SP500X
INTEGRATORS GUIDE

handheld

1. Build Environment

1.1. Build Tools

1.1.1. IDE

Notepad or IDLE are recommend for editing Python files. It is important to use a text editor that does not add hidden formatting and saves the file as ASCII or UTF-8

1.1.2. Packaging Tool

The SP500x supports TAR packing of files. Peazip is the recommend Windows tool for this packing.

1.1.3. Device File Structure

1.1.3.1. Archive Folder – Used to save a copy of the last update package
1.1.3.2. Downloads – Used to store an update package (tar file) to be applied to the unit
1.1.3.3. Icons – Used to store PNG files for drawing to the LCD
1.1.3.4. Images – Used to store IG files. These are the templates used to print by the SP500x
1.1.3.5. Settings – Used to store device specific settings
1.1.3.6. Upgrade – Used to store update python files to applied to the unit

1.1.4. How to Package update files for the SP500x

To create an APP update package, create a local folder that mimics Device File Structure with the folders you want to update (Icons, Images, and Upgrade):

- Copy your files to the appropriate folders. Then create a TAR file packing the folders.
- Copy the TAR file to the Downloads folder on the device.
- Eject the device from the PC – this is important to force Windows to flush your files to the device
- Reboot the device and the TAR file will be automatically unpacked and applied by the device

1.2. Debugging Tools

1.2.1. Terminal Applications

The SP500x does not have a native debugger. It is recommend to use the print() function of Micropython to send debug messages to the devices terminal to aid in debugging. The following terminal application work with the SP500x:

- Tera Term: https://osdn.net/projects/ttssh2/releases/
- PuTTY: https://osdn.net/projects/ttssh2/releases/

The terminal is used by opening a Serial Port connection to the device. The device will connect as a USB Serial Port.

2. Usage

2.1. STM Peripheral Resources

The following sections describe the STM processor's pre-allocated peripheral resources required to be used for the PySDK. Each REQUIRED resource is hardwired in the on-board electronics. OPTIONAL resources can be switched, but are required for background operations to be performed.

2.1.1. Timers

There are 14 total timers on the STM processor.

ID	USAGE	Attached Peripheral
1	OPTIONAL	Background Buzzer management
2	OPTIONAL	Used for background capper motion monitoring
3	OPTIONAL	Proximity sensor monitoring
4	OPTIONAL	Used for the hardware managed heartbeat LED
5	REQUIRED	Buzzer PWM control
6	OPTIONAL	LED Blink control
7	UNUSED	
8	REQUIRED	Capper PWM generation
9	UNUSED	
10	UNUSED	
11	UNUSED	
12	REQUIRED	LCD frame rate control
13	UNUSED	
14	UNUSED	

2.1.2. UART

There are 2 on-board UART peripherals that have required use.

ID	Baudrate	Peripheral
7	115200	Scanner module
1	115200	Network module

2.2. Initialization and Runtime configuration

The SDK modules initialization sequence is broken up into 3 stages:

1. Setup the Database module and Logger
2. Pre-Initialize the modules: access through Py_SDK.SystemInit()
    a. Wait for external peripherals to finish booting
    b. Open access to all hardware ports for external peripherals
3. Load the configuration files into the Database
4. Perform the post configuration steps once data is set: done through Py_SDK.SystemConfig()
    a. Perform one time only steps for configuring rendering engine
    b. Perform any one time memory allocations

Example:
From Py_SDK import Py_SDK

DB.UpdateParams('/mmc/Settings/Settings.ini')
DB.UpdateHiddenParams('/mmc/DB.ini')
try:
 Logger.SetBase(DB.config.get('Logger','PATH'))
 Logger.SetLogSize(int(DB.config.get('Logger','LOG_SIZE')))
 Logger.SetLogCount(int(DB.config.get('Logger','LOG_COUNT')))
 Logger.SetLogLevel(int(DB.config.get('Logger','LOG_LEVEL')))
 Logger.Debug("Main","Initialize Logging")
except Exception as e:
 print(str(e))
 print("ERROR:: Main App trying to parse Config")

Py_SDK.SystemInit()
#Load all settings files, see Required Configuration Files
DB.rsrc.Load('HwConfig','/mmc/Settings/hw.xml')
DB.rsrc.Load('PrintSettings','/mmc/Settings/ps.ini')
DB.rsrc.Load('Density','/mmc/Settings/den.csv')
Py_SDK.SystemConfig()

