity Name of numeric variable (U8, U16 etc), text variable, text. dst - Formatted file time and date. - Entity Name of text variable. format - Format of time and date. See Date Format. Default formatting will apply if not supplied (optional) - Immediate string or text variable of format. fileName - File name. See File and Directory Names. - Immediate string or text variable of file name. If the file does not exist or there is a file error then the default date is returned Saturday 1st January 2000 00:00:00.

Example	FILE("DATE", varRes, varTxt, "d M Y H:i:s", "NAND/tree.bmp");
	FILE("DATE", , varTxt, "Y", varSrcName);

Command	FILE - "DELETE"
Description	Delete a file on SD/SDHC or NAND.
Syntax/Parameters	FILE("DELETE", *[fileRes]*, fileName);
Options	"DELETE" - Delete file. - Immediate string. fileRes - Result of file operation. See File Result. (Optional) - Entity Name of numeric variable (U8, U16 etc), text variable, text. fileName - File name. See File and Directory Names. - Immediate string or text variable of file name.

Example	FILE("DELETE", varRes, "SDHC/music/A-Nice-Bit-Of-Music.wav");
	FILE("DELETE", , varFileName);

Command	FILE - "EXISTS"
Description	Check whether a file or directory exists.
Syntax/Parameters	FILE("EXISTS", *[fileRes]*, *[dst]*, fileName);
Options	"EXISTS" - Check if file or directory exists. - Immediate string. fileRes - Result of file operation. See File Result. (Optional) - Entity Name of numeric variable (U8, U16 etc), text variable, text. dst - Result of file existence check. 0 = File/path does not exist. 1 = File/path exists. - Entity Name of numeric variable (U8, U16 etc), text variable, text. fileName - File name or path name. See File and Directory Names. - Immediate string or text variable of file/path name. This command returns a fileRes of 0 ("OK") if the file/path exists or a fileRes of 4 ("File Not Found") or 5 ("Path Not Found") if the file/path does not exist. Other fileRes values will be returned for disk errors etc.

Example	FILE("EXISTS", varRes, varExists, "SDHC/music/A-Nice-Bit-Of-Music.wav");
	FILE("EXISTS", , varExists, varFileName);

Command	FILE - "GETPOS"
Description	Get the current read/write position in the open file on SD/SDHC.
Syntax/Parameters	FILE("GETPOS", *[fileRes]*, fileObj, posn);
Options	"GETPOS" - Get current read/write position. - Immediate string. fileRes - Result of file operation. See File Result. (Optional) - Entity Name of numeric variable (U8, U16 etc), text variable, text. fileObj - File object variable. See File Object Variable. - Entity name of file object variable. posn - Current absolute read/write position from start of file. - Entity Name of numeric variable (U8, U16 etc), text variable, text.

Example	FILE("GETPOS", varRes, varFileObj3, varU32);
	FILE("GETPOS", , varFileObj, varPosn);

Command	FILE - "MKDIR" - v49.18
Description	Make a directory on the SD/SDHC
Syntax/Parameters	FILE("MKDIR", *[fileRes]*, data *[, data [, ...]]*);
Options	"MKDIR" - Make directory. - Immediate string. fileRes - Result of file operation. See File Result. (optional) - Entity Name of numeric variable (U8, U16 etc), text variable, text. data - One of more sets of data to concatenate to form the directory name. - Immediate strings or numbers plus entity names of text or numeric variables or pointers to variables.

Example	FILE("MKDIR", , "SDHC/dir1/dir2"); // Note dir1 must exist otherwise returns path error
	FILE("MKDIR", txtVarRes, "SDHC/", varDir, "/", varSubDir); // Note varDir must exist otherwise returns path error

Command	FILE - "MKFN"
Description	Make a file name from a list of entities and strings.
Syntax/Parameters	FILE("MKFN", *[fileRes]*, fileName, data *[, data [, ...]]*);
Options	"MKFN" - Make filename. - Immediate string. fileRes - Result of file operation. See File Result. (optional) - Entity Name of numeric variable (U8, U16 etc), text variable, text. fileName - Destination for concatenated file name. See File and Directory Names. -Text variable for file name. data - One of more sets of data to concatenate to form the file name. -Immediate strings or numbers plus entity names of text or numeric variables or pointers to variables.

Example	FILE("MKFN", txtVarRes, txtVarFileName, "SDHC/", varDir, "/", varBaseName, ".txt");
	FILE("MKFN", , txtVarFileName, "NAND/file", varNum, ".bmp");

Command	**FILE - "OPEN"**
Description	Open a file on SD/SDHC for read/write/append/overwrite. If the file does not exist then it is created. Not supported on NAND. Use this command with a file object variable and "READ" or "WRITE" and "CLOSE" if you wish to read or modify the file by small blocks of data. See "READALL", "WRITEALL" and "APPEND" for details of reading or modifying a whole file.

Syntax/Parameters	FILE("OPEN", *[fileRes]*, fileObj, fileName [, fileName [, ...]]); // File Open for Read (from start of file) - same as "OPEN+R"
	FILE("OPEN+R", *[fileRes]*, fileObj, fileName [, fileName [, ...]]); // File Open for Read (from start of file)
	FILE("OPEN+W", *[fileRes]*, fileObj, fileName [, fileName [, ...]]); // File Open for Write (truncate file to zero length)
	FILE("OPEN+A", *[fileRes]*, fileObj, fileName [, fileName [, ...]]); // File Open for Append (start at end of file)
	FILE("OPEN+O", *[fileRes]*, fileObj, fileName [, fileName [, ...]]); // File Open for Write (overwrite from start of file)

Options	"OPEN" - Open a file for reading data (from start of file). - Immediate string. "OPEN+R" - Open a file for reading data (from start of file). - Immediate string. "OPEN+W" - Open a file for writing data (truncate any existing file to zero length). - Immediate string. "OPEN+A" - Open a file for appending data to end of file. - Immediate string. "OPEN+O" - Open a file for overwriting data (from start of file). - Immediate string. fileRes - Result of file operation. See File Result. (optional) - Entity Name of numeric variable (U8, U16 etc), text variable, text. fileObj - File object variable. See File Object Variable. - Entity name of file object variable. fileName - File name to open. Can be a concatenation of strings. See File and Directory Names. - Immediate string or text variable of file name.

Example	FILE("OPEN+A", varRes, varFileObj, "NAND/file.txt");
	FILE("OPEN+R", varRes, varFileObj1, "SDHC/info/", varBaseName, ".txt");
	FILE("OPEN+W", , varFileObj3, varFileName);

Command	**FILE - "READ"**
Description	Read data from an open file on SD/SDHC. The file must already be opened for read access using "OPEN+R".
Syntax/Parameters	FILE("READ", *[fileRes]*, fileObj, *[numRead]*, *[numToRead]*, data);
Options	"READ" - Read data from a file at the current file pointer. - Immediate string. fileRes - Result of file operation. See File Result. (Optional) - Entity Name of numeric variable (U8, U16 etc), text variable, text. fileName - File name. See File and Directory Names. - Immediate string or text variable of file name. numRead - Number of bytes read from file. (Optional) - Entity Name of numeric variable (U32 etc). numToRead - Maximum number of bytes to read from file. (Optional) Either the size of data is read or all remaining bytes (whichever is smaller) if this omitted. - Entity Name of numeric variable (U32 etc). data - Destination of bytes read in. - Immediate strings or numbers plus entity names of text or numeric variables or pointers to variables.

Example	FILE("READ", txtVarRes, varFileObj3, , varData, 1024);
	FILE("READ", , varFileObj2, varNumBytes, varData);

Command	**FILE - "READALL"**
Description	Read data from a file on SD/SDHC or NAND. Opens file, reads data, closes the file.
Syntax/Parameters	FILE("READALL", *[fileRes]*, fileObj, fileName, *[numRead]*, *[numToRead]*, data);
Options	"READALL" - Read data from a file. - Immediate string. fileRes - Result of file operation. See File Result. (Optional) -Entity Name of numeric variable (U8, U16 etc), text variable, text. fileObj - File object variable. See File Object Variable. - Entity name of file object variable. numRead - Number of bytes read from file. (Optional) - Entity Name of numeric variable (U32 etc). numToRead - Maximum number of bytes to read from file. (Optional) Either the size of data is read or all remaining bytes (whichever is smaller) if this omitted. - Entity Name of numeric variable (U32 etc). data - Destination of bytes read in. - Immediate strings or numbers plus entity names of text or numeric variables or pointers to variables.

Example	FILE("READALL", txtVarRes, varFileObj, "SDHC/logs/mylog1.txt", , varData, 1024);
	FILE("READALL", , varFileObj, varFileName, varNumBytes, varData);

Command	**FILE - "RENAME"**
Description	Rename a file on SD/SDHC or NAND or move a file from SD/SDHC to SD/SDHC or between SD/SDHC and NAND.

Syntax/Parameters	FILE("RENAME", *[fileRes]*, dstFileName, srcFileName); // Rename/Move File - doesn't move and reports error if file exists
	FILE("RENAME+O", *[fileRes]*, dstFileName, srcFileName); // Rename/Move File - overwrites file if it exists

Options	"RENAME" - Move source file to destination file if destination file does not exist. - Immediate string. "RENAME+O" - Move source file to destination file and overwrite if destination file already exists. - Immediate string. fileRes - Result of file operation. See File Result. (Optional) -Entity Name of numeric variable (U8, U16 etc), text variable, text. dstFileName - Destination file name. See File and Directory Names. - Immediate string or text variable of file name. srcFileName - Source file name. See File and Directory Names. - Immediate string or text variable of file name. If the source file is read only, then the source file will be copied, if possible, and the "Access Denied" file result will be returned.

Example	FILE("RENAME", varRes, "NAND/tree.bmp", "SDHC/images/A Tree Somewhere.bmp");
	FILE("RENAME+O", , varDstName, varSrcName);

Command	**FILE - "SAVE"**
Description	Save media entity (image, audio etc) to a file on SD/SDHC or NAND in the appropriate file format.

Syntax/Parameters	FILE("SAVE", *[fileRes]*, fileObj, fileName, *[numWritten]*, *[numToWrite]*, data); // Save File - reports error if file exists
	FILE("SAVE+O", *[fileRes]*, fileObj, fileName, *[numWritten]*, *[numToWrite]*, data); // Save File - overwrites file if it exists

Options	"SAVE" - Save formatted source to destination file if destination file does not exist. - Immediate string. "SAVE+O" - Save formatted source to destination file and overwrite if destination file already exists. - Immediate string. fileRes - Result of file operation. See File Result. (Optional) -Entity Name of numeric variable (U8, U16 etc), text variable, text. fileObj - File object variable. See File Object Variable.- Entity name of file object variable. fileName - Destination file name, required if file not already open. See File and Directory Names. -Immediate string or text variable of file name. numWritten - -Number of bytes written to file. (Optional) - Entity Name of numeric variable (U32 etc). numToWrite - Maximum number of bytes to write to file. (Optional) -Entity Name of numeric variable (U32 etc). data - Source data entity name. -Immediate string or text variable of entity name. File extensions are: TFT raw image - ".tri" - TFT specific image file designed for fast loading. TFT raw audio - ".tra" - TFT specific audio file designed for fast loading. TFT raw layer - ".trl" - TFT specific layer file designed for fast loading. R8G8B8 bitmap - ".bmp" - Standard Bitmap. Note: Only saving processed library image to a TFT raw image ".tri" currently supported.
Example	FILE("SAVE", varRes, varFileObj, "NAND/tree.tri", , , libImgTree);

FILE("SAVE+O", , varFileObj, , , , varSrcName);

Command **FILE - "SETPOS"**

Description Set a new read/write position in the open file on SD/SDHC.

Syntax/Parameters
FILE("SETPOS", *[fileRes]*, fileObj, *[actPosn]*, reqPosn); // Set new absolute position - same as "SETPOS+A"
FILE("SETPOS+A", *[fileRes]*, fileObj, *[actPosn]*, reqPosn); // Set new absolute position
FILE("SETPOS+R", *[fileRes]*, fileObj, *[actPosn]*, reqPosn); // Set new relative position from current position

Options
"SETPOS" - Set new file position from start of file (Absolute). - Immediate string.
"SETPOS+A" - Set new file position from start of file (Absolute). - Immediate string.
"SETPOS+R" - Set new file position relative to current file position. - Immediate string.
fileRes - Result of file operation. See File Result. (Optional) - Entity Name of numeric variable (U8, U16 etc), text variable, text.
fileObj - File object variable. See File Object Variable. - Entity name of file object variable.
actPosn - New read/write position after move - absolute position from start of file. (Optional) - Entity Name of numeric variable (U8, U16 etc), text variable, text.
reqPosn - Required read/write position. For relative positions, negative numbers move the position towards the start of the file and positive numbers move the position towards the end of the file. - Immediate number or entity name of numeric variable (U8, S8, U16, S16 etc).

Example
FILE("SETPOS", varRes, varFileObj2, , varPosn);
FILE("SETPOS+R", , varFileObj, varNewPosn, -10);

Command **FILE - "SIZE"**

Description Get file size on SD/SDHC or NAND.

