FLYPORT APIs 2.2 – Quick reference

revision 1.0

Generated by Doxygen 1.8.0

[this page has intentionally been left blank]

Contents

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Module Documentation

2.1 Delays

Functions

- void vTaskDelay (int delay)
- void DelayMs (WORD ms)
- void Delay10Us (WORD ms)
- void vTaskSuspendAll ()
- void xTaskResumeAll ()

2.1.1 Detailed Description

In this section are explained all the commands to manage delays and task execution.

2.1.2 Delays

2.1.3 Function Documentation

2.1.3.1 void Delay10Us (WORD *ms*)

Delay10us - Introduces a delay inside the execution of the firmware.

Parameters

delay	- time to delay expressed in 10 microseconds. Delay10us(100) = delay of 1 millisecond.

Returns

None

Warning

this command can assure a real delay ONLY if included inside a critical section.

2.1.3.2 void DelayMs (WORD *ms*)

DelayMs - Introduces a delay inside the execution of the firmware.

Parameters

delay	- time to delay expressed in 1ms. DelayMS(100) = delay of 0.1 second.

Returns

None

Warning

this command can assure a real delay ONLY if included inside a critical section.

2.1.3.3 void vTaskDelay (int *delay*)

vTaskDelay - Delays the Flyport task and executes the TCP task. This delay is the preferred way to delay the task, because optimizes the time the microcontroller dedicates to any task.

Parameters

delay	- time to delay expressed in 10ms. vtaskDelay(100) = delay of one second.

Returns

None

2.1.3.4 void vTaskSuspendAll ( )

vTaskSuspendAll - this function suspend any other working task on the Flyport. So enters in the such called **critical section**. The OS resident in the Flyport can accomplish different tasks (in particular, your task and the task with the TCP/IP and WiFi stack), but if you want to execute ONLY your Task, you must enter into the critical section.

Parameters

None	

Returns

None

Warning

when inside the crtitical section, the stack will not work, so use it with caution. It can be useful if you have some time critical operation to do, or some delay you must strict respect. Once you issue this command you MUST ALWAYS exit from the critical section. Otherwise, the WiFi will stop working.

2.1.3.5 void xTaskResumeAll ( )

xTaskResumeAll - exits from the **critical section**.

Parameters

None	

Returns

None

2.2 ARPlib stack

Functions

- BYTE ARPResolveMAC (char ipaddr[])

2.2.1 Detailed Description

ARP provides the commands to manage ARP-IP association. With ARPResolveMAC is possible to force an arp request, but usually this operation in managed by TCP-IP Stack.

2.2.2 Function Documentation

2.2.2.1 BYTE ARPResolveMAC (char *ipaddr[]*)

ARPResolveMAC - Force an arp request for specific IP address

Parameters

ipaddr	IP address

2.3 FTPlib stack

Functions

- TCP_SOCKET FTPClientOpen (char ftpaddr[], char ftpport[])
- void FTPRead (TCP_SOCKET ftpsockread, char ftpreadch[], int ftprlen)
- WORD FTPWrite (TCP_SOCKET ftpsockwr, BYTE *ftpstrtowr, int ftpwlen)
- void FTPClose (TCP_SOCKET Sockclose)
- BOOL FTPisConn (TCP_SOCKET sockcon)
- WORD FTPRxLen (TCP_SOCKET ftpsocklen)

2.3.1 Detailed Description

FTP provides Internet communication abilities.

2.3.2 Function Documentation

2.3.2.1 TCP_SOCKET FTPClientOpen (char *ftpaddr[]*, char *ftpport[]*)

FTPClientOpen - Creates a FTP client on specified IP address and port

Parameters

ftpaddr	- IP address of the remote server. Example: "192.168.1.100" (the char array must be NULL terminated).
ftpport	- Port of the remote server to connect. Example: "1234" (the char array must be NULL terminated).

Returns

- INVALID_SOCKET: the operation was failed. Maybe there are not available sockets.
- A TCP_SOCKET handle to the created socket. It must be used to access the socket in the program (read/write operations and close socket).

2.3.2.2 void FTPClose (TCP_SOCKET *Sockclose*)

FTPClose - Closes the client socket specified by the handle.

Parameters

Sockclose	- The handle of the socket to close (the handle returned by the command FTPClientOpen).

Returns

None.

2.3.2.3 BOOL FTPisConn (TCP_SOCKET *sockcon*)

FTPisConn - Verifies the connection of a remote FTP device with the socket.

