**ESP32 picoc C Language Interpreter**

The sketch at<https://github.com/rlunglh/IOT_with_your_PC/blob/master/ESP32Program.zip> implements a C Language Interpreter on the ESP32 using the Arduino IDE to create and upload the sketch to your ESP32. There are many good

explanations of setting up the Arduino IDE for the ESP32; a good place to look for this information is at

<https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/>

Once the Arduino IDE is installed just unzip the ESP32Program.zip file into your Arduino folder. You need to edit and upload some files to the ESP32 for the sketch to run. First look at the WIFIname.data file in the ESP32Program/data/data folder on your PC. This file supplies the SSID for the ESP32 to use when connecting to WiFi as a station, so change its name to match your WiFi environment. You also need to edit the ESP32Program/data/data/WIFIpass.dat file to hold your WiFi network password for the ESP32 to use when connecting. After editing these files, you can use the Tool/ESP32 sketch data upload menu in the Arduino IDE item to upload the data folder to your ESP32. To install the ESP32 upload tool follow the instructions at <https://github.com/me-no-dev/arduino-esp32fs-plugin> You may want to make some changes to the ESP32Program file in the Arduino IDE prior to building and uploading your sketch to your ESP32. The first few lines of the sketch are shown below:

#define ALWAYS\_STATION

// define ALWAYS\_STATION to restart EPS32 if it can't connect to the specified SSID

// when defined it will not enter AP mode on a Station Connect

// failure. It will restart until it successfully connects

// to the SSID specified in /data/WIFIname.dat using the

// password specified in /data/WIFIpass.dat

//#define TFT

// define TFT to enable use of an attached 320x240 TFT Display

//#define OLED

// define OLED to enable use of an attached 128x64 OLED Display

//#define SSD1306OLED

// define SSD1306OLED if using the Wemos ESP32 WROOM with OLED

//#define BME280

// defining BME 280 will include support for BME280 sensor

//#define NEO\_PIXEL

// defining NEO\_PIXEL includes Adafruit\_NeoPixel support functions;

//#define ePAPER

// defining ePAPER includes 1.54 in ePaper Display from WaveShare

As shown in the code comments ALWAYS\_STATION is defined to assure that the ESP32 will not enter Access Point mode. If a station connect fails the ESP32 will repeatedly restart and try to connect to the specified SSID until connection is

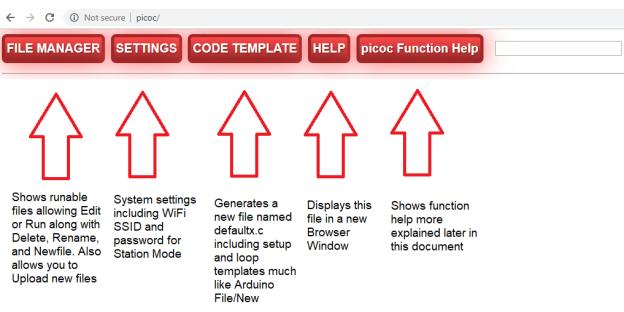
successful. If you do have a problem connecting, make sure that the WIFIname.dat and WIFIpass.data files in

ESP32Program/data/data on your PC have the correct information for your environment. If you want to use the esp32 in AP mode, just put a blank into the WIFIname.data file and comment out the #define ALWAYS\_STATION line.

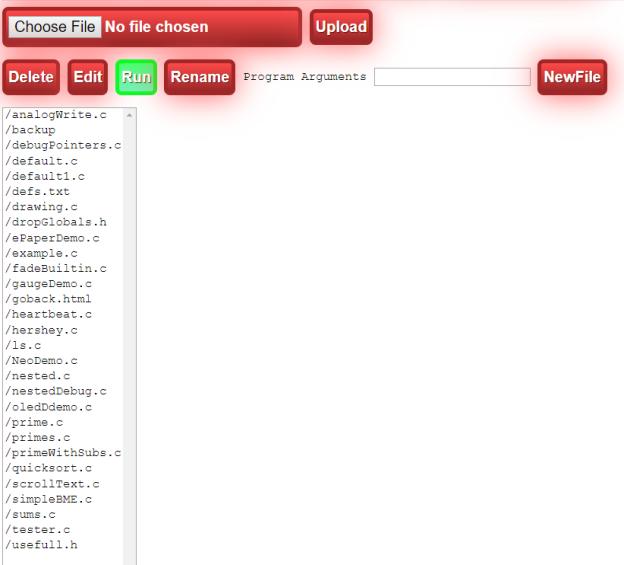
He ESP32PEogram supports an SPI connected TFT color graphic display or a 128x64 OLED display. These are enabled by uncommenting the //#define TFT . //#define OLED, and //#define ePAPER lines. You can use ePAPER, OLED, and TFT

defines concurrently and get a sketch that will work with the defined displays. Once you have the setting needed in the sketch, build and Upload for your sketch to the ESP32. When running, the Serial Monitor (at 1000000 baud) will show the IP address where the picoc Interpreter is running when the ESP32 restarts. In my environment I have added the line <192.168.0.21> picoc to my ..\Windows\System32\drivers\etc\hosts file to allow me to use picocas an alias for the

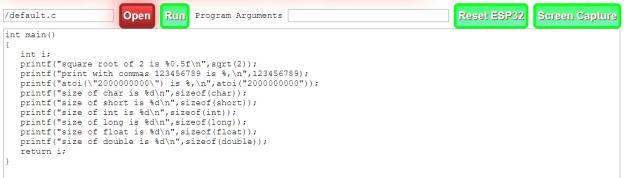
server. In a browser, navigate to the ESP32 address to see the first display. An example browser view is show below:



The following paragraphs give a quick tour of operation. Click the File Manager button giving the display shown below:

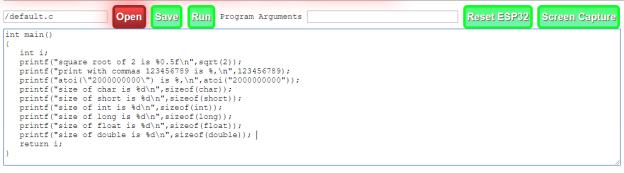


Double Click the /default.c line in the select giving:



This shows the content of the /default.c file and allows editing the content and saving changes using the Save Button.

The Save button will not appear unless you make a change in the edit textarea. When a Save is required, the display will appear as shown in the following figure.



You can also change the edited file by changing the filename in the first text box and clicking Open. If the Save button is available, you can change the filename in the text box and the file will be saved to the named entered. Filenames must start with / and can include additional / to breakout files into different directories and you are free to organize the files anyway you choose. The File Manager will list files showing each file’s full path. The file manager only shows program

files and does not show .jpg .ico or .dat files. The ls() function in the Interpreter will show all files along with their file sizes.