Syntax/Parameters FILE("SIZE", *[fileRes]*, dst, fileName);

Options
"SIZE" - Get file size. - Immediate string.
fileRes - Result of file operation. See File Result. (Optional) - Entity Name of numeric variable (U8, U16 etc), text variable, text.
dst - File size in bytes. -Entity Name of numeric variable (U8, U16 etc), text variable, text.
fileName - File name. See File and Directory Names. - Immediate string or text variable of file name.

Example
FILE("SIZE", varRes, varSize, "NAND/tree.bmp");
FILE("SIZE", , varSize, varFileName);

Command **FILE - "WRITE"**

Description Write data to an open file on SD/SDHC. The file must already be opened for write/append/overwrite access using "OPEN+W", "OPEN+A", or "OPEN+O".

Syntax/Parameters FILE("WRITE", *[fileRes]*, fileObj, *[numWritten]*, *[numToWrite]*, data *[, data [, ...]]*);

Options
"WRITE" - Write data to a file at current file pointer - Immediate string.
fileRes - Result of file operation. See File Result. (Optional) - Entity Name of numeric variable (U8, U16 etc), text variable, text.
fileObj - File object variable. See File Object Variable. - Entity name of file object variable.
numWritten - Number of bytes written to file. (Optional) - Entity Name of numeric variable (U32 etc).
numToWrite - Maximum number of bytes to write to file. (Optional) - Entity Name of numeric variable (U32 etc).
data - One of more sets of data to write to file. - Immediate strings or numbers plus entity names of text or numeric variables or pointers to variables.

Example
FILE("WRITE", txtVarRes, varFileObj2, , , "Hello", " World: ", varCount, "\\0a");
FILE("WRITE", , varFileObj1, varNumBytes, 1024, varData);

Command **FILE - "WRITEALL"**

Description Write data to a file on SD/SDHC or NAND. Opens file for write, truncates file, writes data, closes the file.

Syntax/Parameters
FILE("WRITEALL", *[fileRes]*, fileObj, fileName, *[numWritten]*, *[numToWrite]*, data *[, data [, ...]]*); // SD/SDHC
FILE("WRITEALL", *[fileRes]*, fileObj, fileName, *[numWritten]*, *[numToWrite]*, data); // NAND

Options
"WRITEALL" - Write data to a file (truncate to zero if already exists). - Immediate string.
fileRes - Result of file operation. See File Result.(Optional) - Entity Name of numeric variable (U8, U16 etc), text variable, text.
fileObj - File object variable. See File Object Variable. - Entity name of file object variable.
fileName - File name. See File and Directory Names. - Immediate string or text variable of file name.
numWritten - Number of bytes written to file.(Optional) - Entity Name of numeric variable (U32 etc).
numToWrite - Maximum number of bytes to write to file.(Optional) - Entity Name of numeric variable (U32 etc).
data - One of more sets of data to write to file. NAND only supports one data source. - Immediate strings or numbers plus entity names of text or numeric variables or pointers to variables.

Example
FILE("WRITEALL", txtVarRes, varFileObj, "SDHC/logs/mylog1.txt", , , "Hello", " World: ", varCount, "\\0a");
FILE("WRITEALL", , varFileObj, varFileName, varNumBytes, 1024, varData);

Command **CALC - Numeric Handling**

Description This provides a fast simple calculation placed in the Result variable according to the type of Method using + , - , / , *, %(modulus) or logical functions | (OR) & (AND) ^ (EXOR) for non float.
The source parameters can be text, numeric, variables or pointers as appropriate.

More complex calculation methods for maths, trigs, text and buffers are described below.

Syntax/Parameters CALC(Result,VarA,VarB,Method)

+ - Add
- - Minus
/ - Divide
* - Multiply
% - Modulus
| - OR (Logical) non float
& - AND (logical) non float
^ -EXOR (logical) non float

Command **CALC -Maths Functions**

Description This provides a fast simple calculation placed in the Result variable according to the type of Method using + , - , / , *, %(modulus) or logical functions | (OR) & (AND) ^ (EXOR) for non float.

Syntax/Parameters CALC(Result,VarA,VarB,Method)

ABS - Absolute Value of varX - CALC(varD, varX, "ABS");

EXP - Exponential Function of varX - CALC(varD, varX, "EXP");
LOG - Natural Logarithm of varX - CALC(varD, varX, "LOG");
LOG10 - Base-Ten Logarithm of varX - CALC(varD, varX, "LOG10");
POW - varX Raised to the Power of vary - CALC(varD, varX, varY,"POW");
SQRT - Non-Negative Square Root of varX - CALC(varD, varX, "SQRT");
CBRT - Cube Root of varX - CALC(varD, varX, "CBRT");
RND - Random Number Generation - CALC(varD, varX, "RND"); - v49.08

Command	CALC - Trigonometric Functions
Description	This provides a fast simple calculation placed in the Result variable according to the type of Method using + , - , / , *, %(modulus) or logical functions | (OR) & (AND) ^ (EXOR) for non float.

Syntax/Parameters	varD is result, varX is source, set system parameter angle=degrees or radians;
	COS - Cosine of varX - CALC(varD, varX, "COS");
	SIN - Sine of varX - CALC(varD, varX, "SIN");
	TAN - Tangent of varX - CALC(varD, varX, "TAN");
	
	ACOS - Arc Cosine of varX - CALC(varD, varX, "ACOS");
	ASIN - Arc Sine of varX - CALC(varD, varX, "ASIN");
	ATAN - Arc Tangent of varX - CALC(varD, varX, "ATAN");
	ATAN2 - Arc Tangent of varX/varY - CALC(varD, varX, varY, "ATAN2");
	
	COSH - Hyperbolic Cosine of varX - CALC(varD, varX, "COSH");
	SINH - Hyperbolic Sine of varX - CALC(varD, varX, "SINH");
	TANH - Hyperbolic Tangent of varX - CALC(varD, varX, "TANH");
	ACOSH - Hyperbolic Arc Cosine of varX - CALC(varD, varX, "ACOSH");
	ASINH - Hyperbolic Arc Sine of varX - CALC(varD, varX, "ASINH");
	ATANH - Hyperbolic Arc Tangent of varX - CALC(varD, varX, "ATANH");

Command	CALC - RND - *Random Number Generation*
Description	Gives varD equal to a Random number (0 to 0.999999) multiplied by varX.
Syntax/Parameters	CALC(varD, varX, "RND");

Topic	Text and Cursor Handling
Description	Calc can be used for text and cursor manipulation where editable text is to be placed on the screen as in a calculator or editable text field. Various methods allow cursor movement and type, text insertion and deletion, find or delete text, cursor position and length. VarA contains the existing text and VarB the modifier text, cursor position or a text length. Example: CALC(EditBox,EditBox, "A","INS"); Inserts the letter 'A' into the text at the cursor position Cursor and Text Types \\01 defines the text as a PASSWORD so that only ***** are shown until another \\01 or end;. \\02 defines a hidden cursor with over write and \\03 a hidden cursor with insert ON \\04 defines an underline cursor with over write and \\05 an underline cursor with insert ON \\06 defines a block cursor with over write and \\07 a ertical cursor with insert ON A detailed description follows for each method. 'dst' and 'src' can be the same text variable. 'src' is unmodified unless same text variable as 'dst'. Supported data types: The parameters can be text, numeric, variables or pointers as appropriate

Command	CALC - POS - *Move Cursor to Absolute Position*
Description	Moves cursor in text 'src' to absolute position 'pos' and stores result text in 'dst'. If 'pos' is less than zero, then cursor is put before first character ('pos'=0). If 'pos' is greater than the length of 'src' then the cursor is placed after the last character.
Syntax/Parameters	CALC(dst, src, pos, "POS");

Command	CALC - REL - *Move Cursor to Relative Position*
Description	Moves cursor in text 'src' by displacement specified in 'mov' and stores result text in 'dst'. Positive values of 'mov' move the cursor to the right and negative values of 'mov' move the cursor to the left. If the move results in a cursor position of less than zero, then the cursor is put before first character. If the move results in a cursor position greater than the length of 'src' then the cursor is placed after the last character.
Syntax/Parameters	CALC(dst, src, mov, "REL");

Command	CALC - INS - *Insert / Overwrite Text at Cursor*
Description	Puts text from 'src2' into 'src1' at the cursor and stores the result text in 'dst'. The text will either be overwritten or inserted depending on the cursor type in 'src1'. If no cursor is present then the text is appended to the end of 'src1'. 'src1' and 'src2' are unmodified unless same text variable as 'dst'
Syntax/Parameters	CALC(dst, src1, src2, "INS");

Command	CALC - DEL - *Delete Text at Cursor*
Description	Deletes 'num' characters from text 'src' at the cursor and stores the result text in 'dst'. If 'num' is positive, then 'num' characters will be deleted after cursor. If 'num' is negative, then -'num' characters will be deleted before cursor (backspace). If no cursor is present and 'num' is negative, then -'num' characters will be deleted from the end of the text in 'src'. If no cursor is present and 'num' is positive, then 'num' characters will be deleted from the start of the text in 'src'.
Syntax/Parameters	CALC(dst, src, num, "DEL");

Command	CALC - TRIM - *Trim Characters from Start and End of Text String*
Description	Removes all text characters found in 'list' from the start and end of text in 'src' and stores the result text in 'dst'. If 'list' is "" (empty string) then spaces (20hex), tabs (09hex), line feeds (0Ahex), and carriage returns (0Dhex) are removed.
Syntax/Parameters	CALC(dst, src, list, "TRIM");

Command	**CALC - LTRIM** - ***Trim Characters from Start of Text String***
Description	**Removes all text characters found in 'list' from the start of text in 'src' and stores the result text in 'dst'. If 'list' is "" (empty string) then spaces (20hex), tabs (09hex), line feeds (0Ahex), and carriage returns (0Dhex) are removed.**
Syntax/Parameters	**CALC(dst, src, list, "LTRIM");**

Command	**CALC - RTRIM** - ***Trim Characters from End of Text String***
Description	**Removes all text characters found in 'list' from the end of text in 'src' and stores the result text in 'dst'. If 'list' is "" (empty string) then spaces (20hex), tabs (09hex), line feeds (0Ahex), and carriage returns (0Dhex) are removed.**
Syntax/Parameters	**CALC(dst, src, list, "LTRIM");**

Command	**CALC - UPPER** - ***Convert Text to Uppercase***
Description	**Converts the characters 'a'-'z' to uppercase 'A'-'Z' in text 'src' and stores result text in 'dst'.**
Syntax/Parameters	**CALC(dst, src, 0, "UPPER");**

Command	**CALC - LOWER** - ***Convert Text to Lowercase***
Description	**Converts the characters 'A'-'Z' to lowercase 'a'-'z' in text 'src' and stores result text in 'dst'.**
Syntax/Parameters	**CALC(dst, src, 0, "LOWER");**

Command	**CALC - BEF** - ***Get Characters from Before Cursor***
Description	**'num' characters are copied from before the cursor in text 'src' and stored in text 'dst'. If no cursor in present then 'num' characters are copied from the end of 'src'. If 'num' is larger than the number of characters available in 'src' then only the available characters are copied. If 'num' is negative, then the function performs as "AFT".**
Syntax/Parameters	**CALC(dst, src, num, "BEF");**

Command	**CALC - AFT** - ***Get Characters from After Cursor***
Description	**'num' characters are copied from after the cursor in text 'src' and stored in text 'dst'. If no cursor in present then 'num' characters are copied from the start of 'src'. If 'num' is larger than the number of characters available in 'src' then only the available characters are copied. If 'num' is negative, then the function performs as "BEF".**
Syntax/Parameters	**CALC(dst, src, num, "AFT");**

Command	**CALC -CUR** - ***Change Cursor Type***
Description	**The cursor in text 'src' is changed to type 'type' and the result is stored in text 'dst'. If no cursor is present, then the new cursor is appended to the end. If 'type' is a string then the first character is taken as the cursor type. 'type' => integer variable | pointer to integer variable | integer | text variable | pointer to text variable | "string"**
Syntax/Parameters	**CALC(dst, src, type, "CUR");**

Command	**CALC - LEN** - ***Get Text Length***
Description	**The length of text 'src' plus 'num' is stored in variable 'len'. Cursor characters are not included in the length.**
Syntax/Parameters	**CALC(len, src, num, "LEN");**

Command	**CALC - LOC** - ***Get Cursor Location***
Description	**The location of the cursor in text 'src' plus 'num' is stored in variable 'loc'. If no cursor is present then a value of 0 is used.**
Syntax/Parameters	**CALC(loc, src, num, "LOC");**

Command	**CALC - TYPE** - ***Get Cursor Type***
Description	**The cursor type in text 'src' is stored in variable 'type'. If no cursor is present then a value of 0 is used.**
Syntax/Parameters	**CALC(type, src, 0, "TYPE");**

Command	**CALC - FIND** - ***Find Location of Text1 in Text2***
Description	**The first location of the match of text 'src2' (needle) in text 'src1' (haystack) is returned in 'loc'. If no matches are found then -1 is returned in 'loc'. Cursor characters are not included in the calculation.**
Syntax/Parameters	**CALC(loc, src1, src2, "FIND");**

Command	**CALC - LFIND** - ***Find Location of Text1 in Text2***
Description	**The last location of the match of text 'src2' (needle) in text 'src1' (haystack) is returned in 'loc'. If no matches are found then -1 is returned in 'loc'. Cursor characters are not included in the calculation.**
Syntax/Parameters	**CALC(loc, src1, src2, "LFIND");**

Command	**CALC - IFIND** - ***Find Location of Case Insensitive Text1 in Text2***
Description	**The first location of the case insensitive match of text 'src2' (needle) in text 'src1' (haystack) is returned in 'loc'**

If no case insensitive matches are found then -1 is returned in 'loc'.
Cursor characters are not included in the calculation.