Parameters

sockcon	- The handle of the socket to control (the handle returned by the command FTPOpen).

Returns

TRUE - The remote connection is established.
FALSE - The remote connection is not established.

2.3.2.4 void FTPRead (TCP_SOCKET *ftpsockread,* char *ftpreadch[ ],* int *ftprlen*)

FTPRead - Reads the specified number of characters from a FTP socket and puts them into the specified char array

Parameters

ftpsockread	- The handle of the socket to read (the handle returned by the command FTPClientOpen).
ftpreadch	- The char array to fill with the read characters.
ftprlen	- The number of characters to read.

Warning

The length of the array must be AT LEAST = ftprlen+1, because at the end of the operation the array it's automatically NULL terminated (is added the '\0' character).

Returns

None.

2.3.2.5 WORD FTPRxLen (TCP_SOCKET *ftpsocklen*)

FTPRxLen - Verifies how many bytes can be read from the specified FTP socket.

Parameters

ftpsocklen	- The handle of the socket to control (the handle returned by the command FTPOpen).

Returns

The number of bytes available to be read.

2.3.2.6 WORD FTPWrite (TCP_SOCKET *ftpsockwr,* BYTE * *ftpstrtowr,* int *ftpwlen*)

FTPWrite - Writes an array of characters on the specified socket.

Parameters

ftpsockwr	- The socket to which data is to be written (it's the handle returned by the command TCPClient-Open or TCPServerOpen).
ftpstrtowr	- Pointer to the array of characters to be written.
ftpwlen	- The number of characters to write.

Returns

The number of bytes written to the socket. If less than ftpwlen, the buffer became full or the socket is not conected.

2.4 ADC

Functions

- void ADCInit ()
- int ADCVal (int ch)

2.4.1 Detailed Description

The ADC library contains the command to manage the ADC, so it's possible to easy convert an analog voltage value. The precision of the ADC is 10 bit.

2.4.2 Function Documentation

2.4.2.1 void ADCInit ( )

ADCInit - initializes the ADC module with default values. **NOTE:** this function is already called at Flyport startup.

Parameters

None	

Returns

None

2.4.2.2 int ADCVal (int *ch*)

ADCVal - Reads the value of the analog channel specified.

Parameters

ch	- the number of the analog channel to read. For the number of the channel, refer to the Flyport pinout.

Returns

the value read by the function.

2.5 GPIOs

Functions

- void IOPut (int io, int putval)
- void IOInit (int io, int putval)
- int IOGet (int io)
- int IOButtonState (int io)

2.5.1 Detailed Description

The GPIOs commands can be used the manage the IO pins, to configure, to change or to read their values.

2.5.2 Function Documentation

2.5.2.1 int IOButtonState (int *io*)

IOButtonState - Polls for the state of the button implemented on the specified pin. This command doesn't return the voltage level of the pin, but if the button has been pressed or released. No problem with the "debounce" of the button. It doesn't matter if the button is implemented with a "low logic" or "high logic". You just have to initialize the pin like "inup" or "indown".

Parameters

io	- the pin to read.

Returns

- **pressed:** if the button has been pressed.
- **released:** if the button has been released.

2.5.2.2 int IOGet (int *io*)

IOGet - Reads the value of the specified pin.

Parameters

io	- the pin to read.

Returns

the value read by the function.

2.5.2.3 void IOInit (int *io,* int *putval*)

IOInit - Initializes the specified pin like output, input, input with pull-up or pull-down resistor.

Parameters

io	- specifies the pin.
putval	- specifies how the pin must be initialized. The valid parameters are the following: • **in (or IN)** input pin. • **inup (or INUP)** input pin with pullup resistor (about 5 KOhm). • **indown (or INDOWN)** input pin with pulldown resistor (about 5 Kohm). • **out (or OUT)** output pin. • **UART1RX** UART1 RX input pin. • **UART1CTS** UART1 CTS input pin. • **UART2RX** UART2 RX input pin. • **UART2CTS** UART2 CTS input pin. • **UART3RX** UART3 RX input pin. • **UART3CTS** UART3 CTS input pin. • **UART4RX** UART4 RX input pin. • **UART4CTS** UART4 CTS input pin. • **EXT_INT2** External Interrupt 2 input pin. • **EXT_INT3** External Interrupt 3 input pin. • **EXT_INT4** External Interrupt 4 input pin. • **SPICLKIN** SPI clock input pin (only in slave mode). • **SPI_IN** SPI data input pin. • **SPI_SS** SPI slave select input pin (only in slave mode). • **TIM_4_CLK** External Timer 4 input pin. • **UART1TX** UART1 TX output pin. • **UART1RTS** UART1 RTS output pin. • **UART2TX** UART2 TX output pin. • **UART2RTS** UART2 RTS output pin. • **UART3TX** UART3 TX output pin. • **UART3RTS** UART3 RTS output pin. • **UART4TX** UART4 TX output pin. • **UART4RTS** UART4 RTS output pin. • **SPICLKOUT** SPI clock output pin (only in master mode). • **SPI_OUT** SPI data output pin. • **SPI_SS_OUT** SPI slave select output pin (only in master mode).