2.2.1. Required Configuration Files

Name	Format	Description
Settings	INI	Top level settings file responsible for setting: ● Module level configuration ● Logger configuration ● Server details
DB	INI	Non-User controlled parameters containing: ● Unit Serial Number ● Last Pen detected ○ Pen ID ○ Page Count ○ Approx. Ink used ● Date Time at last power cycle
Network	XML	Detailed network settings XML that can be sent to onboard network module for fast setting large amounts of parameters.
HwConfig	XML	Rendering engine print tuning. **Should not be changed**
PrintSettings	INI	Rendering engine print quality tuning. The only parameter that can be changed is **Density** with the restriction 0 < value < 9. This will control the ink usage in 6.25% increments, given speed requirements the value cannot be above 8 (50%)
Density	CSV	Rendering engine density control. **Should not be changed**

2.3. Py_SDK Interface

The Py_SDK interface module is responsible for high level initialization of the on-board peripherals and modules as well as providing a method to switch backend modules such as Zebra or Honeywell for Scanner.
i.e. from Honeywell import Honeywell as Scanner
Or
from Zebra import Zebra as Scanner

Import statement for this module:
from Py_SDK import *

2.3.1. Member Functions

Py_SDK.SystemInit()

Pre-Initialize the external peripherals and modules.

Py_SDK.SystemConfig()

Apply the current configuration to the SDK

Py_SDK.PowerLevel(mode)

Enter the specified Power Level and shutdown/enable the appropriate peripherals.

2.4. Database

The Database module is used to manage the external volatile memory of the control board as well as holding current status/settings information used by SDK modules.

Import statement:
from Database import *

2.4.1. Member Functions

DB.UpdateParams(path='/mmc/Settings/Settings.ini')

Read in the top level settings INI file and set all the local values accordingly.

Parameters:
Path - Full path to settings file
Returns:
-0 on Successful load
-1 for invalid file
-2 for memory allocation error

DB.UpdateHiddenParams(path='/mmc/DB.ini')

Read in the last instance of the hidden parameters managed by the SDK

Parameters:

Path - Full path to settings file

Returns:

0 on Successful load
-1 for file access error
-3 Invalid data

DB.rsrc.Load(name, path)

Load a Rendering engine resource.

Parameters:

Name - Resource Name
Path - Path to file

Returns:

0 on success
-1 on File access error

2.5. Power Modes

The SDK provides 3 levels of power mode:
1 - Active Mode: Enable all peripherals, LCD features, and allow for scanning and operational mode network messages
2 - Reduced Power Mode: turn off external peripherals, disable LCD and scanner. Network transmissions will be halted as well, but connection maintained. Any button access will exit this mode
3 - Deep Sleep Mode: shut down ALL peripherals and disable network connection. Any button access will exit this mode and force a required unit reboot.

The SDK interface to enter different power modes is:
Py_SDK.PowerLevel([Mode])

3. Peripheral/Support Modules

The Following sections describe the various custom modules in the PySDK package. **Anything not described here should be considered for private use and should NOT be changed or accessed.**

3.1. Print

The Print Module is responsible for managing the external Printhead interface and rendering engine.

Import statement to use the module:
import Print

3.1.1. Usage

Pre-Usage

Before the Module can be used, the following steps must be performed:

1. Print.Init() must be called. Ideal location is in Py_SDK.SystemInit()
2. A properly and valid pen must be checked through Print.CheckPen()
3. The Rendering engine configuration files must be loaded

Example:

```
import Print

doorCloseEvent = False
validPen = False

def doorEvent(p):
   if(p.value() == 0):
         doorCloseEvent = True
Print.Init()
# Perform rest of initialization
if(doorCloseEvent):
         if(Print.CheckPen() == 0):
               validPen = True
```

Print Usage

Before a page can be printed the following steps must be performed:

1. Pen must be powered on using Print.Start. This can be done well ahead of time if desired, it is mainly a power saving measure to call once ready.
2. A template file must be loaded using Print.LoadImage
3. For dynamic print data, Print.SetFields must be called.
4. Print.PrintImage to process and download print data.
5. OPTIONAL: Print.ClearFields can be called at this point to prevent old data re-printing.