Syntax/Parameters **CALC(loc, src1, src2, "FIND");**

Command **CALC - ILFIND** - ***Find Location of Case Insensitive Text1 in Text2***

Description **The last location of the case insensitive match of text 'src2' (needle) in text 'src1' (haystack) is returned in 'loc'**
If no case insensitive matches are found then -1 is returned in 'loc'.
Cursor characters are not included in the calculation.

Syntax/Parameters **CALC(loc, src1, src2, "FIND");**

Command **CALC - REM** - ***Remove Every Text1 in Text2***

Description **Remove every occurrence of text 'src2' (needle) from text 'src1' (haystack) and store the result text in 'dst'.**

Syntax/Parameters **CALC(dst, src1, src2, "REM");**

Command **CALC - IREM** - ***Remove Every Case Insensitive Text1 in Text2***

Description **Remove every case insensitive occurrence of text 'src2' (needle) from text 'src1' (haystack) and store the result text in 'dst'.**

Syntax/Parameters **CALC(dst, src1, src2, "IREM");**

Command **CALC - SPLIT** - ***Split Text at Character or String***

Description **Split the text 'src' at the character 'char' or string 'str' storing the text after 'char' or 'str' back into 'src' and storing the text before 'char' or 'str' into 'dst' or converting to number 'num'. If no 'char' or 'str' are present then the whole of 'src' is processed.**
If we have string 'str' then the first character is taken as the split character. 'src' is modified during this operation.
See BCUT for similar function where length is used instead of delimiter.

Added count option to split after nth occurrence of split string.
If n is positive then split at nth occurrence from start of src.
If n is negative then split at nth occurrence from end of src.
CALC(dst, src, str, n, "SPLIT");

Syntax/Parameters

```
CALC( dst, src, char, "SPLIT" );
CALC( num, src, char, "SPLIT" );
CALC( dst, src, str, "SPLIT" );

CALC( dst, src, str, n, "SPLIT" );
Examples (assuming 's' is set each time):
VAR(s,"Theheshethee",TXT);
VAR(d,"",TXT);
    CALC(d,s,"e","SPLIT"); // d="Th"; s="heshethee"
    CALC(d,s,"she","SPLIT"); // d="Thehe"; s="thee"
    CALC(d,s,\\65,"SPLIT"); // d="Th"; s="heshethee"
    CALC(d,s,"e",2,"SPLIT"); // d="Theh"; s="shethee"
    CALC(d,s,"e",20,"SPLIT"); // d="Theheshethee"; s=""
    CALC(d,s,"p",2,"SPLIT"); // d="Theheshethee"; s=""
    CALC(d,s,"e",-2,"SPLIT"); // d="Thehesheth"; s="e"
    CALC(d,s,"e",-20,"SPLIT"); // d="Theheshethee"; s=""
    CALC(d,s,"she",-1,"SPLIT"); // d="Thehe"; s="thee"
    CALC(d,s,"he",-1,"SPLIT"); // d="Theheshet"; s="e"
    CALC(d,s,"he",-2,"SPLIT"); // d="Thehes"; s="thee"
```

Command **CALC - PIXX** - ***Get Width of Entity***

Description **The display width in pixels of entity 'ent' plus 'num' is stored in 'size'.**
Note, variables do not have a size and return 0. Text, image, draw, touch keys, and pages do have sizes.

Syntax/Parameters **CALC(size, ent, num, "PIXX");**

Command **CALC - PIXY** - ***Get Height of Entity***

Description **The display height in pixels of entity 'ent' plus 'num' is stored in 'size'.**
Note, variables do not have a size and return 0. Text, image, draw, touch keys, and pages do have sizes.

Syntax/Parameters **CALC(size, ent, num, "PIXY");**

Command **CALC - PCOPY** - ***Pointer Copy***

Description **Pointers can be copied using CALC(dst,src,"PCOPY");**

Syntax/Parameters

```
CALC(dst,src,"PCOPY");
Example:
    VAR( v1 > "v3", PTR );
    VAR( v2 > "v4", PTR );
    VAR( v3, "alpha", TXT );
    VAR( v4, "beta", TXT );
    gives: v1>v3; v2>v4; v1=alpha; v2=beta; v3=alpha; v4=beta      CALC( v1, v2, "PCOPY" );
    gives: v1>v4; v2>v4; v1=beta; v2=beta; v3=alpha; v4=beta
```

Command **CALC - SHIFT** - ***Array Data Shift***

Description **Function shifts the values in an array up or down its indices.**

```
> CALC( array, carry, shift, "SHIFT" );
> CALC( array, shift, "SHIFT" );
- 'array' is shifted by 'shift' places, one shift at a time.
- If 'shift' is positive then array.1 -> array.2; array.0 -> array.1 etc
- If 'shift' is negative then array.1 -> array.0; array.2 -> array.1 etc
- The shift is 'circular', so the shifted out value is shifted in the other end.
- If 'carry' is specified then the shift passes through the carry, ie the carry is shifted in and a new carry is shifted out.
```

Syntax/Parameters

```
CALC( array, carry, shift, "SHIFT" );
CALC( array, shift, "SHIFT" );
```

Command	**CALC - MIN** - ***Array Minimum Values***
Description	**Function obtains the minimum values stored in an array. The minimum value in the 'array' is stored in 'val'**
Syntax/Parameters	**CALC(val, array, "MIN");**

Command	**CALC - MAX** - ***Array Maximum Values***
Description	**Function obtains the maximum values stored in an array. The maximum value in the 'array' is stored in 'val'**
Syntax/Parameters	**CALC(val, array, "MAX");**

Command	**CALC - BCOPY** - ***Buffer Copy - Copy n bytes***
Description	**Copy length from start of src or if length negative, from end Copy length from posn of src or if length negative, before posn**

Syntax/Parameters	**CALC(dst,src,len,"BCOPY");
	CALC(dst,sc,pos,len,"BCOPY");**

Command	**CALC - BCUT** - ***Buffer Cut - Cut n bytes***
Description	**Cut length from start of src or if length negative, from end and put in dst. src is modified Cut length from posn of src or if length negative, before posn and put in dst. src is modified**

Syntax/Parameters	**CALC(dst,src,len,"BCUT");
	CALC(dst,src,pos,len,"BCUT");**

Command	**CALC -BBINS** - ***Buffer Insert - Insert bytes at position***
Description	**Insert insvar at posn of src**
Syntax/Parameters	**CALC(dst,src,insvar,pos,"BINS");**

Command	**CALC - BREP** - ***Buffer Replace - Replace bytes at position***
Description	**Replace from pos of src insvar content and save in dest**
Syntax/Parameters	**CALC(dst,src,insvar,pos,"BREP");**

Command	**CALC - BFIND** - ***Buffer Find - Locate data from position***
Description	**Find first location of fvar content from pos of src. Returns -1 if not exist**
Syntax/Parameters	**CALC(dst,src,fvar,"BFIND");**

Command	**CALC - BLFIND** - ***Buffer Find Last - Locate last data from position***
Description	**Find last location of fvar content from pos of src. Returns -1 if not exist**
Syntax/Parameters	**CALC(dst,src,fvar,"BLFIND");**

Command	**CALC - BLEN** - ***Get Buffer Length - Calculate number of bytes***
Description	**Dst is length of src + num**
Syntax/Parameters	**CALC(dst,src,num,"BLEN");**

Command	**CALC - BTRIM** - ***Buffer Trim Start and End - Remove bytes beginning and end***
Description	**Remove bytes in trimvar from front and end of src**
Syntax/Parameters	**CALC(dst,src,trimvar,"BTRIM");**

Command	**CALC - BLTRIM** - ***Buffer Trim Start - Remove bytes from start***
Description	**Remove bytes in trimvar from front of src**
Syntax/Parameters	**CALC(dst,src,trimvar,"BLTRIM");**

Command	**CALC - BRTRIM** - ***Buffer Trim End - Remove bytes from end***
Description	**Remove bytes in trimvar from end of src**
Syntax/Parameters	**CALC(dst,src,trimvar,"BRTRIM");**

Command	**CALC - BREM** - ***Buffer Remove - Find and remove bytes***
Description	**Remove every occurance of remvar from src**
Syntax/Parameters	**CALC(dst,src,remvar,"BREM");**

Command	**CALC - BCNV** - ***Convert hex data in buffer to number***
Description	**Ability to convert hex data in buffer to number.**

Syntax/Parameters	**CALC(dstVar, srcTxtVar, "HEX8A", "BCNV"); // Convert 2 ASCII hex chars
	CALC(dstVar, srcTxtVar, "HEX8D", "BCNV"); // Convert 1 Binary char
	CALC(dstVar, srcTxtVar, "HEX16A", "BCNV"); // Convert 4 ASCII hex chars**

```
CALC( dstVar, srcTxtVar, "HEX16D", "BCNV" ); // Convert 2 Binary chars
CALC( dstVar, srcTxtVar, "HEX24A", "BCNV" ); // Convert 6 ASCII hex chars
CALC( dstVar, srcTxtVar, "HEX24D", "BCNV" ); // Convert 3 Binary chars
CALC( dstVar, srcTxtVar, "HEX32A", "BCNV" ); // Convert 8 ASCII hex chars
CALC( dstVar, srcTxtVar, "HEX32D", "BCNV" ); // Convert 4 Binary chars
Example
    VAR( varSrc, "12AB0123", TXT );
    VAR( varU32, 0, U32 );
    CALC( varU32, varSrc, "HEX32A", "BCNV" );
    LOAD( RS2, varU32 ); \\ Outputs 313196835
    LOAD( RS2, %h08%varU32 ); \\ Outputs 12ab0123
```

Command **CALC - BREPB -** ***Buffer Replace - Replace all occurrences of one string (buffer) with another (buffer)***

Description **Function to replace all occurrences of one string (buffer) with another (buffer)**
CALC(dst, src, search, replace, "BREPB");
All occurrences of 'search' in 'src' are replaced with 'replace' and the result is stored in 'dst'
dst and src can be the same variable.

Syntax/Parameters **Examples**

```
    VAR(s,"Theheshethee",TXT); VAR(d,"",TXT);
    CALC(d,s,"e","","BREPB"); // d="Thhshth"; s="Theheshethee"
    CALC(d,s,"e","ABC","BREPB"); // d="ThABChABCshABCthABCABC"; s="Theheshethee"
    CALC(d,s,"she","ABC","BREPB"); // d="TheheABCthee"; s="Theheshethee"
    CALC(d,s,"SHE","ABC","BREPB"); // d="Theheshethee"; s="Theheshethee"
    CALC(s,s,"e","ABC","BREPB"); // d=""; s="ThABChABCshABCthABCABC"
```

Command **CALC - BCUTB -** ***Buffer Cut - Cut chars from start of string (buffer)***

Description **Functions to cut chars from start of string (buffer)**
CALC(dst, src, "str", n, "BCUTB"); // cut up to "str" in src
CALC(dst, src, "str", "BCUTB"); // cut up to 1st "str" in src
If n is positive then cut at nth occurrence from start of src
If n is negative then cut at nth occurrence from end of src
dst and src can be the same variable.