Returns

None

2.5.2.4 void IOPut (int *io,* int *putval*)

IOPut - Puts the putval value on the specified IO output pin.

Parameters

io	- specifies the ouput pin.
putval	- the value to assign to the pin: • **on (or ON, or 1)** high level (about 3.3V). • **off (or OFF, or 0)** low level (0V).

Returns

None

2.6 UART

Functions

- void UARTInit (int port, long int baud)
- void UARTOn (int port)
- void UARTOff (int port)
- void UARTFlush (int port)
- int UARTBufferSize (int port)
- int UARTRead (int port, char *towrite, int count)
- void UARTWrite (int port, char *buffer)
- void UARTWriteCh (int port, char chr)

2.6.1 Detailed Description

The UART section provides serial communication. The flyport implements a buffer of 256 characters for the UART, to make serial communicate easier.

2.6.2 Function Documentation

2.6.2.1 int UARTBufferSize (int *port*)

UARTBufferSize - Returns the RX buffer size of the specified UART port.

Parameters

port	- the UART port to read.

Returns

the number of characters that can be read from the specified serial port.

2.6.2.2 void UARTFlush (int *port*)

UARTFlush - Flushes the buffer of the specified UART port.

Parameters

port	- the UART port to flush.

Returns

None

2.6.2.3 void UARTInit (int *port*, long int *baud*)

UARTInit - Initializes the specified uart port with the specified baud rate.

Parameters

port	- the UART port to initialize. **Note:** at the moment the Flyport Framework supports just one UART, but the hardware allows to create up to four UARTs. Others will be added in next release, however is possible to create them with standard PIC commands.
baud	- the desired baudrate.

Returns

None

2.6.2.4 void UARTOff (int *port*)

UARTOff - Turns off the specified UART port.

Parameters

port	- the UART port to turn off.

Returns

None

2.6.2.5 void UARTOn (int *port*)

UARTOn - After the initialization, the UART must be turned on with this command.

Parameters

port	- the UART port to turn on.

Returns

None

2.6.2.6 int UARTRead (int *port,* char * *towrite,* int *count*)

UARTRead - Reads characters from the UART RX buffer and put them in the char pointer "towrite" . Also returns the report for the operation.

Parameters

port	- the UART port to read.
towrite	- the char pointer to fill with the read characters.
count	- the number of characters to read

Returns

the report for the operation:

- <Bn>0: N characters correctly read.
- **n<0:** N characters read, but buffer overflow detected.

2.6.2.7 void UARTWrite (int *port,* char * *buffer*)

UARTWrite - writes the specified string on the UART port.

Parameters

port	- the UART port to write to.
buffer	- the string to write (a NULL terminated char array).

Returns

None

2.6.2.8 void UARTWriteCh (int *port*, char *chr*)

UARTWriteCh - writes a single character on the UART port.

Parameters

port	- the UART port to write to.
char	- the char to write.

Returns

None

2.7 PWM

Functions

- void PWMInit (BYTE pwm, float freq, float dutyc)
- void PWMOn (BYTE io, BYTE pwm)
- void PWMDuty (float duty, BYTE pwm)
- void PWMOff (BYTE pwm)

2.7.1 Detailed Description

With the PWM library is possible to easily manage up to nine different PWM (different frequency and duty cycle). To use the PWM on a pin you first have to initialize the PWM, then assign it to the desired pin. At runtime it' possible to change the duty cycle of the PWM.

NOTE :it possible to assign a PWM to any pin, also to the input pins.

2.7.2 Function Documentation

2.7.2.1 void PWMDuty (float *duty,* BYTE *pwm*)

PWMDuty - changes the duty cycle of the PWM without turning it off. Useful for motors or dimmers.