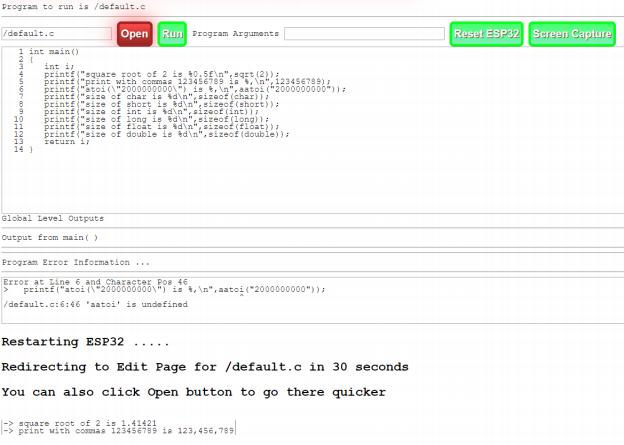
This example is done just to show a simple program’s operation. Click the Run button giving the output shown in the following figure:



When programs are run, you’ll be shown what file is running and any arguments passed along with a listing of the file, Global outputs, the program output, the return value, and the amount of free memory available on the ESP32 when the program completes. This example was chosen to let you see that int and long types are equivalent as both are 32 bit

values, and that the float and double types are equivalent as both are 64 bit values in the Interpreter.

If a running program has a syntax error, you will be shown the Program Error Information. As an example, click the Edit button and change the 2nd occurrence of atoi to aatoi and click the Save button. Now when you click Run you’ll see the display shown below:



The Program Error Information section shows the error, along with the line and character position where the interpreter reached an error condition. It also shows the error cause, in this case, ‘aatoi’ is undefined. This particular error is well

diagnosed and you can verify that clicking Open and changing aatoi to atoi, Saving the file, and clicking Run causes the program to again run without errors.

The handlers around the picoc interpreter eliminate any Interpreter error cleanup problems by simply restarting the

ESP32, then, after a delay of 30 seconds, you will be returned to the Edit page for program you are running. You can also click the Open button on the Run page to get back to the editor more quickly when you quickly recognize the error cause and are ready to make program changes to fix the problem.

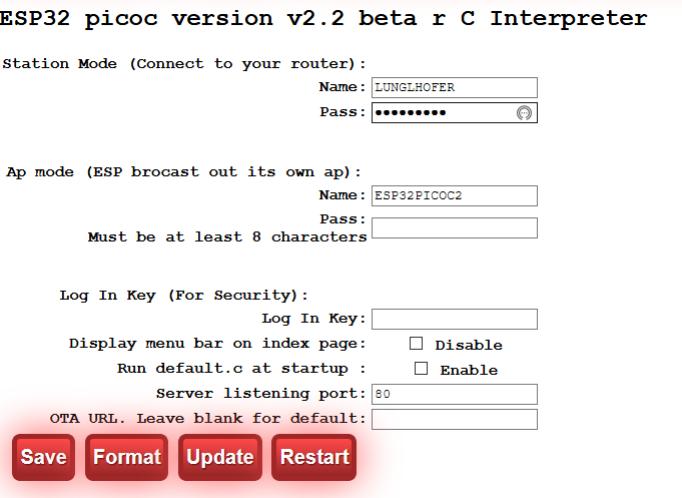
Behind the screens, the interpreter collects all globally defines functions and variables and automatically deletes them after a program run so the program can be run repeatedly without having the interpreter complain that things such as main are already defined. This is accomplished by recording global scope object creation in the /DropGlobals.h file – when youran the /default.c program, its contents were updated to what is shown below

drop("main");

drop(" exit\_value");

This file is processed by the interpreter at program completion to cleanup globals to make the next run.

The SETTIGNS button on the Admin tool bar gives you the Settings dialog shown in the following figure.



Important fields are the Station Mode Name and Pass fields if you want to run in Station mode these need to be set correctly for your environment. You can assure this, without the chicken and egg problem, by editing the

data/data/WIFIname.dat and WIFIpass.dat files and doing a Tools/ESP32 Sketch Data Upload menu operation in the Arduino IDE. Tou may want to also Run default.c at startup by clicking its Enable box towards the bottom of the page. Be sure to click the Save button after any changes are made.

**Sample Programs without External Displays**

There are sample programs included in the sketch data directory, and hence uploaded to the ESP32 using Tool/ESP32 Sketch Data Upload menu item ( make sure the Serial Monitor is closed before doing the upload ) that will let you

explore picoc C Language Interpreter basics.

The programs

● AnalogWrite.c

● debugPointers.c

● default.c

● example.c

● fadeBuiltin.c

● heartbeat.c

● ls.c

● nested.c

● nestedDebug.c

● prime.c

● Primes.c

● primeWithSubs.c

● quicksort.c

● sums.c

● tester.c

Can all be run without needing an external TFT, OLED, orePaper display. Each program has a short synopsis in the following paragraphs.

analogWrite.c – this program brightens and dims the built in LED on the ESP32, The program is set to use pin 2,if you use an ESP module other than the DH-ET line MiniKit, or the DOIT ESP32 DEVKIT that I have tested with, you may need to change the pin being used. For instance, on a HELTEC WiFi Kit 32 you need to use pin 25 which is accomplished by changing the line int LED=2; to int LED=25;

debugPointers.c – an example program showing how the interpreter presents pointer reference data in the Debug mode.

default.c – you’ve already been shown this program’s operation in the initial ESP32Program introduction. example.c – shows some Debug output for nested loops.

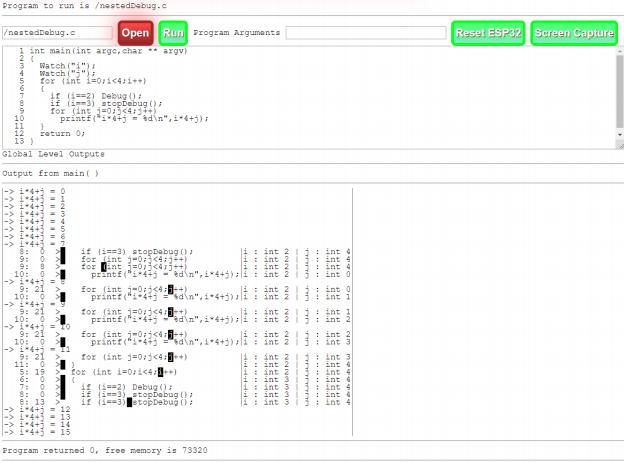
fadeBuiltin.c – example program showing the loop() function in uninterrupted use. You can stop the loop by clicking the ResetESP32 button. This will restart the ESP32 and return you to the Edit view of the file being run

heartbeat.c – simply blinks the built in LED in a heartbeat fashion, again you may have to change the pin number for your specific board.

ls.c – lists the SPIFFS filesystem showing all filenames and sizes.

nested.c – shows the operation of nested for loops in the interpreter and is really present to set the stage for nestedDebug.c

nestedDebug.c - This program introduces the interpreter’s debug features. Its output when run is shown in the following figure.