Syntax/Parameters **Examples**

```
    VAR(s,"Theheshethee",TXT); VAR(d,"",TXT);
    CALC(d,s,"e","BCUTB"); // d="eheshethee"; s="Theheshethee"
    CALC(d,s,"she","BCUTB"); // d="shethee"; s="Theheshethee"
    CALC(d,s,\\65,"BCUTB"); // d="eheshethee"; s="Theheshethee"
    CALC(d,s,"e",2,"BCUTB"); // d="eshethee"; s="Theheshethee"
    CALC(d,s,"e",20,"BCUTB"); // d=""; s="Theheshethee"
    CALC(d,s,"p",2,"BCUTB"); // d=""; s="Theheshethee"
    CALC(d,s,"e",-2,"BCUTB"); // d="Theheshethe"; s="Theheshethee"
    CALC(d,s,"e",-20,"BCUTB"); // d=""; s="Theheshethee"
    CALC(d,s,"she",-1,"BCUTB"); // d="Theheshe"; s="Theheshethee"
    CALC(d,s,"he",-1,"BCUTB"); // d="Theheshethe"; s="Theheshethee"
    CALC(d,s,"he",-2,"BCUTB"); // d="Theheshe"; s="Theheshethee"
    CALC(s,s,"e",2,"BCUTB"); // d=""; s="eshethee"
```

Command **CALC - BCUTBI -** ***Buffer Cut - Cut chars from end of string(buffer)***

Description **Functions to cut chars from end of string (buffer)**
CALC(dst, src, "str", n, "BCUTBI"); // cut up to and including "str" in src
CALC(dst, src, "str", "BCUTBI"); // cut up to and including 1st "str" in src
If n is positive then cut at nth occurrence from start of src
If n is negative then cut at nth occurrence from end of src
*** dst and src can be the same variable.**

Syntax/Parameters *** Examples**

```
    VAR(s,"Theheshethee",TXT); VAR(d,"",TXT);
    CALC(d,s,"e","BCUTBI"); // d="heshethee"; s="Theheshethee"
    CALC(d,s,"she","BCUTBI"); // d="thee"; s="Theheshethee"
    CALC(d,s,\\65,"BCUTBI"); // d="heshethee"; s="Theheshethee"
    CALC(d,s,"e",2,"BCUTBI"); // d="shethee"; s="Theheshethee"
    CALC(d,s,"e",20,"BCUTBI"); // d=""; s="Theheshethee"
    CALC(d,s,"p",2,"BCUTBI"); // d=""; s="Theheshethee"
    CALC(d,s,"e",-2,"BCUTBI"); // d="Thehesheth"; s="Theheshethee"
    CALC(d,s,"e",-20,"BCUTBI"); // d=""; s="Theheshethee"
    CALC(d,s,"she",-1,"BCUTBI"); // d="Thehe"; s="Theheshethee"
    CALC(d,s,"he",-1,"BCUTBI"); // d="Theheshet"; s="Theheshethee"
    CALC(d,s,"he",-2,"BCUTBI"); // d="Thehes"; s="Theheshethee"
    CALC(s,s,"e",2,"BCUTBI"); // d=""; s="shethee"
```

Command **CALC - FEXISTS**

Description **Checks for existence of file in NAND or on SDHC card using CALC(dstVar, src, "FEXISTS");**
Returns dstVar=0 if file 'src' does not exist, or dstVar=1 if the file 'src' exists

Syntax/Parameters **CALC(dstVar, src, "FEXISTS");**

Command **CALC - FREAD**

Description **A text file can be read into a text buffer using CALC(dstTxtVar, src, "FREAD");**
The buffer must be bigger than the file size

Syntax/Parameters **CALC(dstTxtVar, src, "FREAD");**

Command **CALC - DIR -** ***directory listing***

Description **Text variable can be populated with filenames in NAND:**

Syntax/Parameters **CALC(dstTxtVar, src, filter, sep, "DIR");**
CALC(dstTxtVar, src, filter, "DIR");
CALC(dstTxtVar, src, "DIR");

dstTxtVar is a text variable and will contain number of files and subdirectories in list followed by specified separator between each filename. src is "nand" (in quotes). sep is separator character for filenames and subdirectories in string (must be in quotes),
if not included then "," assumed

filter is types of files to list, supported filters:
"*/" - list all files and subdirectories
"*.*", "*",

"*.bmp", "*.jpg", "*.png" (or "img" to get all image files)
"*.wav", "*.mp3", "*.wma" (or "snd" to get all sound files)
"*.fnt",
"*.txt",
"*.mnu",
"*.bin",
"*.tft",
"*.log",
"*.ppf",
"*.tri"
If not specified then "*" assumed.
Multiple filters can be included, separated by commas: "*.bmp,*.jpg,*.fnt"

Examples: (nand contains 1.bmp, 2.bmp, x.mnu)

```
    CALC( txtVar, "nand", "*.bmp", ",", "DIR" ); // list all bitmap image files
    > txtVar = "2,1.bmp,2.bmp";
    CALC( txtVar, "nand", "*.fnt", "DIR" ); // list all font files
    > txtVar = "0";
    CALC( txtVar, "nand", "*.mnu", "DIR" ); // list all menu files
    > txtVar = "1,x.mnu";
    CALC( txtVar, "nand", "DIR" ); // list all files
    CALC( txtVar, "nand", "*", "DIR" ); // same
    CALC( txtVar, "nand", "*", ",", "DIR" ); // same
    > txtVar = "3,x.mnu,1.bmp,2.bmp";
```

Command **CALC - FINFO - *Get Info***

Description Get info Function -
Fills dest string with the following :
filename, title, artist, length, samplerate, bitspersample, numchannels

Syntax/Parameters CALC(dest, filename, "FINFO");

Command **CALC -Checksums - "MCHK", "TCHK"**

Description Add a checksum and check for a checksum error

Syntax/Parameters

```
CALC( dst, src, "type", "MCHK" ); // Copy src buf to dst buf, make checksum of "type" and append to dst buf
CALC( res, src, "type", "TCHK" ); // Test checksum of "type" in src buf and set res to 1 if checksums same, else 0.
```

where "type" is:
"SUM8ZA" - Sum all data in src as type U8, checksum is two's complement of the sum, stored as two ASCII hexadecimal characters
(when sum added to checksum is zero, res is 1)
"SUM8ZD" - Sum all data in src as type U8, checksum is two's complement of sum, stored as single U8
(when sum added to checksum is zero, res is 1)
"SUM8A" - Sum all data in src as type U8, checksum is sum, stored as two ASCII hexadecimal characters
(when sum is same as checksum, res is 1)
"SUM8D" - Sum all data in src as type U8, checksum is sum, stored as single U8
(when sum is same as checksum, res is 1)
"XOR8A" - Exclusive-OR (XOR) of all data in src as type U8, checksum is XOR, stored as two ASCII hexadecimal characters
(when XOR of src with checksum is zero, res is 1)
"XOR8D" - Exclusive-OR (XOR) of all data in src as type U8, checksum is XOR, stored as single U8
(when XOR of src with checksum is zero, res is 1)

Example:

```
    LOAD(txData,"1234");
    FUNC(SendData)
    {
    CALC(txData,txData,"SUM8ZA","MCHK"); // Add checksum (txData="123436")
    LOAD(RS2,txData); // Send data
    }

    FUNC(ReceiveData)
    {
    LOAD(rxData,RS2); // Read data
    CALC(res,rxData,"SUM8ZA","TCHK"); // Check for checksum error
    IF(res=1?ProcessData:DataError);
    }
```

Command **CALC - CRC16 - *16 support***

Description * Support for additional CRC-16 algorithms.
> Note CALC(dst32, srcBuf, "", "CRC16"); will use the MODBUS ("modbus") algorithm

CALC() "type"	Poly-nominal	Initial Value	Reflect In	Reflect Out	XOR Out Value	Names and aliases
"arc"	0x8005	0x0000	Yes	Yes	0x0000	"ARC", "CRC-16", "CRC-IBM", "CRC-16/ARC", "CRC-16/LHA"
"kermit"	0x1021	0x0000	Yes	Yes	0x0000	"KERMIT" "CRC-16/CCITT", "CRC-16/CCITT-TRUE", "CRC-CCITT"
"modbus"	0x8005	0xFFFF,	Yes	Yes	0x0000	"MODBUS"
"x-25"	0x1021	0xFFFF	Yes	Yes	0xFFFF	"X-25", "CRC-16/IBM-SDLC", "CRC-16/ISO-HDLC", "CRC-B"
"xmodem"	0x1021	0x0000	No	No	0x0000	"XMODEM", "ZMODEM", "CRC-16/ACORN"
"ccitt-f"	0x1021	0xFFFF	No	No	0x0000	"CRC-16/CCITT-FALSE"
"usb"	0x8005	0xFFFF	Yes	Yes	0xFFFF,	"CRC-16/USB"
"spi"	0x1021	0x1D0F	No	No	0x0000	"CRC-16/SPI-FUJITSU", "CRC-16/AUG-CCITT"
"buypass"	0x8005	0x0000	No	No	0x0000	"CRC-16/BUYPASS", "CRC-16/VERIFONE"
"dds-110"	0x8005	0x800D	No	No	0x0000	"CRC-16/DDS-110"
"dect-r"	0x0589	0x0000	No	No	0x0001	"CRC-16/DECT-R"
"dect-x"	0x0589	0x0000	No	No	0x0000	"CRC-16/DECT-X"
"dnp"	0x3D65	0x0000	Yes	Yes	0xFFFF	"CRC-16/DNP"
"en13757"	0x3D65	0x0000	No	No	0xFFFF	"CRC-16/EN-13757"
"genibus"	0x1021	0xFFFF	No	No	0xFFFF	"CRC-16/GENIBUS", "CRC-16/EPC", "CRC-16/I-CODE", "CRC-16/DARC"
"maxim"	0x8005	0x0000	Yes	Yes	0xFFFF	"CRC-16/MAXIM"
"mcrf4xx"	0x1021	0xFFFF	Yes	Yes	0x0000	"CRC-16/MCRF4XX"
"riello",	0x1021	0xB2AA	Yes	Yes	0x0000	"CRC-16/RIELLO"
"t10-dif",	0x8BB7	0x0000	No	No	0x0000	"CRC-16/T10-DIF"
"teledsk"	0xA097	0x0000	No	No	0x0000	"CRC-16/TELEDISK"
"tms371x"	0x1021	0x89EC	Yes	Yes	0x0000	"CRC-16/TMS37157"
"a"	0x1021	0xC6C6	Yes	Yes	0x0000	"CRC-A"

* Results have been confirmed using the "123456789" test with the values at http://regregex.bbcmicro.net/crc-catalogue.htm

Syntax/Parameters CALC(dst16, srcBuf, "type", "CRC16");

Command **CALC - CRC32 -*32 support***

Description * Support for CRC-32 algorithms
> Note CALC(dst32, srcBuf, "", "CRC32"); will use the CRC-32 ("adcpp") algorithm

CALC() "type"	Poly- nominal	Initial Value	Reflect In	Reflect Out	XOR Out Value	Names and aliases
"adcpp"	0x04C11DB7	0xFFFFFFFF	Yes	Yes	0xFFFFFFFF	"CRC-32", "CRC-32/ADCCP", "PKZIP"
"bzip2"	0x04C11DB7	0xFFFFFFFF	No	No	0xFFFFFFFF	"CRC-32/BZIP2", "CRC-32/AAL5", "CRC-32/DECT-B", "B-CRC-32"
"c"	0x1EDC6F41	0xFFFFFFFF	Yes	Yes	0xFFFFFFFF	"CRC-32C", "CRC-32/ISCSI", "CRC-32/CASTAGNOLI"
"d"	0xA833982B	0xFFFFFFFF	Yes	Yes	0xFFFFFFFF	"CRC-32D"
"mpeg-2"	0x04C11DB7	0xFFFFFFFF	No	No	0x00000000	"CRC-32/MPEG-2"
"posix"	0x04C11DB7	0x00000000	No	No	0xFFFFFFFF	"CRC-32/POSIX", "CKSUM"
"q"	0x814141AB	0x00000000	No	No	0x00000000	"CRC-32Q"
"jamcrc"	0x04C11DB7	0xFFFFFFFF	Yes	Yes	0x00000000	"JAMCRC"
"xfer"	0x000000AF	0x00000000	No	No	0x00000000	"XFER"

* Results have been confirmed using the "123456789" test with the values at http://regregex.bbcmicro.net/crc-catalogue.htm

Syntax/Parameters CALC(dst32, srcBuf, "type", "CRC32");

Command **CALC - MSPLIT** *-User Protocol Split*

Description Perform a multiple split of a buffer to a series of variables.

```
  >>> CALC( dstPtr, srcBuf, char, "MSPLIT" );

- The buffer 'srcBuf' is split at each 'char' and each result is stored in an incrementing series of variables prefixed with the name in 'dstPtr'.
- If 'dstPtr' contains "svar", the first variable will be 'svar0', then 'svar1', 'svar2', ..., 'svar9', 'svar10' etc.
- If a particular svarN is not defined then the result is not stored for that split.
- The data is stored in the format specified in each 'svarN' allowing the buffer to be split into text, unsigned/signed integers and floats.
Example
    VAR( dst > "svar", PTR );
    VAR( svar0, 0, U8 );
    VAR( svar1, 0, S32 );
    VAR( svar2, 0.0, FLT4 );
    VAR( svar3, "", TXT );
    VAR( buf, "123,-67,3.14,Hi", TXT );
    CALC( dst, buf, ",", "MSPLIT" );   // Gives: svar0 = 123, svar1 = -67, svar2 = 3.14, svar3 = "Hi"
  >>> CALC( dstPtrArray, srcBuf, char, "MSPLIT" );

- The buffer 'srcBuf' is split at each 'char' and each result is stored in the variables pointed to by  successive subscripts of the Pointer Array 'dstPtrArray'.
- The first variable will be the variable name stored in 'dstPtrArray.0', then 'dstPtrArray.1', 'dstPtrArray.2',
- If a particular 'dstPtrArray.N' is not a variable or not defined then the result is not stored for that split.
- The data is stored in the format specified in each 'dstPtrArray.N' variable allowing the buffer to be split into text, unsigned/signed integers and floats.
Example
    VAR( arr > "", PTR, 4 );
    VAR( alpha, 0, U8 );        LOAD( arr.0 > "alpha" );
    VAR( bravo, 0, S32 );       LOAD( arr.1 > "bravo" );
    VAR( charlie, 0.0, FLT4 );  LOAD( arr.2 > "charlie" );
    VAR( delta, "", TXT );      LOAD( arr.3 > "delta" );
    VAR( buf, "123,-67,3.14,Hi", TXT );
    CALC( dst, buf, ",", "MSPLIT" );
    // Gives: alpha = 123, bravo = -67, charlie = 3.14, delta = "Hi"

- Using this method with arrays of pointers, testing found the time to split 64 parameters using "MSPLIT"
  was 8ms compared to 64 individual "SPLIT"s which took 30ms.
```