Example:

```
import Print

Print.LoadImage('/mmc/images/Template.ig')

#continue operation until dynamic data detected
Print.SetFields('Template', Fields)
Print.PrintImage()
```

```
Print.ClearFields()
#Wait for page to start
while(Print.WaitPageEvent(0,100) != 0) {}
#Perform an start of print operations

#wait for Page to end
while(Print.WaitPageEvent(1,100) != 0) {}
```

3.1.2. Data Members

Print.ProcessTime

This value holds the time in ms it took to process the last image.

Print.height

This value holds the current loaded plot downweb height in pixels.

Print.TemplateName

This string is the currently loaded template

3.1.3. Member Functions

Print.Init()

Initialize the rendering engine and external print head control module.

Print.CheckPen()

Perform an electrical connection check and compare the currently installed Pen with the supported Pen Ink.
On Successful electrical check and validation, the pen will be powered, configured, and ready to receive print data.

Returns:

0 on Success
-1 Electrical connection check failed
-2 Invalid Pen installed.

Print.Start(doorOpen=True)

Apply power to the print head, and if doorOpen == true, then auto open the printhead cap.

Parameters:

doorOpen - if True, the capper door will open after the printhead is powered
if False, the capper door will remain in current state

Print.LoadImage(path,scale_mode=4)

Pre-Load a template file, must be called BEFORE Print.SetFields. This method will perform pre-calculations for dynamic fields, pre-render static fields and pre-calculate the image size.

Parameters:

path - Full path to template IG file
scale_mode - downweb scaling relative to the external encoder resolution.
Default value of 4 is the correct setting for SP500x

Print.SetFields(TemplateName, fields)

Load the dynamic fields into the rendering engine. If the TemplateName does not match the current loaded name, then Print.LoadImage will be auto-called with default path '/mmc/Images/'+TemplateName + '.ig'.

Parameters:

TemplateName - Template containing the dynamic fields
fields - array of 2 elements with index 0 = key, index 1 = value

Print.PrintImage()

Process the current image and download to external Pen Controller. On return, the system is ready to print.

Print.ClearFields()

Clear the dynamic print fields, use this to prevent old data from reprocessing the next time Print.PrintImage is called.

Print.WaitPageEvent(Eid, timeout)

Wait for a page event to occur. The supported events are: Start of Print(SOP) and End of Print(EOP)

Parameters:

Eid - Event ID to wait for, 0 = SOP, 1 = EOP
Timeout - timeout in ms to wait

Returns:

0 - Event occurred
-1 - Timeout

Print.IdleTime()

Return the time sense the last successful print in ms.

Print.PrintTime()

Total time the print took to occur in ms.

Print.SetDensity(density)

Update the print density in percent. The valid increments are in 6.25%, the value will be rounded down to the nearest supported value. If a template has already been preloaded, then it will be auto re-loaded in order for the updated density value to take effect, NOTE this could take upto 100ms.

Parameters:

Density - Density percent, valid values are 6.25%-100% in 6.25% increments

3.2. Buzzer

The Buzzer module controls the on board piezo buzzer.

The Pitch and Volume of the buzzer is controlled using the tim object.

Import statement for this module:
from Buzzer import Buzzer

3.2.1. Usage

No specific steps needed to initialize the buzzer module, the static class performs all the device access allocation required.

3.2.2. Data Members

Buzzer.tim

This timer controls the PWM timer instance number 5, required to provide the volume and pitch control of the piezo buzzer.

Pitch is controlled using the base frequency of the PWM signal.

Volume is controlled using the duty cycle of the PWM signal (NOTE: max volume is around 50-60%)

Buzzer.pollTimer

The pollTimer object is used to control the background duration of the tone. This is allocated to use Timer instance 1.

3.2.3. Member Functions

Buzzer.beep(pitch, volume, num_ticks)

Perform a blocking beep with the specified pitch, volume and duration. This function will return once the tone is complete.
Parameters:

pitch - frequency in Hz
volume - Duty cycle percent (0-100)
num_ticks - Duration in ms

Buzzer.play_tones(tones)

Play a collection of Tones in a background timer handler. This function will return immediately before the tone actually finishes.

Each tone has 3 parameters: [tone frequency in Hz, duty cycle percent (0-100), duration in ms]
Parameters:

tones - array of tones type

3.3. LEDControl

The LEDControl module allows for controlling the onboard LEDs on/off and blink control.

Supported LEDs are:

- Green1
- Green2
- Red1
- Red2
- Green3

Green 1,2 and Red 1,2 are the back/tail lights. Green3 is an internal LED to the unit, visible with the enclosure removed.