You see the program listing as in the program examples shown for default.c. Lines 3 and 4 set watches for variables i and j. Line 7 initiates debug operation when i is equal to 2. Line 8 stops debug operation when i is equal to 3. The interpreter gives you a conditional debugging feature since the Debug() functions can be called with any combination of conditional logic you’d like. The Output from main() section shows a tabular format. Outputs from the program ( here caused by

printf functions ) are entered on a separate line preceded by ‘-> ‘ . The source lines being executed begin to show after

the Debug() function is called. Line 7 of the program causes debug operation to start when i is 2. The display then begins showing debug trace output with a line#: col# > leader and the line being executed next shown with an inverse character at the position where the parser willscan next. Also shown in this table is a second column that presents any Watch

variables. At the first trace line 8,i and jare shown to be 2 and 4 respectively. The next line shows that the for j

statement is about to be executed. The 9:8 line shows that the for evaluation will next execute the intj=0 assignment. Line 10:0 shows that the printf is the next statement to be executed and that j is now 0. After this the program output form the printf function is printed as i\*4+j = 8 . The for loop trace then continues until we get to 8:13 when the

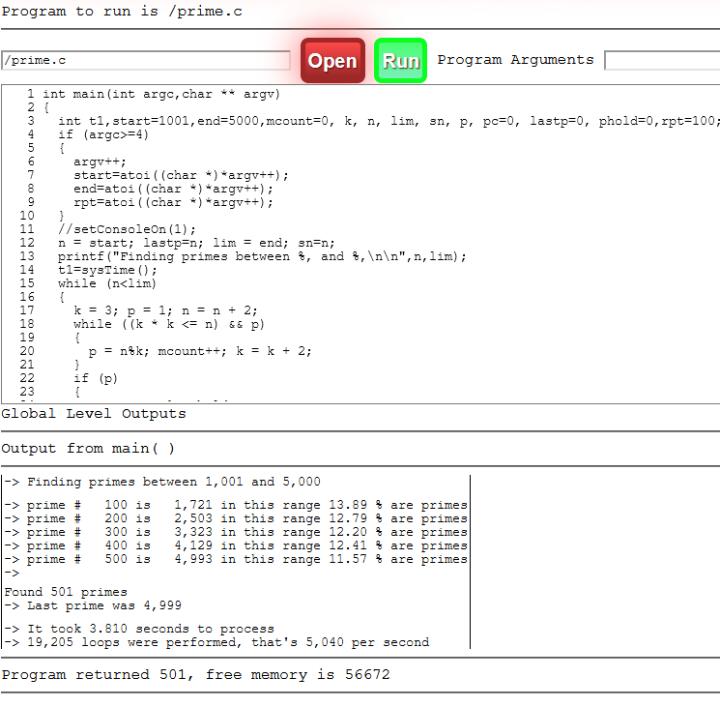
stopDebug() function is executed and tracing stops. After that the remaining program outputs are printed up to i\*4+j = 15 and the program returns the value 0.

These combinations of Debug, stopDebug, Watch, and stopWatch function, along with conditional execution of these statements, give you great flexibility in examining program execution and working out problems in C language code.

The prime.c, primes.cand primeWithSubs.call explore computationally demanding application for the ESP32. These programs are all implementations of a prime number sieve that checks for integer divisors of a number with no

remainder and if no even divisors are found the number is added to the prime number count. Also prime numbers found

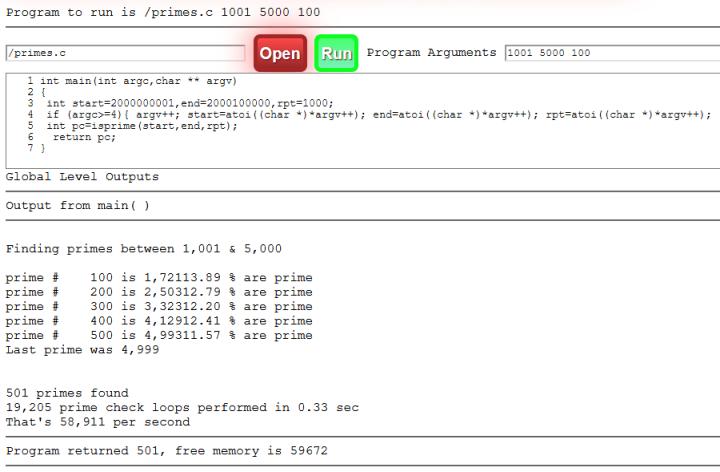
can be output for each n primes specified as range. Let’s look at the output of prime.cas provided in the stock ESP32Program data directory.



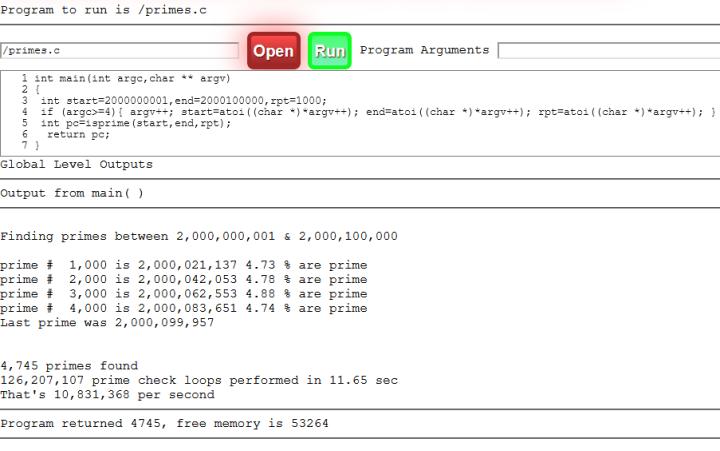
As you can see above, it lists the prime found after each 100 discoveries, and processes over a range of 1001 to 5000.

The program also times its execution and counts how many while ( (k\*k<=n && p) loops are performed to allow

reporting the loops completed per second, in this case, 5,040 per second. I acknowledge that the interpreter completes 5,040 loops per second including a multiplication, two compares, a modulo divide, an increment and an add but, to say the least, this is not exceptional performance. The primes.c program illustrates how this can be improved by placing the while loop into compiled code that is called by the interpreter. Primes.c Run output is shown in the next figure.



This shows that the compiled code ran over 10 times faster. We can realize event greater speed improvements if we run over larger numbers as shown below.



Here we ran the prime number sieve over much larger numbers, to get more while loop iterations and showed results for every 1000 primes found. The startling thing is that when the Interpreter calls compiled code to do the

computationally demanding parts of the program, the while loops soar to close to 11 million per second which is over a 21,000 times improvement. The compiler shouldn’t get all the credit because this interpreter is particularly slow – but it does fit nicely on the ESP32.