Syntax/Parameters CALC(... "MSPLIT")

Command **CALC - CFLT** - *Calc float to/from U32*

Description Added function to convert a U32 containing a IEEE 754 binary 32 formatted float into a float and vice versa.
Action depends on source and destination types.

```
  > CALC( flt, u32, "CFLT" );  // Converts a float to IEEE 754 binary 32 formatted float
  > CALC( u32, flt, "CFLT" );  // Converts an IEEE 754 binary 32 formatted float to a float
```

Syntax/Parameters VALC(flt, u32, "CFLT");
CALC(u32, flt, "CFLT");

```
Example 1:
    var(f,0,FLT4);
    var(u,0,U32);
    load(f,68.123);
    calc(u,f,"cflt");
    load(rs2,"f=",f,"; u=",%H08%u,"\\0d\\0a");
    > f=68.1230; u=42883EFA

Example 2
    load(u,\\41460000);
    calc(f,u,"cflt");
    load(rs2,"f=",f,"; u=",%H08%u,"\\0d\\0a");
    > f=12.3750; u=41460000
```

Command **CALC - Entity Information**

Description * More Calc commands added to obtain entity information

```
> CALC(var,ename,"ESIZE"); -> returns allocated display size in bytes
> CALC(var,ename,"EDEL"); -> returns 1 if entity has been deleted, else 0
> CALC(var,ename,"EVIS"); -> returns 1 if entity is visible, else 0
> CALC(var,ename,"EALIGN"); -> returns value representing alignment:
0 = Top Left, 1 = Top Centre, 2 = Top Right,
3 = Centre Left, 4 = Centre Centre, 5 = Centre Right,
6 = Bottom Left, 7 = Bottom Centre, 8 = Bottom Right
```

Syntax/Parameters

```
> CALC(var,ename,"ESIZE");
> CALC(var,ename,"EDEL");
> CALC(var,ename,"EVIS"); -
> CALC(var,ename,"EALIGN");
```

Command **Array Sort - CALC(..."ASORT"); and CALC(..."ASORTR");**

Description Added functions to sort arrays

```
CALC(arrname, "ASORT|ASORTR");
CALC(arrname.index, "ASORT|ASORTR");
Examples :-
VAR(arrname, 0, U8, 10);
```

```
LOAD(arrname, 9, 34, 2, 42, 102, 33, 52, 1, 67, 19);
CALC(arrname, "ASORT");
Results :-
arrname.0 = 1
arrname.1 = 2
arrname.2 = 9
arrname.3 = 19
arrname.4 = 33
arrname.5 = 34
arrname.6 = 42
arrname.7 = 52
arrname.8 = 67
arrname.9 = 102

VAR(arrname, 0, U8, 3, 4);
LOAD(arrname.0, 34, 2, 67, 4);
LOAD(arrname.1, 123, 45, 6, 127);
LOAD(arrname.2, 3, 109, 16, 5);
CALC(arrname, "ASORT");
Results :-
arrname.0 = 3, 109, 16, 5
arrname.1 = 34, 2, 67, 4
arrname.2 = 123, 45, 6, 127

VAR(arrname, 0, U8, 3, 4);
LOAD(arrname.0, 34, 2, 67, 4);
LOAD(arrname.1, 123, 45, 6, 127);
LOAD(arrname.2, 3, 109, 16, 5);
CALC(arrname.1, "ASORT");
Results :-
arrname.0 = 34, 2, 67, 4
arrname.1 = 6, 45, 123, 127
arrname.2 = 3, 109, 16, 5
```

Additional Len parameter added to sort only a specified number of elements

Syntax/Parameters

```
CALC(arrname, "ASORT");
CALC(arrname.index, "ASORT");
CALC(arrname, "ASORTR");
CALC(arrname.index, "ASORTR");

CALC(arrname,len, "ASORT");
CALC(arrname.index,len, "ASORT");
CALC(arrname,len, "ASORTR");
CALC(arrname.index,len, "ASORTR");
```

Command	**RUN**
Description	**Run previously defined user code or functions. User code is supplied in C and compiled by our firmware department subject to order. Maximum Function Size increased: 8192 ASCII characters - v49.32.**
Syntax/Parameters	**RUN(Name);**

Options	**Functions can be run as macros for compact menu design.
	RUN(Func1); or RUN(Func1,Func2,Func3...FuncN); or a pointer to a function RUN(func-ptr);
	
	Inline Command Blocks - v30.00
	Inline functions supported - RUN([CmdA(); CmdB(); ... Cmdn();]);
	> RUN([LOAD(RS2, "Hello");]);
	
	RUN(varcmd) - v49.02
	Added support for running commands from a text variable.
	This is useful when sending a SMART command over a serial link embedded in a user protocol.
	It is then possible to dynamically create new entities and pages remotely from a host in a user protocol.
	LOAD(cmd, "LOAD(RS2,1);LOAD(RS2,\\22Hello\\22);"); // cmd is a text variable.Use \\22 to insert double quotes
	RUN(cmd); //Sends 1Hello via RS232 port**

Example	**RUN(UpFunc);
	
	RUN(SaveFunc,ExitFunc);**

Command	**FUNC**
Description	**Create a function called by commands which returns to the next command on completion. Functions can call other functions and themselves. No storing or passing of variables occurs as these are all global even if created in a function. Maximum 28 nested loops or functions - v49.32. Maximum Function Size increased: 8192 ASCII characters - v49.32.**
Syntax/Parameters	**FUNC(Name) {...}**
Options	**You can exit a function by using the EXIT command EXIT(Name) - end functions - v47.24 > EXIT(name); // exit nested loops/functions up to and including loop/function with name**

Example	**> FUNC(fn1) { if(x=5?[EXIT(fn1);]); ...... } // exits function when x=5 without running rest of code
	> FUNC(fn2) { LOOP(lp3,100){ LOAD(RS2,"*"); if(quit=1?[EXIT(fn2);;]);
	// sends 100 *'s through RS2 unless quit is set to 1, then loop and the function are exited (A screen refresh occurs before the exit)
	
	Note, if the name provided in the EXIT(name); command does not exist in the current function/loop nesting, then all loops
	and functions are exited up to the top level. It is not possible to exit the page loop in this way.
	
	Restriction: If processing a function called from a KEY() command then further key presses <u>will be ignored</u>. Each touch key press function must be processed to completion before another can be processed. Please refer to the project example 'keyboard' for the technique to process keys.**

Command **IF**

Description **Compare variables, buffers or text for value or length.**
If true, do function1, if false do function2 (optional).
The ~ operator types can compare text length with another text or a numeric length.
When comparing floating point numbers (max 17 decimal places - v47.12) the lowest bit is masked prior to comparison.

Syntax/Parameters **IF(Var~Var?Function1:Function2)**

Options

The operators allowed for numeric values are:

=, ==	equal to
<>, !=	not equal to
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
+	sum not equal to zero
-	difference not equal to zero
*	multiplication not equal to zero
/	division not equal to zero
%	modulus not equal to zero
&	logical AND
|	logical OR
^	logical exclusive-OR
=-	equal to the negative of
&&	Boolean AND
||	Boolean OR

The operators allowed for text strings are:

=, ==	equal to
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
<>, !=	not equal
~=	same text length
~<	text length shorter than
~>	text length longer than
~!	not same text length

IF() - Multiple tests - v49.32
Support for multiple tests added.

IF(AopB AND CopD OR EopF AND GopH ? funcY : funcN);
"op" is one of the existing operators (=, !=, > etc).
Use AND or OR between pairs of comparisons. Comparisons are performed left to right.

Entity Exists Test - IF(name # 0 ? fncThen : fncElse); - v49.47
*** New operator '#' added to test if entity exists**
*** The result of ‘#’ is 1 for “exists” or 0 for “not exists”.**
*** The result of ‘!#’ is 1 for “not exists” or 0 for “exists”.**

```
IF( entName # 0 ? fncThen : fncElse ); // entName does not exist: do "fncThen" else do "fncElse"
IF( entName # 1 ? fncThen : fncElse ); // entName exists: do "fncThen" else do "fncElse"

IF( entName !# 1 ? fncThen : fncElse ); // entName does not exist: do "fncThen" else do "fncElse"
IF( entName !# 0 ? fncThen : fncElse ); // entName exists: do "fncThen" else do "fncElse"
```

Example:
```
IF(varTest#1?[LOAD(RS2,"Y");]:[LOAD(RS2,"N");]); // outputs N
IF(varTest!#1?[LOAD(RS2,"Y");]:[LOAD(RS2,"N");]); // outputs Y
IF(varTest#0?[LOAD(RS2,"N");]:[LOAD(RS2,"Y");]); // outputs N
IF(varTest!#0?[LOAD(RS2,"N");]:[LOAD(RS2,"Y");]); // outputs Y
VAR(varTest,0,U8);
IF(varTest#1?[LOAD(RS2,"Y");]:[LOAD(RS2,"N");]); // outputs Y
IF(varTest!#1?[LOAD(RS2,"Y");]:[LOAD(RS2,"N");]); // outputs N
IF(varTest#0?[LOAD(RS2,"N");]:[LOAD(RS2,"Y");]); // outputs Y
IF(varTest!#0?[LOAD(RS2,"N");]:[LOAD(RS2,"Y");]); // outputs N
```

Example
```
 IF(K0=“L”?HELPFNC);     //single condition
IF(HIGHVAL < ACTVAL ? HIGHFUNC : LOWFUNC);
IF(STRVAR~>0? SHOWFUNC);   //if STRVAR length > 0 show data
IF(STARVAL >= -STARTMP?SHOWSTAR);
IF(STARVAL > 0? [ LOAD(vReqd,15); TEXT(txtCurFlr,"15"); RUN(fncGo); ] ); //uses in line code [..]
```

Command **SWITCH / SELECT - CASE method**

Description **It is possible to emulate the SELECT CASE or SWITCH CASE function found in other languages.**
This is used to test the contents of a variable and selectively process data according to its value.

The method used in the Itron TFT modules is different since it can directly jump to functions located anywhere in the program provided they use a common naming method.

It makes use of the ability to compile a function name in a variable and then use the RUN(variable); command
The example combines "case_" with "DC" to form a function name "case_DC".
The program can then contain functions to serve all the input options

This removes the need for multiple IF statements.

Example **A typical 'c' example is as follows**

```
switch ( input )
      {
         case "DC":
            DCfunc();
            break;
         case "DCT":
            DCTfunc();
            break;
         case "C":
            Cfunc();
            break;
         case "D":
            Dfunc();
            break;
         default :
            defunc();
      }

public DCfunc() {..................}
public DCTfunc() {.................}
public Cfunc() {..................}
public Dfunc() {..................}
public defunc() {................}
```

The equivalent method is shown below

```
LOAD(chkstr , "," , input , ",");
CALC(tmp, ",C,D,DC,DCT," , chkstr, "FIND");
IF(tmp< 0 ? case_default : [LOAD(chkstr,"case_",input); RUN(chkstr);]);
```

This 3 line technique adds "," to front and end of the input value and loads into chkstr
If input="DC" then chkstr=",DC,"

A CALC command compares the chkstr with a list to identify if the input value exists
The existing commands are defined by ",C,D,DC,DCT,"

If tmp is -1 the input does not exist and the default function "case_default" is RUN.

If tmp is 0+n, the command exists and a prefix is added to input and RUN.
Where input = "DC", it exists so the function name "case_DC" is created and RUN

tmp and chkstr are predefined variables type S8 and TXT.

```
FUNC(case_DC) {.............}
FUNC(case_DCT) {.............}
FUNC(case_C) {.............}
FUNC(case_D) {.............}
FUNC(case_default) { ...........}
```

Command **EXIT**

Description **Command EXIT() to exit functions or loops.**

Syntax/Parameters **EXIT(Name)**

```
> EXIT();
  Exit current loop/function.

> EXIT(name);
  Exit nested loops/functions up to and including loop/function with name.
```

Example

```
> loop(lp1,FOREVER){ calc(x,y,z,"+"); if(x=5?[exit(lp1);]); }       // exit loop when x=5
> loop(lp2,FOREVER){ calc(x,y,z,"+"); if(x=5?[exit(lp2);]); }       // exit loop when x=5 (as above)
> FUNC(fn1) { if(x=5?[EXIT(fn1);]); ...... }                        // exits function when x=5 without running rest of code
> FUNC(fn2) { LOOP(lp3,100){ LOAD(RS2,"*"); if(quit=1?[EXIT(fn2);;]);
  // sends 100 *'s through RS2 unless quit is set to 1, then loop and the function are exited (A screen refresh occurs before the exit)
```

Note, if the name provided in the EXIT(name); command does not exist in the current function/loop nesting, then all loops and functions are exited up to the top level. It is not possible to exit the page loop in this way.