Import statement for this module:
from LEDControl import LEDControl

3.3.1. Usage

The LEDControl module will require a Timer instance allocated, default is Timer instance 6.
Use LEDControl.Init for this.

3.3.2. Data Members

Map

Friendly name mapping to external LED PIN, this also keeps track of the current Blink configuration.
[LED PIN, Blink Duration, Blink Countdown value]

3.3.3. Member Functions

LEDControl.Init(p=1, t=6)

Initialize the LEDControl object, set all the LEDs to the specified initial value and allocate the Timer instance for blink support.

Parameters:

p - Initial LED value. 0 = one, 1 = off
t - Timer instance to use, DEFAULT is instance 6

LEDControl.TurnOn(ledId)

Turn on the specified LED
Parameters:
ledId - Friendly name for the LED

Returns:
True - successfully turned on LED
False - if invalid friendly name

LEDControl.TurnOff(ledId)

Turn Off the specified LED.

Parameters:
ledId - Friendly name for the LED

Returns:
True - successfully turned off LED
False - if invalid friendly name

LEDControl.Blink(ledId, freq)

Start blinking the specified LED and frequency. Can only support frequencies 1-10 Hz.

Parameters:
ledId - Friendly name for the LED
freq - Frequency in Hz to blink the LED

Returns:
True - successfully started blinking
False - frequency or ledID is invalid

LEDControl.Stop(ledId)

Stop blinking the LED and turn off.

Parameters:
ledId - Friendly name for the LED

Returns:
True - successfully started blinking
False - ledId is invalid

3.4. MLCD

The MLCD module is responsible for managing the memlcd and frontlight.

Import statement for this module:
from MLCD import MemLCD

3.4.1. Usage

No special Usage for this module. The import will allocate the external GPIO, DAC and SPI interfaces required to control the memlcd.

3.4.2. Data Members

MemLCD.fb

Frame buffer holding the current pixel data for the LCD.

MemLCD.dac

Digital to Analog(DAC) instance reference to control the frontlight.

MemLCD.width

Width in pixels of the LCD, current LCD is 230 pixels wide

MemLCD.height

Height in pixels of the LCD, current LCD is 303 pixels high

3.4.3. Member Functions

MemLCD.frontLightOff()

Turn off the LCD front light.

MemLCD.frontLightOn(pct)

Turn on the the LCD front light with the specified percent.

Parameters:

Pct - Percent between 0-100 for the front light intensity.

MemLCD.pixel(x,y,val)

Set the pixel value at the Column(x) row(y) in the local Framebuffer. Update must be called before the LCD displays the pixel.

Parameters:

X - column to set, must be > 0 and < width
Y - Row to set, must be > 0 and < height
Val - value to set, 1 = set, 0 = clear

MemLCD.text(str,x,y,color=0)

Add the specified text string to the local Framebuffer. Update must be called before the LCD displays the text.

Parameters:

Str - string to display
X - upper left column to set, must be > 0 and < width
Y - upper left row to set, must be > 0 and < height
Color - color value to set.

MemLCD.png(path,x,y)

Add the specified PNG file to the local Framebuffer. Update must be called before the LCD displays the icon. Only 1 bit/pixel format is accepted.

Parameters:

Path - Path to PNG file
X - upper left column to set, must be > 0 and < width
Y - upper left row to set, must be > 0 and < height

MemLCD.update()

Push the current Framebuffer into the LCD. Will return immediately, LCD will update in the background

MemLCD.clear()

Clear the entire Framebuffer.

MemLCD.fill_rect(x,y,w,h,v=1)

Fill the local Framebuffer with a rectangle with described parameters. Update must be called before the LCD displays the rectangle.

Parameters:

X - upper left column to set, must be > 0 and < width
Y - upper left row to set, must be > 0 and < height
W - Width in columns for the rectangle
H - Height in rows for the rectangle
V - value to set, 1 = set, 0 = clear

MemLCD.fill(val)

Fill the entire Framebuffer with the specified value. Update must be called before the LCD displays.

Parameters:

Val - value to set, 1 = set, 0 = clear

MemLCD.Busy()

Check if the LCD is in the middle of an update.

Returns:

True - if Busy
False - if Idle

3.5. Motor

General Motor control. The only motor available on the SP500x is the capper motor.