ESP32Program can give you some very rewarding experiences if you are new to C programming – just put code that you

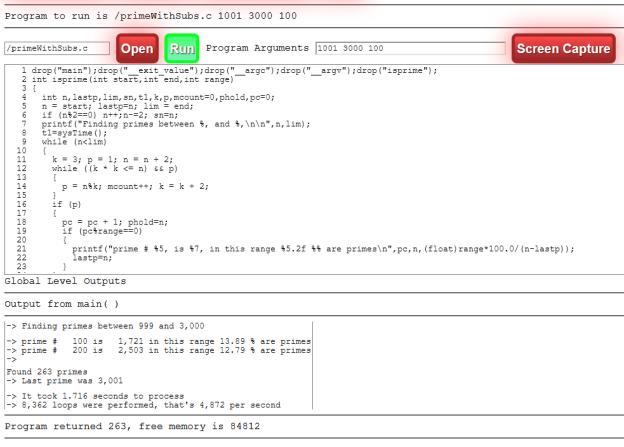
want to verify into a file on the ESP32 and test away without the Arduino compile and upload delays, and with a

conditional trace and watch facility. If you have a section of sketch code that is not operating as you expected or hoped – just paste the appropriate portion of the code into an ESP2 file and run the Debug and Watch functions to see what is really happening without reverting to myriad Serial.print(f) functions (not to mention recompiling and uploading for

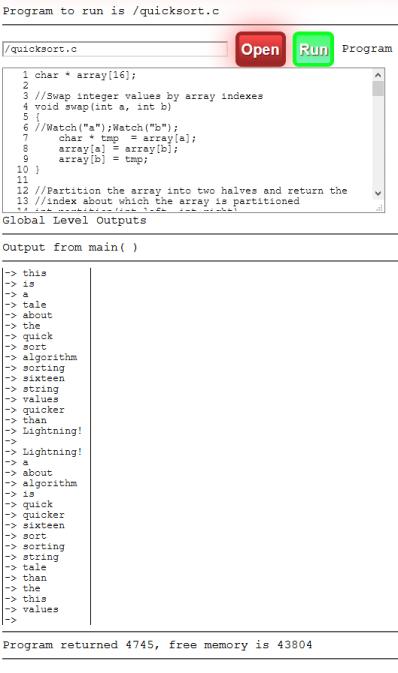
each Serial.print.. statement) .

The primesWithSubs.c program illustrates using a subroutine to perform the while ((k\*k<n) && p) loop.

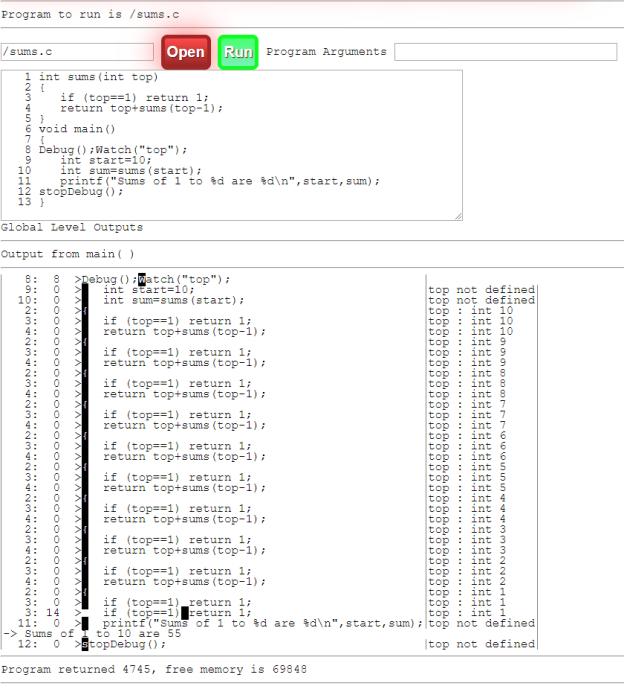
All three of these programs can be run using program arguments entered, logically enough in the Program Arguments text box. A run of the primeWithSubs.c program with program arguments is shown in the following figure.



Quicksort.c – quicksort is a classic sorting algorithm that includes recursion and is included in the samples to illustrate a more complex program logic task being run on the interpreter. Its output is shown in the following figure, first it lists the String array to be sorted followed by the sort results.



Sums.c – is another recursion algorithm that is easier to follow since it is a small piece of code. The output of a sums.c run with Debug and Watch is shown in the following figure.



In the interpreter output, you can clearly see the value of top that is being passed to the sums function decreases from

10 to 1 and that at a value of 1 all the sums are returned and the total of Sums 1 to 10 is 55. Tester.c – a simple program that illustrates the ls() and Cat() function outputs.

**Sample Programs Using External Displays**

The programs

● drawing.c

● example.c

● hershey.c

● scrollText.c

● tester.c

Are used with a 320x240 TFT display which is my favorite way of running the ESP32.

If you use the MH-ET LIVE Minikit for ESP32 you get a Wemos mini D1 compatible footprint that can be used with the LOLIN TFT 2.4 Touch Shield making a simple, ready to use, ESP2 with aTFT color graphic display.

These programs are summarized in the following paragraphs.

When using a 320x240 TFT, the ESP32 will display the following after reset.



This display is a harmless advertisement for Micro Image Systems and the book *Building Blocks for IOT with your PC and WiFi Peripherals.* It also shows that the ESP32 is running in station mode with an IO address to <192.168.0.24> – your IP

address will likely be different. If the ESP32 is running in AP mode it will show the display shown in the following figure.



When in AP mode the Network Name will be ESP32PICOC but you can use the Settings page available from the Admin toolbar to change the name.

drawing.c – Produces some graphics drawings and then goes through a demo of the Hershey fonts built into the

Interpreter by the author along with a slideshow of images that are loaded on the ESP32. The drawing program calls screenCapture after each major display change and this will capture the TFT display images using the Processing

platform and the Processing sketch ScreenCapture.pde available from the following URL

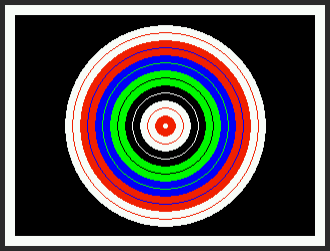
<https://github.com/rlunglh/IOT_with_your_PC/blob/master/ScreenCapture.pde>You will need the Processing

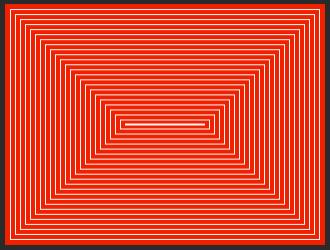
environment to run the ScreenCapture.pdefile and this can be downloaded and installed using the instructions at