Parameters

duty	- new duty cycle desired (0-100).
pwm	- PWM number previously defined in PWMInit.

Returns

None

2.7.2.2 void PWMInit (BYTE *pwm,* float *freq,* float *dutyc*)

PWMInit - initializes the specified PWM with the desired frequency and duty cycle .

Parameters

freq	- frequency in hertz.
dutyc	- ducty cycle for the PWM in percent (0-100).

Returns

None

2.7.2.3 void PWMOff (BYTE *pwm*)

PWMOff - turns off the specified PWM.

Parameters

pwm	- PWM number previously defined in PWMInit.

Returns

None

2.7.2.4 void PWMOn (BYTE *io*, BYTE *pwm*)

PWMOn - turns on the specified PWM on the specified pin.

Parameters

io	- pin to assign the PWM.
pwm	- PWM number previously defined in PWMInit.

Returns

None

2.8 I2C

Functions

- void I2CInit (BYTE I2CSpeed)
- void I2CStart ()
- void I2CRestart ()
- void I2CStop ()
- void I2CWrite (BYTE data)
- BYTE I2CRead (BYTE ack)

2.8.1 Detailed Description

The I2C library allows the user to communicate with external devices with I2C bus, like flash memories or sensors. The Flyport is initialized as I2C master.

2.8.2 Function Documentation

2.8.2.1 void I2CInit (BYTE *I2CSpeed*)

I2CInit - Initializes the I2C module.

Parameters

I2CSpeed	- can be used HIGH_SPEED (400kHz) or LOW_SPEED (100kHz)

Returns

None

2.8.2.2 BYTE I2CRead (BYTE *ack*)

I2CRead - Reads one byte from the data bus.

Parameters

ack	the value of the Acknowledge to send (0 or 1)

Returns

the value read by the function.

2.8.2.3 void I2CRestart ( )

I2CRestart - Sends a repeated start sequence on the bus .

Parameters

None	

Returns

None

2.8.2.4 void I2CStart ( )

I2CStart - Sends a start sequence on the bus.

Parameters

None	

Returns

None

2.8.2.5 void I2CStop ( )

I2CStop - Stops the trasmissions on the bus.

Parameters

None	

Returns

None

2.8.2.6 void I2CWrite (BYTE *data*)

I2CWrite - writes one byte on the bus.

Parameters

data	- the data to write

Returns

None

2.9 Interrupts

Functions

- void INTDetach (int intNum)
- void INTAttach (int intNum, void functionName())
- void INTInit (int intNum, void functionName(), BOOL mode)
- void INTPriority (int intNum, BYTE priority)
- void INTEnable (int intNum)
- void INTDisable (int intNum)

2.9.1 Detailed Description

The INT library contains the command to manage the External Interruts

2.9.2 Function Documentation

2.9.2.1 void INTAttach (int *intNum,* void *functionName()*)

INTAttach - Attaches a generic user function to the External Interrupt.

Parameters

intNum	- the number of external interrupt peripheral (2, 3 or 4).
functionName	- the name of the user function to execute.

Returns

None

2.9.2.2 void INTDetach (int *intNum*)

INTDetach - Detaches the function assigned to the External Interrupt.

Parameters

intNum	- the number of external interrupt peripheral (2, 3 or 4).

Returns

None

2.9.2.3 void INTDisable (int *intNum*)

INTDisable - Disables the External Interrupt.

Parameters

intNum	- the number of external interrupt peripheral (2, 3 or 4).

Returns

None

2.9.2.4 void INTEnable (int *intNum*)

INTEnable - Enables the External Interrupt.

Parameters

intNum	- the number of external interrupt peripheral (2, 3 or 4).

Returns

None

2.9.2.5 void INTInit (int *intNum,* void *functionName(),* BOOL *mode*)

INTInit - Initialize the External Interrupt.

Parameters

intNum	- the number of external interrupt peripheral (2, 3 or 4).
functionName	- the name of the user function to execute.
mode	- the edge mode for interrupt execution (0 on positive edge, 1 on negative edge).

Returns

None

2.9.2.6 void INTPriority (int *intNum,* BYTE *priority*)

INTPriority - Changes the External Interrupt Priority.

Parameters

intNum	- the number of external interrupt peripheral (2, 3 or 4).
priority	- the new priority to use (0->very low, up to 7->very high).