3.5.1. Usage

There are no special pre-Usage for the Motor class, the static initialization will configure the necessary peripherals.

3.5.2. Data Members

Motor.limit

The ADC max value before an auto stop.

Motor.pulse

The PWM duty cycle percent. I.e. 0 - no motion, 100 - max motion and torque

Motor.timeout

Move timeout in ms.

3.5.3. Member Functions

Motor.Open(t=300)

Open the capper until the the ADC feedback reaches the specified value or the pre-configured timeout is reached.

Parameters:

t - ADC threshold to auto-stop the motion

Motor.Close(t=300)

Close the capper until the the ADC feedback reaches the specified value or the pre-configured timeout is reached.

Parameters:

t - ADC threshold to auto-stop the motion

Motor.Stop()

Force stop the motor motion immediately.

3.6. Battery

3.6.1. Usage

The initialization function will communicate with the currently attached battery. On successful return, the current battery model will be detected and current capacity set.

The default parameters for SCL and SDA MUST be used for the SP500x device.

3.6.2. Data Members

Battery.CurrentModel

Currently attached battery model.

Battery.TotalCap

Total capacity reported by the battery at the current usage.

Battery.Remaining

Remaining capacity reported by the battery at the current usage.

Battery.CycleCount

Current measured discharge cycle count. As this field is only updated after a full re-charge or new battery inserted, this will be updated after an initialization call is made.

3.6.3. Member Functions

Battery.Initialize(scl_name='I2C2_SCL', sda_name='I2C2_SDA', freq=40000)

Open communication channel to the attached smart battery.
Parameters:

Scl_name - Pin name used to indicate the I2C Clock pin (Must be 'I2C2_SCL')
Sda_name - Pin name used to indicate the I2C Data pin (Must be 'I2C2_SDA')
Freq - Frequency in Hz for I2C communication, suggest using 40KHz

Battery.GetLevel()

Request the current remaining level in percent.
Returns:

Remaining capacity with current usage.

3.7. Scanner

3.7.1. Usage

Power On Initialization

The Scanner module needs to pre-initialize the HW connection and then apply any runtime configuration data:

from Py_SDK import Scanner

Scanner.Init()
#Read in all configuration parameters
Scanner.Configure()

Scan Barcode

The Scanner will need to start before receiving a barcode, then use WaitBarcodeReady.

3.7.2. Data Members

Scanner.ScanTime

Total time from when the scanner started to when a barcode was received.

Scanner.ActiveTime

The CPU ms time point when the scanner was turned on. Compare this value with the current CPU ms time point to calculate how long the scanner has been on.

3.7.3. Member Functions

Scanner.Init(port=7, baudrate=[9600 for Zebra, 115200 for Honeywell], bufSize=1024)

Initialize the scanner UART port.

Parameters:

Port - UART peripheral port number, for current SP500x this **MUST be 7**
Baudrate - UART baudrate for the scanner, defaults to correct value based on scanner
bufSize - local hw peripheral buffer, defaults to 1024, minimum value must be larger than the maximum number of characters all barcodes in field of view can produce

Scanner.Configure()

Push the scanner settings to the device

Scanner.LoadSettings(path=None)

Load a series of settings to the scanner module, with each command as a single line.

Parameters:

Path - Path to text file containing list of settings to load.

Returns:

1 - Success loading settings
0 - No file specified or empty
-1 - invalid file location
-2 - Invalid file type

-3 - Invalid settings found in file

Scanner.start_scan()

Trigger the scanner and enable processing barcodes

Scanner.stop_scan()

Disable the scanner

Scanner.WaitBarcodeReady(timeout)

Wait for a valid barcode to be received for the specified timeout.

The return format is a tuple with [Status, Symbology, value]

Parameters:
Timeout - wait time in ms

Returns:
No barcode found in time - [‘RET::None’,’RET::None’, ‘RET::None’]
Barcode read error - [‘RET::ERROR’,’RET::ERROR’,’RET::ERROR’]
Valid barcode found - [‘new’, <symbology name>,<barcode value>]

Scanner.Sort_barcode()

WaitBarcodeReady will use this method to filter any incoming barcode. This can be customized on per product.

If an invalid barcode is read, then the value will be ignored and the scanner re-enabled.

This method is also responsible for separating the symbology from the barcode value

3.8. Net

3.8.1. Usage

Power On Initialization

Net.Init() handles all the necessary steps to pre-configured the network module. Previous configuration data is already saved on the module.