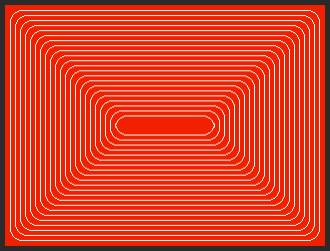
<https://processing.org/tutorials/gettingstarted/>The supplied ScreenCapture.pde sets the correct baud rate and is set to use your second enumerated COMM port. If you look at Device Manager and see only one device under Ports then you will need to change the Processing sketch line " int serial port = 1;"to " int serial port = 0;"The Processing ScreenCapture sketch will not run if the Arduino IDE Serial Monitor is running for your ESP32. Also, when

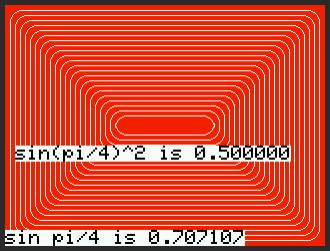
ScreenCapture initially starts, it resets the ESP32. Therefor just get the ScreenCapture.pde running and then proceed

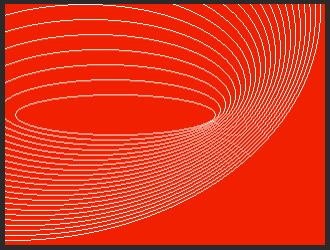
with getting you programs running in the Interpreter. The Screen Capture button on the ESP32Proram web pages will trigger a TFT Screen Capture and the function screenCapture() will trigger a capture if placed within your programs. The following figures are the screen captures obtained automatically when running the drawing.c program.

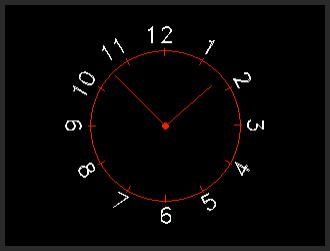


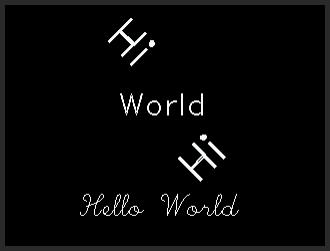








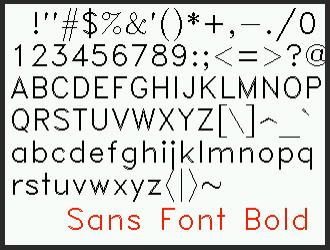




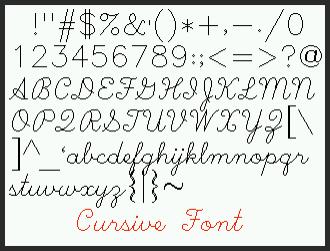


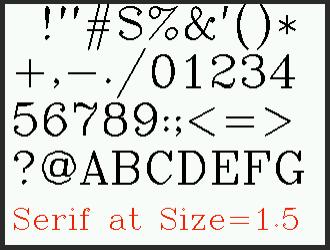


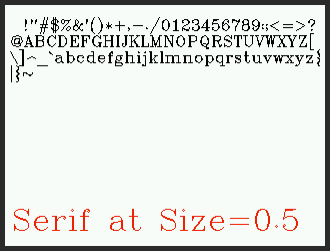


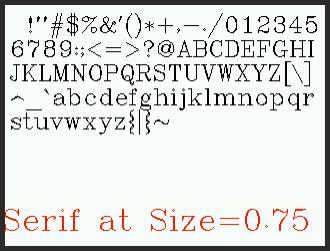




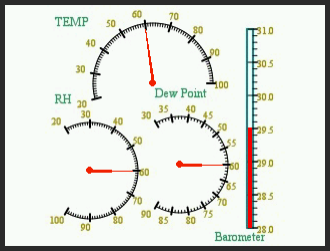






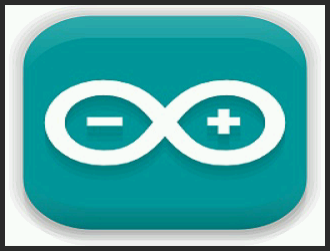


















Eventhough all the displays presented by drawing.c have been shown in the preceding figures, you should watch it run if

you have a TFT display because there are slow stroke drawings of the Cursive font and That’s All Folks that are demonstrative regarding the drawing of the Hershey fonts.

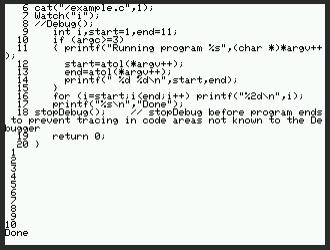
This sample shows that you can display jpeg images on the TFT, draw any number of shapes and fills, along with Hershey Font text of varying sizes and colors, along with native TFT fonts in different fonts, sizes, and colors. These capabilities

open a world of expressive peripherals that can communicate status, show sensor outputs, and provide a compelling

display for your projects. With the Touchscreen capability you can also make projects that have an interactive interface at the Touch Screen display.

example.c – was shown earlier without using the TFT display. This time uncomment the TFTsetTextSize(1) ,

setConsoleOn(2) and comment out the Debug related lines in the program, Save it, and click Run giving the cat function program outputs with console like output on the TFT display. A screen capture following program completion is shown in the following figure.



hershey.c – this program illustrates most of the Hershey font built in functions and shows how to Invert the display for visual effects.

|  |
| --- |
| Years ago (1977), while working at IBM, I was introduced to a set of font definitions using vectors produced by Dr. Allen Vincent Hershey at the[Naval Weapons Laboratory](https://en.wikipedia.org/wiki/Naval_Weapons_Laboratory)in 1967. Hence the name Hershey Fonts. These font definitions are quite well done and include serif and sanserif fonts along with Greek and cursive fonts that are at least entertaining. |

The Hershey functions are

void HFsetCursor(int,int); x,y

void HFdraw(char,float,int); char,size,color

void HFdrawRotated(int,int,char,float,int,float); xc,yx,char,size,color,angle

void HFdrawStringRotated(int,int,float,int,float,char \*); x,y,radius,size,color,angle,char string int HFgetStringSize(char \*,float); char string,size -- returns length

void HFdrawString(int,int,float,int,char \*); x,y,size,color,char string

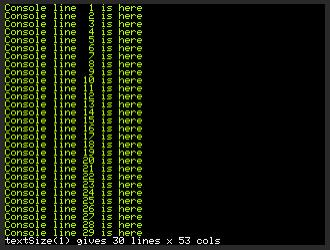
void HFdrawCenteredString(int,float,int,char \*); y,size,color,char string

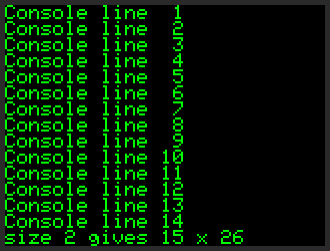
void HFdrawStringOpaque(int,int,float,int,int,char \*); x,y,size,foreColor,backColor,char string void HFsetFont(char \*); font name -- allowed values are serif sans sansbold greek cursive

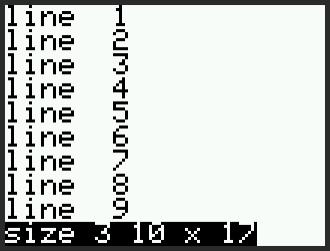
void HFsetStrokeDelay(int); set delay between strokes normally 0, nonzero make for character drawing animation onscreen

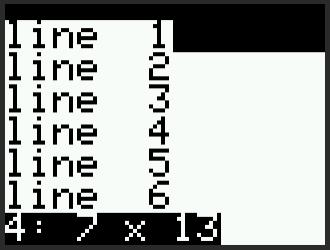
These provides for normal and opaque symbol strokes along with automatic centering and rotation.

scrollText.c – this program does what the name implies and illustrates the scrolling of console output directed to the TFT display. A screenCapture of the TFT screens is shown in the following figures for various text sizes and colors.