Timing

Command	**Real Time Clock RTC**

Description The real time clock requires a battery to be fitted to the rear of the module or a 3VDC supply applied via a connector fitted to the rear of the PCB. The default format is 14 Sep 2010 09:50:06 which can be modified to suit the application which is achieved by loading the RTC into a variable having the required format. Another method is to use predefined variables of individual RTC values.

Syntax/Parameters SET RTC
The RTC is set using 24 hour time with LOAD(RTC, "YYYY:MM:DD:hh:mm:ss");
with fixed format where:
- YYYY is year 1900-2099
- MM is month 01-12
- DD is day of month 01-31
- hh is hours 00-23
- mm is minutes 00-59
- ss is seconds 00-59

Style READ RTC
You can LOAD the RTC into a variable where the format is specified in a style as follows:

```
STYLE( myRtcStyle, Data )
  {
  type = text;            // Setup a text variable
  length = 64;            // with max length of 64 chars
  format = "jS F Y g:ia"; // RTC format string
  }
```

The RTC date/time can be displayed as a formatted string using special characters

> Day:

d	Day of month with leading zeros	01-31
j	Day of month without leading zeros	1-31
S	Ordinal suffix for day of month	st, nd, rd, th

> Month:

F	Full textual representation of month	January-December
m	Numeric representation of month with leading zeros	01-12
M	Short textual representation of month, three letters	Jan-Dec
n	Numeric representation of month without leading zeros	1-12

> Year:

Y	Full numeric representation of year, 4 digits	1900-2099
y	Two digit representation of year	00-99

> Time:

a	Lowercase Ante meridiem and Post meridiem	am, pm
A	Uppercase Ante meridiem and Post meridiem	AM, PM
g	12-hour format of hour without leading zeros	1-12
G	24-hour format of hour without leading zeros	0-23
h	12-hour format of hour with leading zeros	01-12
H	24-hour format of hour with leading zeros	00-23
i	Minutes with leading zeros	00-59
s	Seconds with leading zeros	00-59

> other characters not in list will be shown as is

Format examples:
"d M Y H:i:s" will display as: 14 Sep 2010 09:50:06 (default format)
"d/m/y" will display as: 14/09/10
"jS F Y g:ia" will display as: 14th September 2010 9:50am

Options Predefined variables below can be read, but not set.

RTCSECS - numeric variable containing seconds (0-59) which can be tested or loaded into a text.
RTCMINS - numeric variable containing minutes (0-59) which can be tested or loaded into a text.
RTCHOURS - numeric variable containing hours (0-23) which can be tested or loaded into a text.
RTCDAYS - numeric variable containing days (1-31) which can be tested or loaded into a text.
RTCMONTHS - numeric variable containing month (1-12) which can be tested or loaded into a text.
RTCYEARS - numeric variable containing year (1900-2099) which can be tested or loaded into a text.

Example Use vars to setup the time in a user page

```
VAR(years,2010,U16);
VAR(months,11,U8);
VAR(days,2,U8);
VAR(hours,10,U8);
VAR(mins,30,U8);
```

User changes the vars via buttons then a SAVE button would load the RTC

```
LOAD(RTC,years,":",months,":",days,":",hours,":",mins,":00");

VAR( RtcVar, "", myRtcStyle );  // Create a var to store formatted string
LOAD( RtcVar, RTC );        // Grab the formatted RTC time and date
TEXT( Txt1, RtcVar );;      // Show the formatted time on display in Txt1 and refresh screen
LOAD( RS2, RtcVar );      // Send formatted time on RS232 port
```

Command	**RTC Day Of Week**

Description Added day of week support to RTC.

Options Built in variable RTCWEEKDAY reports day of week where 1=Monday, 2=Tuesday,... 7=Sunday.
Formatting parameters added for RTC
D Short textual representation of day, three letters: Mon-Sun
L Full textual representation of the day of the week: Monday-Sunday
N ISO-8601 numeric representation of the day of the week: 1 (for Monday) - 7 (for Sunday)
Note RTC day of week is indeterminate if RTC has not been set.
The RTC Alarm does not support day of week.
For an alarm that triggers every Thursday at 16:00, the following example can be used:

Example

```
INT( RTA, fnc_Alarm );
LOAD( RTA, ":::16:00:00" );
FUNC( fnc_Alarm )
  {
   IF( RTCWEEKDAY != 4 ? [ EXIT( fnc_Alarm ); ] );
     // Do Thursday alarm code here...
  }
```

Command	**Real Time Clock Alarm (RTA)**

Description Support for an RTC Alarm is provided using RTA. This can be set for duration, time or time and date.
You can set an alarm for every minute, at 17.45 every day or on the 15th March at 12.52 each year.
To setup the interrupt which is triggered at the alarm point:

To load the alarm time, use same format as setting RTC.
Only populated values are used to set the alarm, therefore alarms can be set to go off every minute, hour, hour:minute:second, day or month etc..
Note, the alarm does not support the years parameter, and is ignored when setting the alarm.

Syntax/Parameters INT(name,RTA,function);

Style Read RTA

```
STYLE( myRtaStyle, Data )
  {
  type = text;            // Setup a text variable
  length = 64;            // with max length of 64 chars
  format = "jS F Y g:ia"; // RTC format string
  }
```

Options Settings can be read by accessing the built in variables
RTAMONTHS, RTADAYS, RTAHOURS, RTAMINS, RTASECS
If a value has not been set then -1 is returned.

Example Setting the alarm:

```
        LOAD(RTA,":5:26:14:7:03"); // Alarm will occur every year on 26th May at 14:07:03
        LOAD(RTA,":::13:15:");    // Alarm will occur every day at 13:15:00
        LOAD(RTA,":::",hours,":",mins,":",secs); // Alarm will occur every day at hours:mins:secs
        LOAD(RTA,":::::20"); // Alarm will occur every 20 seconds past the minute.
```

To clear alarm:

```
 LOAD(RTA,0); // Clear Alarm
 LOAD(RTA,":::::"); // Clear Alarm

VAR( RtaVar, "", myRtaStyle );   // Create a var to store formatted string
LOAD( RtaVar, RTA );        // Grab the formatted RTC time and date
TEXT( Txt1, RtaVar );;      // Show the formatted time on display in Txt1 and refresh screen
LOAD( RS2, RtaVar );      // Send formatted time on RS232 port
```

Command	**I/O Counter**
Description	The 31 I/O counters use pre-define variables which can be reset and tested for value. The counter uses an unsigned 32bit register (U32) with names CNTKxx where xx=00 to 30. They require the I/O to be set as an interrupt but do not require an associated INT() command. Counter increment depends on the rising or falling edge of the interrupt. The command RESET(CNTK00) resets to zero the I/O counter on K00. The maximum counter speed is 0-10kHz+ and is dependent on other interrupt and entity usage.
Syntax/Parameters	CNTK00-CNTK30

Example	IF(CNTK00>300?Func300); //if greater than 300 run function called Func300
	TEXT(K00Text,CNTK00);; //update counter value on page and refresh screen

Command	**RunTime Counter**
Description	The RUNTIME counter uses pre-define variables which can be set and tested for values The command Reset(RUNTIME) sets all vales to zero and starts the timer. This runtime counter is independent of the real time clock and runs continually so no setup is required.
Syntax/Parameters	CNTMILLI - Increments every millisecond 0-999 CNTSECS - Increments every second 0-59 CNTMINS - Increments every minute 0-59 CNTHOURS - Increments every hour 0-23 CNTDAYS - Increments every 24 hours CNTRUN - Increments every millisecond since system reset. 86,400,000 = 1 day - v47.12.

Options	**RunTime Counter Interrupts**
	Wrap-around interrupt for the RunTime counter have been added.
	INT(name,CNTMILLI,function); // function called every 1000ms
	INT(name,CNTSECS,function); // function called every 60s
	INT(name,CNTMINS,function); // function called every 60mins
	INT(name,CNTHOURS,function); // function called every 24hours
	INT(name,CNTDAYS,function); // function called every 2^32days
	For timer resolutions of less than a second, use TIMER0 to TIMER9.

Example	IF(CNTMINS>30?FuncHalfHour); //if greater than 30 minutes run function called FuncHalfHour
	TEXT(MinsText,CNTMINS);; //update counter value on page and refresh screen

Command	**TIMER**
Description	Twenty (20) count-down timers with 1ms resolution - TIMER0 to TIMER19 - v49.25
Syntax/Parameters	TIMER0 - TIMER19

Options	To setup the interrupts:
	INT(name,TIMER0,function); to INT(name,TIMER19,function);
	
	To read the remain time before expiry
	LOAD(var,TIMER0);
	
	To run the timer once
	LOAD(TIMER0,time); // time is in milliseconds
	
	To run the timer multiple times
	LOAD(TIMER0,time,num); // time is in milliseconds
	num is number of times timer runs, 1 = 1 time, 2 = 2 times etc, 0 = non-stop
	
	To clear the timer
	LOAD(TIMER0,0);

Example	LOAD(TIMER6,1000); // TIMER6 runs once and expires after one second
	LOAD(TIMER9,1000,0); // TIMER9 runs forever, expiring every second
	LOAD(TIMER14,500,5); // TIMER14 runs five times, expiring every 500ms
	LOAD(TIMER3,0); // Clear TIMER3
	LOAD(TIMER17,time); // TIMER17 runs once and expires after value in var time

Command	**WAIT**
Description	Wait for a period of milliseconds before processing menu commands. Wait timer accuracy of 1ms ±200ns - v42.00. Interrupts and key presses still occur during the wait period and can be processed. WAIT() can be used with INTs, VARs or PTR - v36.00. Restriction: If the WAIT() command is within a function called from a KEY() command then further key presses <u>will be ignored</u>. Each touch key press function must be processed to completion before another can be processed. Please refer to the project example 'keyboard' for the technique to process keys.
Syntax/Parameters	WAIT(Time);

Example	WAIT(5000);
	
	WAIT(TimeV1);

Command **Loop**

Description **Repeats the specified actions a number of times in a PAGE then continue. Max 28 nested loops or functions - v49.32. The value for Var1 can be a number from 1-65000 or the text FOREVER. You can exit a LOOP using the command EXIT(Name); Loops can be nested and used in PAGES or FUNCTIONS.**

Syntax/Parameters **LOOP(Name,Var1){...........}**

Options
EXIT(Name) - end loops - v47.24
```
> EXIT(name); // exit nested loops up to and including loop with name
```
Examples:
```
> LOOP(lp1,FOREVER){ CALC(x,y,z,"+"); IF(x=5?[EXIT(lp1);]); } // exit loop when x=5
```
Note, if the name provided in the EXIT(name); command does not exist in the current loop nesting, then all loops and functions are exited up to the top level. It is not possible to exit the page loop in this way.

Precautions when using LOOP() including "Array Error - Subscript Out Of Range" message
At the start of each pass through a loop, a check is performed to see if a touch screen key is being pressed and, if it is, then the associated touch key function is called. Caution must be observed with the touch key function to not modify variables that are being used within the loop otherwise undesired results can occur which can be difficult to spot or result in an error message.

Good coding practice
 > Make sure variables used in a loop are not modified from a touch key function (unless this is a desired action)
 > If a variable does need to be changed then set a 'flag' in the key function and test the flag in the page loop and make the change there instead.

Example
```
LOOP(MyLoop,12){SHOW(Page1);WAIT(100);SHOW(page2);WAIT(100);}  //repeat 12 times
LOOP(MyLoop,FOREVER) {SHOW(Page1);WAIT(100);SHOW(page2);WAIT(100);}
```

Loop Example 1
```
   FUNC(fn1)
     {
     VAR(ii,0,U8);
     VAR(jj,0,U8);
     VAR(kk,0,U8);
     LOOP(lp0,10)
       {
       LOOP(lp1,10)
         {
         LOOP(lp2,10)
           {
           LOAD(RS2,ii,",",jj,",",kk,"\\0d");
           CALC(kk,kk,1,"+");
           }
         CALC(jj,jj,1,"+");
         }
      CALC(ii,ii,1,"+");
      }
    LOAD(RS2,"\\0d");
    }
     RS2 Outputs: 0,0,0\\0d0,0,1\\0d\\0,0,2\\0d...9,9,9\\0d<\\0d
```

Loop Example 2
```
    KEY(k0,[LOOP(klp,10){LOAD(RS2,"*");}LOAD(rs2,"\\0d\\0a");],480,136,TOUCH);
    RS2 outputs on key press: **********\\0d\\0a
```

Restriction: If the LOOP() command is within a function called from a KEY() command then further key presses <u>will be ignored</u>.
Each touch key press function must be processed to completion before another can be processed.
Please refer to the project example 'keyboard' for the technique to process keys.