Example using the xPico implementation:
from xPico import xPico as Net

Net.Init()

Update Network Configuration Data

After initialization is complete, the network configuration data can be updated using either a configuration file or manual commands:

Example using configuration file:

```
from py_SDK import Net

Net.LoadConfig('/mmc/Settings/Network.xml')
```

Using manual commands:

```
from py_SDK import Net

Net.SetSSID('SomeSSID')
Net.SetSecuritySuite('WPA2')
Net.SetPassPhrase('somep@ssword')
Net.SetNetworkBand('2.4GHz Only')
```

Open a UDP Communication Tunnel

```
from py_SDK import Net

Net.OpenListeningTunnel('6500')
Net.OpenRequestTunnel('192.168.10.12','6500')
```

Send/Receive data packets

Before packets can be transmitted/received, the proper communication tunnels need to be setup.

```
from py_SDK import Net

Net.SendMessage(packet)

#Wait for a valid response for 2 attempts for a packet starting with 'S' with max size 512 bytes
#Wait for a total of 1 second with 100ms polling
#UART transfers of 512 Bytes would take approximately 44ms = ((512UartB/Packet*10b/UartB) /(115200baud))
*1000.
#The delay_t parameter allows the low level hardware to pre-load data before access
for x in range(2):
  rsp = Net.ReceiveMessage(size=512, timeout_t=100, retries=10, delay_t=40)
  try:
    res = res[res.index('KS')+1:]
  except:
    continue
    #Handle response
    break
```

Set Security Modes

Security modes can be set after initialization is complete

WPA2

```
From py_SDK import Net

Net.SetSecuritySuite('WPA2')
Net.SetPassPhrase("Passcode")
```

WPA2 Enterprise

```
from py_SDK import Net

Net.SetSecuritySuite('WPA2',authentication='8021X')
Net.SetPrivateKey('/mmc/PrivKey.pem')
Net.SetTLSCertificate('/mmc/CA_Certificate.ca')
```

Setup a Bluetooth SPP Connection

Assuming Initialization is complete.

```
from py_SDK import Net
import utime
Net.BLIfcState(True)
Net.BTScan()
utime.sleep(10)
Devices = Net.BTList()
for Device in Devices:
        print(Device)

#Choose index of the Device to Pair I.e. Windows PC = index 2
Net.BTPair(Devices[2][1])
Net.BTPairSPP(Devices[2][1])
Net.BTOpenTunnel()
#Make sure Teraterm/putty is connected to the Windows COM port
Net.BTSPPWrite("Hello World from Unit!")
#Type some data into the terminal on PC
print(Net.BTSPPRead())
```

3.8.2. Data Members

Net.RSSI

Current WiFi signal strength

Net.BSSID

Currently connected access point MAC address

Net.MAC

Network module MAC address

Net.IP

If Connected to a network, this is the current attached IP address

Net.SSID

Currently attached network name

Net.SNR

Signal to noise ratio measured on last GetWLANInfo call

Net.RoamCount

Roam count measured on last GetWLANInfo call

Net.Radio_en

Ture is WiFi Radio On, False is WiFi Radio Off

Net.BL_en

Ture is Bluetooth Radio On, False is Bluetooth Radio Off

3.8.3. Member Functions

Net.Init()

Initialize the network module. If '/mmc/Settings/DefaultConfig.xml' or '/mmc/Settings/Network.xml' is found, then load the configuration data onto the module.

Net.LoadConfig(file='/mmc/Settings/Network.xml')

Load the detailed network configuration XML for low level initialization of the Network module.

On Successful load, a backup copy will be saved to: '/mmc/Settings/Network.BAK')

Parameters:

File = path to XML file to load

Net.Reset()

Perform a soft reboot of the network module, the device will need to reconnect to the network following this.

Net.FactoryReset()

Perform a factory reset of the network module. Any custom configurations will need to be re-loaded using LoadConfig

Net.GetFwVersion()

Query the network module for it's firmware version

Net.GetWLANInterface()

Query the network module and save the current WLAN interface information:

- IP address (if connected)
- MAC address of module

Net.GetWLANInfo()

Query the network module and save the current WiFi settings into their corresponding data members:

- RSSI
- BSSID
- SSID
- SNR
- RoamCount

Net.OpenVirtualTunnel()

Open a local tunnel for querying network status without having to disconnect other tunnels.