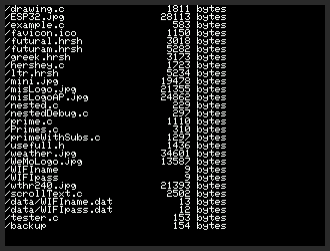








tester.c – this program simply runs the cat() function. If you uncomment the line "//setConsoleOn(1);TFTsetTextSize(1); " add screenCapture(); after the LS(); line, and then Save and Run you get the following output.





**Pulse Width Modulation and AnalogWrite**

If you are going to be using the ESP32 in the Arduino environment it makes sense to have the analog, digital, and PWM capabilities of the Arduino IDE also available in the Interpreter. The interpreter includes the following functions with

arguments that are integers unless otherwise indicated.

● void pwmSetup(pin,frequency,range); - set ESP32 pin to PWM output at frequency with values from 0 to range

● void servoAngle(pin,angle); set pint0 PWM corresponding to float angle. Angle ranges from -90 - +90

● void pwmServo(pin,duty); set pinto float duty dutycycle, duty ranges from 0.0 to 1.0

● void analogWrite(pin,value); set pinto value – user must account for the pins range set in pwmSetup to get the value they really want. Generally the analogWrite function takes values of 0 to 4095.

● void digitalWrite(pin,value);

● int digitalRead(pin);

● void pinMode(pin,char \* mode); set pinto mode "INPUT" "OUTPUT" or "INPUTPULLUP"

The analogWrite program makes use of the setServoReport and analogWrite functions.

**Interactive Help**

You can also get function references directly from the ESP32Program webpages. The “ picoc Function Help” button along with the textbox to the right of the button are your gateway to function definitions. Say that you want to see all TFT\_

functions, just put TFT-\* in the text box and click the Picoc Function Help button. This will give you the following results.

Help for function TFT\_\*

void TFT\_draw(char \*,int,int); filename,x,y -- must be a Jpeg file void TFT\_drawCircle(int,int,int,int); xc,yc,radius,color

void TFT\_drawEllipse(short,short,short,short,short); xc,yc,radiusx,radiusy,color void TFT\_drawLine(int,int,int,int,int); x1,y1,x2,y2,color

void TFT\_drawRect(int,int,int,int,int); x,y,width,height,color

void TFT\_drawRoundRect(int,int,int,int,int,int); x,y,width,height,radius,color void TFT\_drawTriangle(int,int,int,int,int,int,int); x1,y1,x2,y2,x3,y3,color

void TFT\_fillCircle(int,int,int,int); xc,yc,radius,color

void TFT\_fillEllipse(short,short,short,short,short); xc,yc,radiusx,radiusy,color void TFT\_fillTriangle(int,int,int,int,int,int,int); x1,y1,x2,y2,x3,y3,color

void TFT\_fillRect(int,int,int,int,int); x,y,width,height,color

void TFT\_fillRoundRect(int,int,int,int,int,int); x,y,width,height,radius,color void TFT\_fillScreen(int); color

int TFT\_Gauge(float xc,float yc,float sang,float eang,float radius,floatsval,float eval,int divisions,float increments,int color,char \* fmt,char \* valueFmt);

void TFT\_Gauge\_draw(int); gauge#

void TFT\_Gauge\_drawDanger(int,float,float,int); gauge#,startValue,endValue, color

void TFT\_Gauge\_drawDangerByValue(int,float,float,int); gsuge#,startValue,endValue, color void TFT\_Gauge\_dropGauges(); frees all TFT\_Gauge storage

void TFT\_Gauge\_setPosition(int,float); gauge#,value

int TFT\_HbarGraph(float x,float y,int width,intheight,floatsval,float eval,int divisions,float increments,int color,char \* fmt,char \* valueFmt

void TFT\_HbarGraph\_draw(int); gauge#

void TFT\_HbarGraph\_dropGauges(); frees all HbarGraph gauge storage

void TFT\_HbarGraph\_setPosition(int,float); gauge#,value

void TFT\_invertDisplay(int);, pass 0 for normal 1 for inverse void TFT\_print(char \*); char string to print

void TFT\_pushRect(int,int,int,int,short \*); startx,starty,width,height void TFT\_readRect(int,int,int,int,short \*); startx,starty,width,height void TFT\_setCursor(int,int); x,yy increases from 0 at top of display void TFT\_setTextColor(int,int); foreColor,backColor

void TFT\_setTextSize(int); size

int TFT\_VbarGraph(float x,float y,int width,intheight,floatsval,float eval,int divisions,float increments,int color,char \* fmt,char \* valueFmt

void TFT\_VbarGraph\_draw(int); gauge#

void TFT\_VbarGraph\_dropGauges(); frees all VbarGraph gauge storage void TFT\_VbarGraph\_setPosition(int,float); gauge#,value

You can enter \* to get all functions - displays the defs.txt file, a full name, an empty string – to get the defs.txt file

shown in a window with navigation, or as done above a string followed by \* to do a wildcard search. To return to your previous page just use the Go Back button.

**Miscellaneous Buttons on the File Manager Page**

The Buttons on The File Manager page appear as shown in the following figure.



The top button, Browse … allows you to select a file to be uploaded to the ESP32. Once you click the Browse … button and select a file to upload, the Upload button will be made visible. As shown below.



When you click Upload the selected file will be uploaded to the uploads folder on the ESP32 SPIFFS and you will be

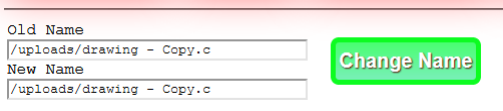
directed to an edit page for the uploaded file. **NOTE *Only text type files can be uploaded in this manner.*** If you want to add other file types to the ESP32 SPIFFS, put them in the ESP32Program’s data folder and use the ESP32 Sketch Data

Upload Tool. Be advised that any files in the SPIFFS that are not in the data directory, or that you have changed since the last upload, will be lost. I carefully copy and paste between the browser window and the PC’s editor when I change a file on the ESP32 to prevent loosing changes if I use the Arduino ESP32 Sketch Data Upload function.