Returns

None

2.10 NETlib stack

Functions

- void NETCustomSave ()
- void NETCustomDelete ()
- BOOL NETCustomExist ()
- void NETCustomLoad ()
- void NETSetParam (int paramtoset, char paramstring[])
- void ETHRestart (int ethprofile)
- void WFConnect (int pconn)
- void WFDisconnect ()
- void WFScan ()
- tWFNetwork WFScanList (int ntscn)
- void WFStopConnecting ()
- void WFSetSecurity (BYTE mode, char *keypass, BYTE keylen, BYTE keyind)
- void WFHibernate ()
- void WFSleep ()
- void WFOn ()
- void WFPsPollEnable (BOOL ps_active)

Variables

- APP_CONFIG **NETConf** []

2.10.1 Detailed Description

Introduction

The **NET library** contains all the functions to manage the Ethernet and WiFi stack. It's possible to connect or disconnect from a network, change the settings, and save them inside the falsh memory of the device, to recall at the startup.

Connection profiles

A *connection profile* contains all the information to connect to or to create a network. The Flyport modules have two possible profile to use: a DEFAULT one (fixed in firmware), and a CUSTOM (this profile can be modified at runtime). **WiFi Profiles:**

- **WF_DEFAULT**: contains all the values set in the IDE, with the **TCP/IP Setup**.
- **WF_CUSTOM**: it's the customizable profile. The user can change any parameter and save them in the flash memory of the Flyport. At the device startup it is the same of WF_DEFAULT

Ethernet Profiles:

- **ETH_DEFAULT**: contains all the values set in the IDE, with the **TCP/IP Setup**.
- **ETH_CUSTOM**: it's the customizable profile. The user can change any parameter and save them in the flash memory of the Flyport. At the device startup it is the same of ETH_DEFAULT

2.10.2 Function Documentation

2.10.2.1 void ETHRestart (int *ethprofile*)

ETHRestart - Connects the Flyport Ethernet Module to the network with the specified profle.

Parameters

pconn	- Specifies the profile used to connect to the network. The following profile are available: ETH_DEFAULT : uses the settings choosen in the IDE TCP/IP setup. This profile cannot be changed. ETH_CUSTOM : this profile can be customized by the user and even saved in the flash memory. At the startup is identical to the default, but you can change it anytime you want.

Returns

None

2.10.2.2 void NETCustomDelete ( )

NETCustomDelete - Deletes the custom settings for the network profile CUSTOM.

Returns

None

2.10.2.3 BOOL NETCustomExist ( )

NETCustomExist - Verifies if in memory is present some data for the CUSTOM profile. It can be useful at the startup of the device, because it's possible to control if in a previous session (before any power off)has been saved some configuration data.

Returns

FALSE: no data present.
TRUE: valid data present.

2.10.2.4 void NETCustomLoad ( )

NETCustomLoad - Loads from the flash memory the previously set parameters for the CUSTOM profile.

Returns

None

2.10.2.5 void NETCustomSave ( )

NETCustomSave - When the Flyport is powered down, all the changes on the CUSTOM profile will be lost, because are stored in RAM. To prevent the loosing of all the data, you can use the command WFCustomSave. It saves all the network parameters for the CUSTOM connection profile. No need to set anything, the data will be saved in a reserved part of the flash memory, so it will be available also if you power off the Flyport.

Parameters

None	

Returns

None

2.10.2.6 void NETSetParam (int *paramtoset,* char *paramstring[ ]*)

NETSetParam - With this command is possible to change any network parameter of the CUSTOM profile.

Parameters

paramtoset	- the parameter to change.
paramstring	- value of the parameter. **Note:** Some parameters are available only for Flyport WiFi module.

Returns

None.

Sintax of the command to set the parameters:

- **IP address of the device:** NETSetParam(MY_IP_ADDR , string with the IP address, for example "192.168.-1.100")
- **Primary DNS server:** NETSetParam(PRIMARY_DNS , string with the IP address, for example "192.168.1.1")
- **Secondary DNS server:** NETSetParam(SECONDARY_DNS , string with the IP address, for example "192.-168.1.1")
- **Default gateway:** NETSetParam(MY_GATEWAY , string with the IP address, for example "192.168.1.1")
- **Subnet mask:** NETSetParam(SUBNET_MASK , string with the subnet mask, for example "255.255.255.0")
- **Netbios name:** NETSetParam(NETBIOS_NAME , string with the Netbios name, for example "Flyport")
- **DHCP client enabled or not:** NETSetParam(DHCP_ENABLE , ENABLED or DISABLED)
- **SSID:** NETSetParam(SSID_NAME , string with the SSID name, for example "FlyportNet") **Note:** Only for Flyport WiFi!
- **Network type (adhoc or infrastructure):** NETSetParam(NETWORK_TYPE , ADHOC or INFRASTRUCTU-RE) **Note:** Only for Flyport WiFi!