Net.OpenRequestTunnel(t_ip,t_port)

Open a UDP tunnel targeting the specified IP and PORT for transmit

Parameters:

t_ip - Target IP port formated as xxx.xxx.xx[x].xxx
t_port - Target UDP part

Net.OpenListeningTunnel(port)

Open a UDP receive listening tunnel on the specified port.
Parameters:

Port - listening port to monitor

Net.SendMessage(data)

Send the specified message packet to the currently opened Tunnel.

Parameters:

Data - bytes array data packet

Net.ReceiveMessage(size=1023,timeout_t=100,delay_t=30,retries=10)

Monitor the listening tunnel for receive data for the specified time with maximum expected data.

Parameters:

Size - Maximum size to receive
Timeout_t - timeout to receive data in ms
Delay_t - timing tuning delay to wait after receiving first byte
Retires - Number of times to wait for the specified timeout

Returns a UTF-8 string of the received data, the beginning of the packet is following the SOF indicator 'K' + expected message start characters. I.e. [ignore characters]KS…

Net.GetRSSI()

Query the network module for the current signal strength. Will return and update the local RSSI value.
Returns: current RSSI in decibels.

Net.SetSSID(ssid)

Support method to immediately set the target SSID without required an XML configuration file
Parameters:

ssid - SSID to set.

Net.SetPassPhrase(passphrase)

Set network passphrase to control connect.
Parameters:

passphrase - Passphrase to set.

Net.SetNetworkBand(band)

Set the target network band.

Parameters:

band - Band to set. Supported values:

- 2.4 GHz Only
- 5 GHz Only
- Dual

Net.SetSecuritySuite(suite,authentication="PSK",option="Key Type:Passphrase")

Set the target security suite.

Parameters:

suite - suite to set. Supported values:

- NONE
- WPA
- WPA2

Authentication - Authentication mode to use DEFAULT = PSK. Supported Values:

- PSK -
- 8021X - WPA2 enterprise

Option - sub-option for authentication mode, DEFAULT = 'Key Type:Passphrase'. The supported options will vary depending on Authentication mode.

PSK supported Values:

- Key Type:Passphrase
- Key Type:Hex,Key:<Hex Key>
  - 64 Hex characters required in option Key ie: ABCEDF..0123455

WPA2 Enterprise supported Values:

- IEEE 8021X:EAP-TLS,Username:<UserName>,Credentials:<Credential>
  - WPA2 Enterprise EAP-TLS using UserName and Credential
- IEEE 8021X:PEAP,Username:<UserName>,Password:<Password>
  - WPA2 Enterprise PEAP using UserName and Password
- IEEE 8021X:PEAP,PEAP ver:0, PEAP Option:EAP-MSCHAPV2
  - `PEAP Version could be 0 or 1`
  - `PEAP Option could be EAP-MSCHAPV2,EAP-MD5, EAP-TLS`
- IEEE 8021X:LEAP,Username:<UserName>,Password:<Password>
  - `WPA2 Enterprise LEAP using UserName and Password`
- IEEE 8021X:FAST,Username:<UserName>,Password:<Password>,FAST Option:MD5
  - `WPA2 Enterprise FAST using UserName and Password`
  - `FAST Option MD5 (MSCHAPV2, GTC)`
- IEEE 8021X:EAP-TTLS,Username:<UserName>,Password:<Password>,Credentials:<Credential>, EAP-TTLS Option:PAP
  - `WPA2 Enterprise EAP-TTLS using UserName and Password, and server Credential`
  - `With EAP-TTLS Options are`
    - `EAP-MSCHAPV2`
    - `MSCHAPV2`
    - `MSCHAP`
    - `CHAP`
    - `PAP`
    - `EAP-MD5)`

Net.GetDeviceInfo()

Returns the current configured SSID and MAC address of module

Returns:

- SSID name
- MAC address

i.e. Net.GetDeviceInfo() returns ('PrinterNetwork','01ABCDEF00112233')

Net.SetTLSCertificate(certificate)

Load CA certificate for WPA2 Enterprise security

Parameters:

- Certificate - path to a CA file to load onto the network module

i.e. Net.SetTLSCertificate('/mmc/CA_Certificate.ca')

Net.SetPrivateKey(key)

Load private key for WPA2 Enterprise security

Parameters:

- Key - Path to PEM file to load onto the network module

i.e. Net.SetPrivateKey('/mmc/PrivKey.pem')

Net.WiFiIfcState (enable=True)

Turn the WiFi Radio On/Off

Parameters:

- enable – True = Turn Radio On / False = Turn Radio Off

Net.BLIfcState (enable=True)

Turn the Bluetooth Radio On/Off

Parameters:

- enable – True = Turn Radio On / False = Turn Radio Off

Net.BTScan()

Perform a scan for available Bluetooth devices. NOTE: Net.BLIfcState(True) must be called before this will work.