As mentioned above, another means of transferring content from your PC to the ESP3 2 is to use the Edit file and use copy and paste between a file viewer/editor on your PC (my favorite is Notepad++) and the browser Edit page.

The Delete button does exactly what you’d expect it deletes the selected file in the file list from the ESP32 SPIFFS. The Edit and Run buttons have already been discussed and their operation is intuitive.

Therename button lets you rename/move files on the ESP32 SPIFFFS. Its dialog is shown in the following figure.

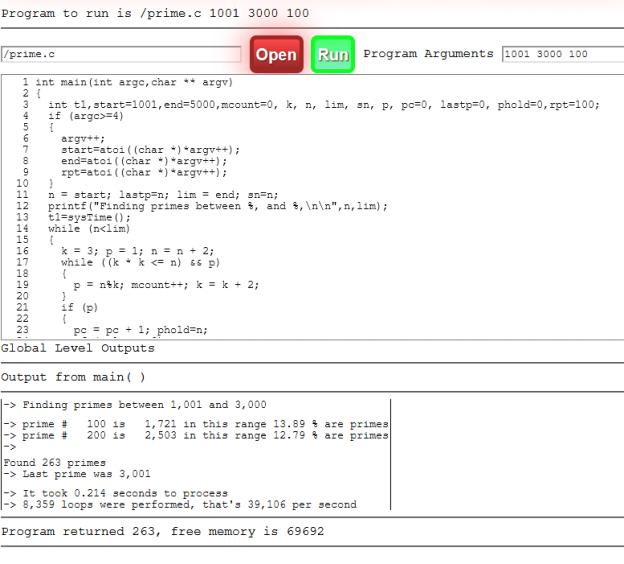


The Change Name button will rename the file from Old Name to New Name.

The New File button creates an empty file with the name newfilex.txt and opens it in the edit view. If you do not save the file, for instance you click the File Manager button in the edit view before a Save, the file will not appear in the file system. Each New File button click will create files with higher sequence numbers than what already exists in the file system.

The Program Arguments filed lets you pass values to a program if it is coded with a main function following the main(int argc,char \*\* argv) {…} conventions. For instance the prime.c program accepts program arguments and its operation with Program Arguments is shown in two following figures.





The program by default runs with start=1001, end=5000, andrpt=100. When at least 3 arguments are passed ( making argc=4 since argv[0] if the program name) the code uses the parameters to fill in start, end, andrpt sequentially. The lines

4 if (argc>=4)

5 {

6 argv++;

7 start=atoi((char \*)\*argv++);

8 end=atoi((char \*)\*argv++);

9 rpt=atoi((char \*)\*argv++);

Illustrate how to use the argvvalues to set integers, they can just as easily be assigned as character string pointers i.e. char \* by not using the atoi() function.

The Code Template and HELP buttons in the Admin toolbar are discussed in the following two paragraphs.

The Code Template button creates a simple program template much like the Arduino File/New button. The template includes main, setup, and loop functions as shown below:

void setup();

void loop();

int main(int argc,char \*\* argv)

{

setup();

//for (;;) loop();

// Uncomment line above to run your loop() function forever.

// When running loop forever,

// you'll get no response to web events

// unless you include doLoop();

// in your interpreted loop() function

return 0;

}

void setup()

{

}

void loop()

{

doLoop();

}

The program’s main function is set to return an int and uses argc and argv inputs which are gathered from the Program Arguments textbox when the program is run. Eventhough argc and argv are declared, you don’t need to make any use of them. The main function calls setup and will thereafter repeatedly call loop() if the for (;;) loop(); line is

uncommented. The template includes doLoop() in the loop function to allow the browser buttons to be active when the sketch is running. Note however, that if doLoop() is called, that program outputs to the console will stop since the

browser session is closed when doLoop() is called. doLoop() exists to provide a means to stop programs that have

continuous Loop operations since it enables the Screen Capture and ResetESP32 buttons to be used. There are two

means of getting program outputs when running a loop() function continuously in a program. First, you can use the

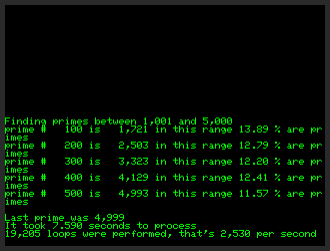
sprint(char \*) function to send output to the Serial Monitor – I like to use Termite by CompuPhase since it supplies and RTS control that will also reset the ESP32.



Secondly, you can use a TFT, OLED, orePaper display with the ESP32 and send output to those devices. Particularly

useful is void setConsoleOn(int); 0 turns console output off for printf function, 1 turns console output on. So setConsoleOn(1); will send your program’s printf output to a console on the TFT. As an example, if you add

setConsoleOn(1); near the top of the prime.c program, Save the file, and click Run, the TFT output will appear as shown below.



A better example of using a continuously running Loop() function including doLoop() is contained in the simpleBME.c program. This program uses a BME280 sensor that measures pressure, temperature, and relative humidity and

communicates over an I2C serial interface. The interpreter includes the following functions for the BME280 sensor.

void BME\_init(); must be called before using the BME280 sensor

float BME\_readPressure(); returns barametric pressure from BME280 sensor

float BME\_readRH(); returns relative humidity from BME280 sensor

float BME\_readTemp(); returns temp from BME280 sensor

The simpleBME.cprogram uses all four functions. Its primary BME 280 calls and output functions are

float temp=BME\_readTemp();

float pressure=BME\_readPressure(); float rh=BME\_readRH();

float tc=(temp-32)\*5/9;

//DP = 243.04\*( LN(RH/100)+( 17.625\*T )/ (243.04 + T )) / (17.625 - LN(RH / 100) - (17.625\*T ) / (243.04 + T ))

float dp = (243.04\*(log(rh/100)+((17.625\*tc) / (243.04 + tc))) / (17.625 - log(rh / 100) - ((17.625\*tc) / (243.04 + tc))))\*9.0 / 5 + 32; printf("\n");

printf(" Temp is %0.1f F\n",temp);

printf(" Pressure is %0.2f in HG\n",pressure); printf(" RH is %0.1f %%\n",rh);

printf("Dew Point is %0.0f F\n",dp);

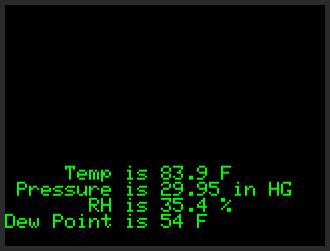
These lines read the temp, pressure, and relative humidity into float values temp, pressure, and rh and calculate the Dew Point using natural log functions as shown in the comment //DP= … Then the program format prints these values with appropriate labels and units. The program’s last lines in the loop() function are shown below.