Example 1 - Variables:
```
  VAR( varX, 0, U8 );
  // We have a simple function...
  FUNC( fnTest1 )
  {
    LOAD( varX, 0 );
    LOOP( lpTest1, 10 )
    {
      // [Touch Keys are effectively tested here]
      LOAD( RS2, varX );
      CALC( varX, varX, 1, "+" );
    }
  }
  // In a page we have...
  KEY( kyTest1, [ LOAD( varX, 0 ); ], 100, 100, TOUCH );
  // Normally we would get 0123456789 sent out of the RS2 port each time fnTest1 is run
  // If however the key kyTest1 is pressed when the loop is being run then the output may be changed to 0123012345!
```

Example 2 - Arrays:
```
  VAR( varArr, 0, U8, 5 );
  VAR( varY, 0, U8 );
  // We have another simple function...
  FUNC( fnTest2 )
  {
    LOAD( varY, 0 );
    LOOP( lpTest2, 5 )
    {
     // [Touch Keys are effectively tested here]
      LOAD( RS2, varArr.varY );
      CALC( varY, varY, 1, "+" );
    }
  }
  // In a page we have...
  KEY( kyTest2, [ LOAD( varY, 0 ); ], 100, 100, TOUCH );
  KEY( kyTest3, [ LOAD( varY, 3 ); ], 100, 100, TOUCH );

  // Normally we would get the contents of varArr.0 varArr.1 varArr.2 varArr.3 varArr.4 sent out of the RS2 port each time fnTest2 is run
  // If however the key kyTest2 is pressed when the loop is being run then the output may be changed to varArr.0 varArr.1 varArr.0 varArr.1 varArr.2!
  // Or, an error when kyTest3 is pressed giving varArr.0 varArr.1 varArr.3 varArr.4 ** Array Error - Subscript Out Of Range ** (ie varArr.5 doesn't
exist!)
```

Protocols

Command **MODBUS**

Description **Modbus RTU is available on RS2, RS4, AS1, AS2 - v49.32.**
More than one of these can be active at the same time if required.

Syntax/Parameters

```
SETUP(RS2)
{
baud = 19200;
parity = N;
rxi = Y;
txi = Y;
proto = ModbusRTU;   //define modbus as the protocol to use
}
```

Options **Modbus protocol for CRC calculation and response events -v49.03.**
Added MODBUS protocol to handle CRC calculation and response events.

All low level handling of Modbus requests and responses are handled by the TFT firmware and the 'interface' to the itronSMART code is via system variables (xxx refers to the interface name: RS2, RS4, AS1 or AS2):

Name	Type	Use
MB_xxx_MODE	U8	Modbus mode (0=master, 1=slave)
MB_xxx_REGS	PTR	Pointer to a U16 array to hold read / write registers
MB_xxx_COILS	PTR	Pointer to a U8 array to hold read / write coil bit values
MB_xxx_SLAVEID	U8	Slave address
MB_xxx_FUNC	U8	Modbus function code (1/3/15/16)
MB_xxx_REGOFF	U16	Offset into U16 register array for read / write action
MB_xxx_REGCNT	U16	Number of registers to read / write (or have been read / written)
MB_xxx_COILOFF	U16	Offset into U16 coil register array for read / write action
MB_xxx_COILCNT	U16	Number of coil registers to read / write (or have been read / written)
MB_xxx_STATUS	U8	Modbus status (0=idle, 1=tx, 1=rx, 2+ error)
MB_xxx_TIMEOUT	U16	Response timeout in milliseconds
MB_xxx_DATAOK	PTR	Pointer to function that is called after a successful transaction
MB_xxx_DATAERR	PTR	Pointer to function that is called after an error has occurred

COIL data is converted from bits to bytes to make itronSMART access easier. For example a Modbus request to read 12 coils (2 Modbus data bytes) with offset of 6 will result in bytes 6 – 17 in the U8 coil registers being used.

Master operation
Write regs / coils
Load values into U16 / U8 variable that MB_xxx_REGS/MB_xxx_COILS points to.
Setup MB_xxx_REGCNT/MB_xxx_COILCNT and MB_xxx_REGOFF/MB_xxx_COILOFF and use command LOAD(RS2, 1); to start request.
MB_xxx_DATAOK/MB_xxx_DATAERR function will be called after the success response is received from the slave.
Read regs / coils
Setup MB_xxx_REGCNT/MB_xxx_COILCNT and MB_xxx_REGOFF/MB_xxx_COILOFF and use command LOAD(RS2, 1); to start request.
MB_xxx_DATAOK/MB_xxx_DATAERR function will be called after the success response is received from the slave.
Values can then be read from variable that MB_xxx_REGS/MB_xxx_COILS points to.

Slave operation
Write regs / coils
MB_xxx_DATAOK/MB_xxx_DATAERR function will be called after the successful request from the master.
Values can then be read from variable that MB_xxx_REGS/MB_xxx_COILS points to.
MB_xxx_REGCNT/MB_xxx_COILCNT and MB_xxx_REGOFF/MB_xxx_COILOFF values should be read to know which registers were updated.
Read regs / coils
MB_xxx_DATAOK/MB_xxx_DATAERR function will be called after the successful request from the master.
MB_xxx_REGCNT/MB_xxx_COILCNT and MB_xxx_REGOFF/MB_COILOFF values can be read to know which registers were read.

Example

Master

```
// New modbus test
lib(fnt, "SDHC/asc_32.fnt");
style(ps, page){back = black;}
style(ts, text){font = fnt; col = white; currel = TL;}
style(ts_cc, text){font = fnt; col = white; currel = CC;}

setup(RS2)
{
baud = 19200;
parity = N;
encode = sd;
txi = Y;
rxi = Y;
proto = ModbusRTU;
}

var(regs, 0, U16, 128);

load(MB_RS2_MODE, 0);
load(MB_RS2_REGS > "regs");
load(MB_RS2_SLAVEID, 1);
load(MB_RS2_TIMEOUT, 1000);

var(v1, 10, U16);
var(v2, 30, U16);
var(v3, 80, U16);
var(v4, 0, U16);
var(v5, 0, U16);
var(v6, 0, U16);

int(tim0, TIMER0, clear_status);

page(p, ps)
{
posn(100, 40); text(t1, "WR", ts);
posn(+0, +35); text(t2, "0", ts);
posn(+0, +30); text(t3, "1", ts);
posn(+0, +30); text(t4, "2", ts);

posn(130, 75); text(t5, v1, ts);
posn(+0, +30); text(t6, v2, ts);
posn(+0, +30); text(t7, v3, ts);

posn(340, 40); text(t8, "RD", ts);
posn(+0, +35); text(t9, "0", ts);
posn(+0, +30); text(t10, "1", ts);
posn(+0, +30); text(t11, "2", ts);

posn(370, 75); text(t12, "", ts);
posn(+0, +30); text(t13, "", ts);
posn(+0, +30); text(t14, "", ts);

posn(120, 136); key(k1, write, 240, 272, TOUCH);
posn(360, 136); key(k2, read, 240, 272, TOUCH);
```

Slave

```
// New modbus test - slave

lib(fnt, "SDHC/asc_32.fnt");

style(ps, page){back = black; }
style(ts, text){font = fnt; col = white; currel = TL;}

setup(RS2)
{
baud = 19200;
parity = N;
encode = sd;
txi = Y;
rxi = Y;
proto = ModbusRTU;
}

var(regs, 0, U16, 128);

load(MB_RS2_MODE, 1);
load(MB_RS2_REGS > "regs");
load(MB_RS2_SLAVEID, 1);
load(MB_RS2_DATAOK > "ok");
load(MB_RS2_DATAERR > "error");

load(regs.0, 10);
load(regs.1, 9102);
load(regs.2, 234);

int(tim0, TIMER0, clear_status);

page(p, ps)
{
posn(240, 50); text(t1, regs.0, ts);
posn(240, 90); text(t2, regs.1, ts);
posn(240, 130); text(t3, regs.2, ts);
posn(240, 256); text(status, "", ts);
}

func(ok)
{
if(MB_RS2_FUNC==16?[text(status, "WRITE OK");;run(update_values);]:
[text(status, "READ OK");;]);
load(TIMER0, 1500, 1);
}

func(error)
{
if(MB_RS2_FUNC==16?[text(status, "WRITE ERROR");;]:[text(status,
"READ ERROR");;]);
load(TIMER0, 1500, 1);
}

func(update_values)
{
```

```
posn(240, 256); text(status, "", ts_cc);
posn(460, 256); text(status_error, "", ts_cc);
}

func(read)
{
load(MB_RS2_FUNC, 3);
load(MB_RS2_REGOFF, 0);
load(MB_RS2_REGCNT, 3);
load(MB_RS2_DATAOK > "read_ok");
load(MB_RS2_DATAERR > "read_error");
load(RS2, 1);
}

func(read_ok)
{
load(v4, regs.0); text(t12, v4);
load(v5, regs.1); text(t13, v5);
load(v6, regs.2); text(t14, v6);;
text(status, "READ OK");;
load(TIMER0, 1500, 1);
}

func(read_error)
{
text(status, "READ ERROR");
text(status_error, MB_RS2_STATUS);;
load(TIMER0, 1500, 1);
}

func(write)
{
load(MB_RS2_FUNC, 16);
load(MB_RS2_REGOFF, 0);
load(MB_RS2_REGCNT, 3);
load(regs.0, v1);
load(regs.1, v2);
load(regs.2, v3);
load(MB_RS2_DATAOK > "write_ok");
load(MB_RS2_DATAERR > "write_error");
load(RS2, 1);
}

func(write_ok)
{
text(status, "WRITE OK");;
load(TIMER0, 1500, 1);
}

func(write_error)
{
text(status, "WRITE ERROR");
text(status_error, MB_RS2_STATUS);;
load(TIMER0, 1500, 1);
}

func(clear_status)
{
text(status, "");;
text(status_error, "");;
}

show(p);
```

```
text(t1, regs.0);
text(t2, regs.1);
text(t3, regs.2);;
}

func(clear_status)
{
text(status, "");;
}

show(p);
```

Command **CANBUS**

Description When attaching a CANBUS adaptor type EMCBK33 to CN3 using a 10 way IDC cable, the connector is fitted to the backside of the module and the following set up is required to match the default settings in the adaptor.

Syntax/Parameters

```
setup(AS1)
{
baud=38400;          //num = 110 to 115200. Any value can be set to allow trimming for deviating clocks i.e. 38450
data=8;              //num = 5, 6, 7, 8
stop=1;              //num = 1, 15, 2  - note 15 is 1.5 bits
parity=N;            //first letter of Odd, Even, None, Mark, Space
rxi=C;               //set receive buffer interface as active (Y), a command processing source (C) or disable (N). Default = N
encode=sr;           //set s=ASCII, w=UNICODE, m=UTF8 or use sr, wr and mr specifying raw data bytes.
flow=H;              //none, hardware RTS/CTS or DTR/DSR, software XON XOFF
}
```

Options The default receive address for the adaptor is ID=155h with 11bit or 29bitID
packets accepted (2.0a or 2.0b spec)
All bytes are received on AS1 with 1 to 8 bytes of data.
The transmit ID is also 155H. with data sent via AS1 with data length of 1

Example

```
setup(AS1)
{
baud=38400;          //num = 110 to 115200. Any value can be set to allow trimming for deviating clocks i.e. 38450
data=8;              //num = 5, 6, 7, 8
stop=1;              //num = 1, 15, 2  - note 15 is 1.5 bits
parity=N;            //first letter of Odd, Even, None, Mark, Space
rxi=C;               //set receive buffer interface as active (Y), a command processing source (C) or disable (N). Default = N
encode=sr;           //set s=ASCII, w=UNICODE, m=UTF8 or use sr, wr and mr specifying raw data bytes.
flow=H;              //none, hardware RTS/CTS or DTR/DSR, software XON XOFF
}
```

Command **DMX512**

Description Added new DMX512 protocol for use with RS4 interface

Syntax/Parameters

```
SETUP(RS4)
{
proto = DMX512;
}
```

Options User must setup a data array (used to hold the DMX512 slot data) and assign it to the built in system pointer variable DMX512_DATA, ie :-
VAR(dat, 0, U8, 513);
LOAD(DMX512_DATA > "dat");
Then to start DMX512 output :-

LOAD(RS4.txi, Y);
DMX512 data packets are then continuously sent from the TFT module. Slot data can then be manipulated by writing to the data array, ie :-
LOAD(dat.4, 128); Array index 0 is the DMX512 'Start Code'.
The DMX512 specification supports shorter packets - in this implementation the packet length = array length.

Example

```
SETUP(RS4)
{
proto = DMX512;
}

VAR(dat, 0, U8, 513);
LOAD(DMX512_DATA > "dat");

LOAD(RS4.txi, Y);
LOAD(dat.4, 128); Array index 0 is the DMX512 'Start Code'.
```

Topics

Command **Fonts**

Description

You can include the character fonts required for an application by downloading the attached files and use the LIB command to store them in memory. You can setup your system to process text as single byte, 2 byte UNICODE or multibyte UTF8. See the LIB command for installing fonts.
System fonts ASCII8,ASCII16 and ASCII32 are built in. The wide rounded fonts are preferred for higher quality designs.
Default font for text style is ASCII16 - v47.12.