The method will return immediately, but the actual scan can take upto 10-20 seconds before all devices are discovered.

Net.BTList()

Return the list of detected Bluetooth devices. NOTE: Net.BTScan() must be called and a wait period of at least a few seconds, before this function will return up to date information.

The return data will be formatted as an array of available devices. Each entry has 4 elements:
[Device Friendly Name, MAC Address, RSSI Value, Radio Type, Paired status]

Net.BTPair(MAC)

Pair with the specified device via MAC address. This is element index 1 of the Device entry returned from Net.BTList().

Net.BTPairSPP(MAC,save=true)

After a Device has been paired via Net.BTPair(MAC), It will need to be set up as a SPP connection. This method will handle marking this device as a SPP device and create the necessary linkages to the HOST UART interface.

Parameters:

- MAC - MAC address of previously paired device to setup for SPP connection
- Save - True = Values will be written to defaults and will survive a reboot(DEFAULT), False = Values will only be valid during the current reboot cycle.
-

Net.BTOpenTunnel()

Before Bluetooth communication can be sent to the host, this method must be called after every reboot or reset.

Net.BTSPPWrite(data)

Transmit data to the paired SPP device.
Parameters:

- Data - string data to transmit.

NOTE: Net.BTOpenTunnel and Net.BTPairSPP will need to be called before a Write can occur.

Net.BTSPPRead()

Read data from the paired SPP device. If no data is available, a blank string will be returned.

NOTE: Net.BTOpenTunnel and Net.BTPairSPP will need to be called before a Write can occur.

3.9. Logger

This module is responsible for managing local log files counts, sizes and what messages are saved. The micropython filesystem is NOT interrupt safe which means that logging calls **CAN NOT BE CALLED IN INTERRUPT CONTEXT**.

3.9.1. Usage

The Logger requires various parameters before the modules can properly use the internal logging:

- Use SetBase to set the base folder path
- Use SetLogSize to set the maximum log file size
- Use SetLogCount to set the maximum log file count
- Use SetLogLevel to set the desired runtime level to log

3.9.2. Data Members

No public data members should be accessed in this module

3.9.3. Member Functions

Logger.SetBase(path)

Set the base folder for all log files to be created. I.e. '/mmc/logs'
Parameters:

Path - folder path to base log file destination. If the folder does not exist, it will attempt to be created.

Returns:

0 on success
-1 on failure to create folder

Logger.SetLogSize(size)

Set the maximum log file size in bytes. Once this size is reached, a new log file will be started.

Parameters:

Size - maximum size in bytes for each log file

Logger.SetLogCount(count)

Set the maximum number of log files to maintain. Once the file count is above this value, the oldest file will be removed.

Parameters:

Count - Maximum number of log files to maintain

Logger.SetLogLevel(level)

Set the desired level of logging to save to the filesystem.

Supported Levels:
>3 - All log messages: DEBUG,WARN,ERROR,CRITICAL are saved
2 - Disable DEBUG messages: WARN,ERROR,CRITICAL are saved

NOTE: ERROR and CRITICAL will always save.

Parameters:

Level - Log level to set

Logger.Flush()

By default the log auto flushes to the filesystem after 10 total log messages are sent. This method will force an immediate flush.

Logger.Debug(name,msg)

Add a log message with DEBUG log level.

Parameters:

Name - module name used for distinguishing source of the log message
Msg - pre-formatted log message to save

Logger.Warn(name,msg)

Add a log message with WARN log level.

Parameters:

Name - module name used for distinguishing source of the log message
Msg - pre-formatted log message to save

Logger.Error(name,msg)

Add a log message with ERROR log level.

Parameters:

Name - module name used for distinguishing source of the log message
Msg - pre-formatted log message to save

Logger.Critical(name,msg)

Add a log message with CRITICAL log level.

Parameters:

Name - module name used for distinguishing source of the log message
Msg - pre-formatted log message to save