2.10.2.7 void WFConnect (int *pconn*)

WFConnect - Connects the Flyport WiFi Module to the network with the specified profle.

Parameters

pconn	- Specifies the profile used to connect to the network. The following profile are available: W-F_DEFAULT : uses the settings choosen in the IDE TCP/IP setup. This profile cannot be changed. WF_CUSTOM : this profile can be customized by the user and even saved in the flash memory. At the startup is identical to the default, but you can change it anytime you want.

Returns

None

2.10.2.8 void WFDisconnect ( )

WFDisconnect - Disconnects the device from the network. No parameters required.

Parameters

None	

Returns

None

2.10.2.9 void WFHibernate ( )

WFHibernate - Enables Hibernate mode on the MRF24WB0M, which effectively turns off the WiFi. The microcontroller is still completely running, so all the modules are working (UART, SPI, analog inputs...). To turn on again the module, use the command WFOn().

Parameters

None	

Returns

None

2.10.2.10 void WFOn ( )

WFOn - Turns on the WiFi module after a WFSleep() or WFHibernate() command.

Parameters

none	

Returns

None

2.10.2.11 void WFPsPollEnable (BOOL *ps_active*)

WFPsPollEnable - Enables or Disable Power-Save Pool at runtime

Parameters

ps_active	TRUE to enable Poll mode for longer battery life or FALSE to disable PS Poll mode, MRF24-WB0M will stay active and not go sleep.

Returns

None

2.10.2.12 void WFScan ( )

WFScan - Starts a scan to detect all the WiFi networks available. The function doesn't return anything. When it has finished, it generates an event, you can catch in the WiFi events file.

Parameters

None	

Returns

None. The number of the retrieved WiFi networks found is passed in the function WF_events like an "event info". All the data for the retrieved networks can be accessed with the command WFScanList.

Attention

The command must be issued when the device is not connected to any network, otherwise it won't give any error message, but it won't work.

2.10.2.13 tWFNetwork WFScanList (int *ntscn*)

WFScanList - This command must be issued after a WFScan request has been completed (so the event is generated). The WFScan command returns in the event handler the number of the WiFi networks found, but all the data related to the networks, can be accessed using the function WFScanList.

Parameters

ntscn	- number of the wifi network (1 to number of the found networks).

Returns

A tWFNetwork structure, which contains all the informations about the specified network.

Warning

The function can't be called inside the WiFi Event handler. You always must call it from the FlyportTask.

2.10.2.14 void WFSetSecurity (BYTE *mode,* char * *keypass,* BYTE *keylen,* BYTE *keyind*)

WFSetSecurity - This command is used to set all the security parameters for the WF_CUSTOM connection profile

Parameters

mode	- the security mode. Valid security mode are the following: • **WF_SECURITY_OPEN:** no security. • **WF_SECURITY_WEP_40:** WEP security, with 40 bit key. • **WF_SECURITY_WEP_104:** WEP security, with 104 bit key. • **WF_SECURITY_WPA_WITH_KEY:** WPA-PSK personal security, the user specifies the hex key. • **WF_SECURITY_WPA_WITH_PASS_PHRASE:** WPA-PSK personal security, the user specifies only the passphrase. • **WF_SECURITY_WPA2_WITH_KEY:** WPA2-PSK personal security, the user specifies the hex key. • **WF_SECURITY_WPA2_WITH_PASS_PHRASE:**WPA2-PSK personal security, the user specifies only the passphrase. • **WF_SECURITY_WPA_AUTO_WITH_KEY:** WPA-PSK personal or WPA2-PSK personal (the Flyport will auto select the mode) with hex key. • **WF_SECURITY_WPA_AUTO_WITH_PASS_PHRASE:** WPA-PSK personal or WPA2--PSK personal (autoselect) with pass phrase.
keypass	- the key or passphrase for the network. A key must be specified also for open connections (you can put a blank string, like "").
keylen	- length of the key/passphrase. Must be specified also for open connections (can be 0).
keyind	- index of the key (used only for WEP security, but must be specified also for all others, in that case, can be 0).