It is possible to overlay one font over another to enable single byte operation with ASCII from 20H to 7FH and Cyrillic, Greek, Hebrew, Bengali, Tamil, Thai or Katakana from 80H to FFH. The LIB command is used to load the extended font at 0080H instead of it's normal UNICODE location. The style for a text can then specify font="MyASCII,MyThai"; causing the Thai to overlap the ASCII from 80H to FFH

Example:
```
LIB( ascii24,”sdhc/asc_24.fnt”);                        //upload ascii 24 pixel wide font
LIB( cur24,”sdhc/cur_24.fnt?start=\\0080”);        //upload currency font to 80H
```

In text style...
```
font=”ascii24,cur24”;      //cur24 overlays ascii24 at 80H-8FH
```

STANDARD ASCII - 20H to 7FH
Standard ASCII text in the range 20H to 7FH can by directly typed from the keyboard.
System fonts named ASCII8, ASCII16, ASCII32 are pre-installed.
```
Example   TEXT( txt1, "Hello World", stTXT );                 //single byte access to 20H to 7FH ASCII characters
```

EXTENDED ASCII - 20H to FFH
2/ When using single byte ASCII in the range 20H to 7FH, you can access extended characters from 80H to FFH using hex code like \\AB
```
Example   TEXT( txt1, "1. AB\\B0CDEF \\AB s", stTXT );    //single byte access to 80H to FFH
```

UNICODE and UTF8
3/ When using single byte ASCII in the range 20H to 7FH, you can access UNICODE characters by using hex code like \\w0D7F
or a UTF8 character using hex code like \\mC2AB. The symbols <....> are used where more than one character is coded.
Examples:
```
      TEXT( txt2, "2. AB\\w00B0CDEF \\w00AB", stTXT );                    // UNICODE double byte access to 0080H to FFFFH
      TEXT( txt3, "3. AB\\mC2B0CDEF \\mC2AB", stTXT );                   // UTF8 multi byte access to 80H to FFFFH
      TEXT( txt5, "5. AB\\sB0C\\w<00440045>F \\w00AB", stTXT );    // <....> are used for long hex strings \\s is used for single byte in a UNICODE or
UTF8 encoded system
      TEXT( txt7, "\\<372E204142B04344454620AB>", stTXT );          //  string of single byte hex in the range 20H to 80H
      TEXT( txt8, "\\w<0038002e00200041004200B00043004400450046002000AB>", stTXT );
      TEXT( txt9, "\\m<392E204142C2B04344454620C2AB>", stTXT );
```

Syntax/Parameters

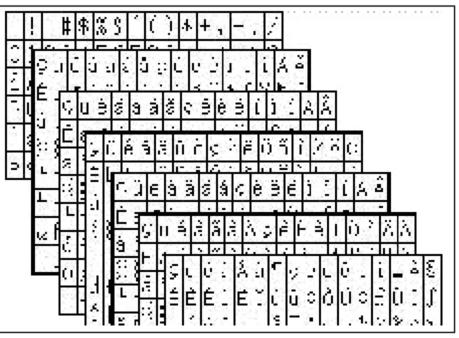
Style
```
STYLE(Txt32ASC16,TEXT)
{
font="ASC16B,16THAI";     //define fonts using built in or preloaded .FNT files via LIB command
size=2;                          //a 24x24 font is expanded to a 48x48 font. default=1
col=white;                        //“\\000000”  to “\\FFFFFF” or reserved words from the colour chart.
maxLen=64;                    //maximum length of text. default =32, maximum=512
maxRows=4;                     //maximum number of rows=32 where new line code \\0D\\0A is used.
rotate=90;                    //rotation relative to screen 0, 90, 180, 270. default=0
curRel=CC;                  //specify placement relative to cursor.
}
```

Options

Compact Narrow Fonts

(Single Byte Range 20H to FFH or UNICODE Range 0020H to 00FFH)
The ASCII base page is included automatically at 20H-7FH and the other fonts are automatically loaded to 80H to FFH.
This gives a single byte range of 20H to FFH.

Name	Characters	Standard Sizes in Pixels
ASCII base Page	96	5x7, 8x16, 16x32
PC437 (USA - European Standard)	128	5x7, 8x16, 16x32
PC850 (Multilingual)	128	5x7, 8x16, 16x32
PC852 (Latin 2)	128	5x7, 8x16, 16x32
PC858 (Multilingual)	128	5x7, 8x16, 16x32
PC860 (Portuguese)	128	5x7, 8x16, 16x32
PC863 (Canadian French)	128	5x7, 8x16, 16x32
PC865 (Nordic)	128	5x7, 8x16, 16x32
PC866 (Cyrilic)	128	5x7, 8x16, 16x32
WPC1252	128	5x7, 8x16, 16x32
Katakana	128	5x7, 8x16, 16x32



Wide Rounded Fonts

(Single Byte Range 20H to FFH or UNICODE Range 0020H to FFFFH)
When loading these fonts into library, it is necessary to specify the offset address for the first character of each
font table if a variation from UNICODE is required. The supplementary characters above FFFF are not supported in UTF8.

Name	Characters	Unicode	Standard Sizes in Pixels*
ASCII + European	467	0021-0217	16, 24, 32, 40, 48, 60, 72
Cyrillic	226	0401-04F9	16, 24, 32
Greek	105	0374-03F3	16, 24, 32
Arabic	194	060C-06F9	16, 24, 32
Hebrew	82	0591-05F4	16, 24, 32
Bengali	89	0981-09FA	16, 24, 32
Tamil	61	0B82-0BF2	16, 24, 32
Thai	87	0E01-0E5B	16, 24, 32
Chinese/Japanese/Korean	211151	3300-9FA5	16, 24, 32
Hangul	11172	AC00-D7A3	16, 24, 32
Katakana	94	30A1-30FE	16, 24, 32

* Custom size fonts are also available upon request

ASCII + European
Cyrillic
Greek
Arabic
Hebrew
Bengali
Tamil
Thai
Chinese/Japanese/Korean
Hangul
Katakana

Example
```
LIB( MyFont24, "SDHC/Font24.fnt" );
LIB( fntAsc16x16, "SDHC/asc16B.fnt?start=\\0020" );
```

Command **Colour Chart**

Description **The colour chart below shows the built in colours of the TFT module. To clarify the reference name of a colour, hover over the hex code.**
You can use either the colour name or its hex code when defining colours in styles

Colours of "none" or "transparent" - v49.37
*** The value of "none" or "transparent" can now be used for all colours in the styles.**

Options

#4682B4 steelblue	#041690 royalblue	#6495ED cornflowerblue	#B0C4DE lightsteelblue	#7B68EE mediumslateblue	#6A5ACD slateblue	#483D8B darkslateblue	#191970 midnightblue	#000080 navy	#00008B darkblue
#0000CD mediumblue	#0000FF blue	#1E90FF dodgerblue	#00BFFF deepskyblue	#87CEFA lightskyblue	#87CEEB skyblue	#ADD8E6 lightblue	#B0E0E6 powderblue	#F0FFFF azure	#E0FFFF lightcyan
#AFEEEE paleturquoise	#48D1CC mediumturquoise	#20B2AA lightseagreen	#008B8B darkcyan	#008080 teal	#5F9EA0 cadetblue	#00CED1 darkturquoise	#00FFFF aqua	#00FFFF cyan	#40E0D0 turquoise
#7FFFD4 aquamarine	#66CDAA mediumaquamarine	#8FBC8F darkseagreen	#3CB371 mediumseagreen	#2E8B57 seagreen	#006400 darkgreen	#008000 green	#228B22 forestgreen	#32CD32 limegreen	#00FF00 lime
#7FFF00 chartreuse	#7CFC00 lawngreen	#ADFF2F greenyellow	#9ACD32 yellowgreen	#98FB98 palegreen	#90EE90 lightgreen	#00FF7F springgreen	#00FA9A mediumspringgreen	#556B2F darkolivegreen	#6B8E23 olivedrab
#808000 olive	#BDB76B darkkhaki	#B8860B darkgoldenrod	#DAA520 goldenrod	#FFD700 gold	#FFFF00 yellow	#F0E68C khaki	#EEE8AA palegoldenrod	#FFEBCD blanchedalmond	#FFE4B5 moccasin
#F5DEB3 wheat	#FFDEAD navajowhite	#DEB887 burlywood	#D2B48C tan	#BC8F8F rosybrown	#A0522D sienna	#8B4513 saddlebrown	#D2691E chocolate	#CD853F peru	#F4A460 sandybrown
#8B0000 darkred	#800000 maroon	#A52A2A brown	#B22222 firebrick	#CD5C5C indianred	#F08080 lightcoral	#FA8072 salmon	#E9967A darksalmon	#FFA07A lightsalmon	#FF7F50 coral
#FF6347 tomato	#FF8C00 darkorange	#FFA500 orange	#FF4500 orangered	#DC143C crimson	#FF0000 red	#FF1493 deeppink	#FF00FF fuchsia	#FF00FF magenta	#FF69B4 hotpink
#FFB6C1 lightpink	#FFC0CB pink	#DB7093 palevioletred	#C71585 mediumvioletred	#800080 purple	#8B008B darkmagenta	#9370DB mediumpurple	#8A2BE2 blueviolet	#4B0082 indigo	#9400D3 darkviolet
#9932CC darkorchid	#BA55D3 mediumorchid	#DA70D6 orchid	#EE82EE violet	#DDA0DD plum	#D8BFD8 thistle	#E6E6FA lavender	#F8F8FF ghostwhite	#F0F8FF aliceblue	#F5FFFA mintcream
#F0FFF0 honeydew	#FAFAD2 lightgoldenrodyellow	#FFFACD lemonchiffon	#FFF8DC cornsilk	#FFFFE0 lightyellow	#FFFFF0 ivory	#FFFAF0 floralwhite	#FAF0E6 linen	#FDF5E6 oldlace	#FAEBD7 antiquewhite
#FFE4C4 bisque	#FFDAB9 peachpuff	#FFEFD5 papayawhip	#F5F5DC beige	#FFF5EE seashell	#FFF0F5 lavenderblush	#FFE4E1 mistyrose	#FFFAFA snow	#FFFFFF white	#F5F5F5 whitesmoke
#DCDCDC gainsboro	#D3D3D3 lightgrey	#C0C0C0 silver	#A9A9A9 darkgray	#808080 gray	#778899 lightslategray	#708090 slategray	#696969 dimgray	#2F4F4F darkslategray	#000000 black

Example
```
col=\\FFEFD5;
col=coral;
LIB(MainImg,"SDHC/HomeImg.bmp?back=\\FFF8DC");
LIB(MainImg,"SDHC/HomeImg.bmp?back=gold");
```

Command **Inline Functions**

Description **The commands which require a function as a parameter ie IF, RUN, INT and KEY can have the function code embedded inside the commands by enclosing the required code in square brackets.**
This allows you to reduce the number of lines of code for simple functions and where the function is unlikely to be used elsewhere.

Syntax/Parameters **[cmd(..); cmd(..);.......cmd(..);]**

Options
```
> IF( VarA op VarB ? [ CmdYA(); CmdYB(); ... CmdYn(); ] : [ CmdNA(); CmdNB(); ... CmdNn(); ] );

> RUN( [ CmdA(); CmdB(); ... Cmdn(); ] );

> INT( Name, Buf, [ CmdA(); CmdB(); ... Cmdn(); ] );

> KEY( Name, [ CmdA(); CmdB(); ... Cmdn(); ], X, Y, Style );
```

Example
```
Without inline:
KEY(keyFlr15,floor15fnc,104,84,TOUCH);   //calls function floor15fnc
FUNC(floor15fnc)
  {
  LOAD(vReqd,15);      TEXT(txtCurFlr,"15");      RUN(fncGo);
  }

With inline:
KEY(keyFlr15, [ LOAD(vReqd,15); TEXT(txtCurFlr,"15"); RUN(fncGo); ],104,84,TOUCH);
Examples:
IF(VarA >= 50 ? [CALC(VarA,VarA,5,"-");TEXT(TxtA,VarA);;] : [LOAD(RS2,"VarA=",VarA);SHOW(PageN);] );
KEY(keyX,[CALC(varX,varX,1,"+");],123,12,stKey);
```

Command **DEL**

Description **Delete a Page, Group, Entity, Variable or Buffer from SDRAM.**
If visible on the display, it will remain until the page is refreshed. If the name refers to an image, font or file stored in the flash library then this is set for memory to be freed using RESET(DELETED);

Syntax/Parameters **DEL(...)**

Options **Delete Single Entities**
Delete Multiple Entities

DEL(EEPROM); - Erase EEPROM without restore - v49.37
*** User EEPROM can be erased (without restoring variables) using DEL(EEPROM);**
*** RESET(EEPROM); erases EEPROM and restores any EEPROM variables currently in use.**

Example
```
DEL(Entity1); delete single entity
DEL(Entity1, Entity2, Entity3...); multiple delete entities
DEL(EEPROM) Deletes eeprom variables
```