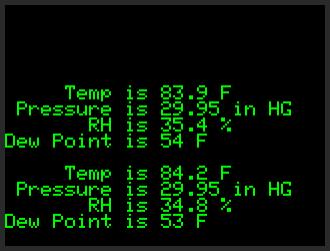
delay(5); // this delays 5 msec and doLoop() calls the ESP32Program loop() which adds another 5 msec of delay

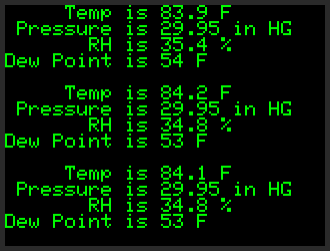
// for a total of 10 msec delay per loop cycle thus lCount%3000==0 is true every 30 seconds

This approach to running loop with a fairly short delay, assures that the ESP32Program gets to process web events

frequently (here, typically 100 times per second) and results in a good user response. The screen captures shown below were taken using the Screen Capture button while the program was continuously running on the ESP32.







The /oledDemo.c program on the ESP32 provides a simple example of using a 128x64 pixel OLED I2C attached display with the interpreter. To use the o\* functions, OLED must be defined when compiling the ESP32Program sketch. The program’s setup function runs the following lines

int i;char buf[33];

oclear();

odrawString(0,0,"Display is 6x21");

for (i=1;i<6;i++) {

sprintf((char \*)&buf,"Display line %d is drawn here",i);

odrawString(0,10\*i,(char \*)&buf);

}

odisplay();

delay(5000);

oclear();

odrawString(0,25,"Hello World");

odisplay();delay(3000);oconsoleInit();

for (i=0;i<21;i++)

{

sprintf((char \*)&buf,"Line %d is put on the display here",i);

oconsolePrintln((char \*)&buf);delay(250);

}

The help lines for these functions are

void oclear(void); clears OLED display, you still have to call odisplay to show

changes

void oconsoleInit(); must be called to start OLED console display

void oconsolePrintln(char \*); draw char string at current cursor x,y

void odisplay(void); update display

void odrawLine(int,int,int,int); x1,y1,x2,y2

void odrawRect(int,int,int,int); x,y,width,height

void odrawString(int,int,char \*); x,y in pixels,char string

void ofillRect(int,int,int,int); x,y,width,height

void oprint(char \*); display in OLED at current cursor position

void oprintln(char \*); display in OLED at current cursor position with [CR]

void osetCursor(int,int); x,y

void osetPixel(int,int); x,y

When using the OLED, be sure to call odisplay(); each time drawing operations are completed or your display changes will not be shown.

Similar to the OLED display, there is support for a 1.54 inch 200x200 ePaper display that is particularly useful on sensor projects since it provides a low power solution that runs well on batteries and can take advantage of the EPS32’s deep sleep modes and still display its information during deep sleep. A fairly complete and useful example ePaper program is /ePaperDemo.c. This program provides an example temperature, relative humidity, barometric pressure, and dew point bar graph display. The ePAPER global must be defined when compiling the ESP32Program to use an EPaper display. The available functions are shown below.

void ePaper\_drawLine(int,int,int,int,int) x1,y1,x2,y2,color

void ePaper\_drawRect(int,int,int,int,int) x,y,width,height,color -- 0 is black 0xffff is white void ePaper\_fillRect(int,int,int,int,int) x,y,width,height,color

void ePaper\_init() initialize ePaper display

void ePaper\_powerDown(); put the ePaper display in powerDown mode void ePaper\_println(char \*);" print char string

void ePaper\_setCursor(int,int); x,y void ePaper\_setTextColor(int); color

Color values are 0 for black and 0xffff for white.

**Extending/Enhancing the Interpreter**

When viewed in the Arduino IDE the ESP32Proram appears to have a bewildering number of files and functions. Most

modifications to ESP32Program will be made by changing code in the top level ESP32Program tabor with changes in

clibrary. You can change the web interface and even translate the web interface elements all in ESP32Program. It goes

without saying, that you need to be comfortable with the WebServer functions provided by the WebServer library and

with C/C++ programming in general before attempting to successfully change parts of either the ESP32Program or

clibrary. Always make a backup copy of these files before making changes so you have a safe fallback if things go horribly wrong after your changes.

The book *Building Blocks for IOT with your PC and WiFi Peripherals* available soon as a Kindle book goes into more detail about the picoc C Language Interpreter and many other Arduino IDE examples using the ESP32 or an ESP8266.

The clibrary tab is where you would make changes to add functions and peripherals to the Interpreter. These functions are basically marshalling function arguments provided by the interpreter and then using them to call the underlying C or C++ functions that your project requires. The OLED functions area good examples of adding marshalling and calling

functions. As an example, search for odrawRect in the clibrary tab. There will be three occurrences, we will first discuss the one nearest the top of the file. It defines a function which receives parameters from the Parser. What is most used in the marchalling of parameters for calls to native functions is the struct Value\*\* Param. Param has an array of

parameters of length NumArgs. The interpreter makes sure that function calls to marshalling calls inclibrary have the right number of arguments and types.

void odrawRect(struct ParseState \*Parser, struct Value \*ReturnValue, struct

Value \*\*Param, int NumArgs)

{

int x=Param[0]->Val->Integer;

int y=Param[1]->Val->Integer;

int w=Param[2]->Val->Integer;

int h=Param[3]->Val->Integer;

display.drawRect(x,y,w,h);

}

The assignment to int values of Param[x]->Val->Integer are collecting the values of x,y,w, and h to pass to a native object external to the interpreter. In this case it is the display object which is made know to clibrary bylines 270to 271 which

read

#include "SSD1306Wire.h"

extern SSD1306Wire display;

This gives clibrary all it needs to interface to the display object and make calls to functions like display.drawRect( … ) If you check, you will see that the display object is created in the ESP32Program tab.

With these underpinnings in place, we can successfully call display.drawRect( … ) with parameters passed from the Interpreter code.

Now we need to let the interpreter knowhow to activate our odrawRect function. Searching again for odrawRect brings you to the line

{ odrawRect, "void odrawRect(int,int,int,int);" }, // x,y,width,height

This is a declaration of a struct LibraryFunction that provides first the name of the function to call when the function

prototype is encountered,in this case odrawRect. The part in quotes is a template for the function which says that the interpreter should recognize avoid function namedodrawRect with 4 int arguments. This gives the interpreter enough information to parsecallstoodrawRect in our interpreted programs and allows it to enforce argument counts and types before calling odrawRect. The struct LibraryFunction entries are all in this format and are enclosed with { },

***Be very careful when making changes in the CLibrary[] definitions, particularly since the function template - the part in quotes - is not evaluated until picoc is starting and any syntax errors within the quotes will cause a runtime error and an immediate crash of the interpreter without giving any clue as to what or where the error is.***

In any event, **if you comprehend what the preceding material expressed**, you can add code to make new sensor/device functions available in the interpreter.