Attention

For WPA/WPA2 with passphrase, the Flyport must calculate the hex key. The calculation is long and difficult, so it will take about 20 seconds to connect!

Returns

None.

2.10.2.15 void WFSleep ( )

WFSleep - Turns off the WiFi module and put the microcontroller in sleep mode, to obtain maximum power saving mode.

Parameters

none	

Returns

None

Warning

: After the command is issued, no the module is completely in sleep mode, no modules are active (UART, SPI, analog inputs...). The only way to awake the module is issue an external interupt. So before the WFSleep command, an interrupt must be set.

2.10.2.16 void WFStopConnecting ( )

WFStopConnecting - When the command WFConnect is lauched, the device tries to connect to the selected WiFi network until it doesn't find it. If you want to stop the retries, you have to issu the command WFStopConnecting.

Parameters

None	

Returns

None

2.11 SMTPlib stack

Functions

- BOOL SMTPStart ()
- void SMTPSetServer (int servparam, char ∗paramfield)
- void SMTPSetMsg (int msgparam, char ∗mparamfield)
- BOOL SMTPSend ()
- BOOL SMTPBusy ()
- BOOL SMTPStop ()
- WORD SMTPReport ()

2.11.1 Detailed Description

The SMTP commands allows to set the parameters for the SMTP connection, and to send emails from the Flyport.

2.11.2 Function Documentation

2.11.2.1 BOOL SMTPBusy ( )

SMTPBusy - Verifies if the SMTP client is busy perfoming some action. Until the client is busy is not possible to other SMTP commands.

Parameters

None	

Returns

TRUE: the client is busy.
FALSE: the client is not performing any action, and can be used by the firmware.

2.11.2.2 WORD SMTPReport ( )

SMTPReport - Closes the SMTP client and gives a report about email sending, reporting the SMTP server response code. To send another email SMTP must be restarted.

Parameters

None	

Returns

the server answer code received by Flyport.

2.11.2.3 BOOL SMTPSend ( )

SMTPSend - Once all the connection and message settings are done, it's possible to send the desired email message with this command

Parameters

None	

Returns

TRUE: the message was correctly sent.
FALSE: there was an error. Maybe the SMTP client is still busy in sending another message, or it's not initialized.

2.11.2.4 void SMTPSetMsg (int *msgparam,* char * *mparamfield*)

SMTPSetMsg - Sets all the parameters for the email message to send (the sender, the destination address, the cc and bcc addresses, email subject and body).

Parameters

msgparam	the parameter to set. Available list is the following: • **MSG_TO:** the destination address. • **MSG_CC:** the cc address. • **MSG_BCC:** the bcc address. • **MSG_FROM:** the sender of the email message. • **MSG_SUBJECT:** the object of the email message. • **MSG_BODY:** the body of the email message.
mparamfield	a string containing the associated value to the parameter. For example: if we want to send a message to example@openpicus.com. SMTPSetMsg(MSG_TO , "example@openpicus.com")

Returns

None.

2.11.2.5 void SMTPSetServer (int *servparam,* char * *paramfield*)

SMTPSetServer - Sets all the parameters for the SMTP remote server (server name, username, password and port).

Parameters

servparam	the parameter to set. Available list is the following: • **SERVER_NAME:** the name of the SMTP server. • **SERVER_USER:** the username for access to the server. • **SERVER_PASS:** the password associated to the username. • **SERVER_PORT:** the port of the SMTP server.
paramfield	a string containing the associated value to the parameter (server name, username, password or SMTP port). For example: SMTPSetServer(SERVER_NAME , "smtp.serverxyz.com")

Returns

None.

2.11.2.6 BOOL SMTPStart ( )

SMTPStart - Initializes the SMTP module. If the client is already in use, the initialization will not be possible.

Parameters

None	

Returns

TRUE: the SMTP client was correctly initialized.
FALSE: the SMTP client wasn't initialized. Maybe it's already used

2.11.2.7 BOOL SMTPStop ( )

SMTPStop (deprecated) - Closes the SMTP client and gives a report about email sending. To send another email SMTP must be restarted.

Parameters

None	

Returns

TRUE: the email was correctly sent.
FALSE: an error occured sending the email.

2.12 TCPlib stack

Functions

- void TCPServerDetach (TCP_SOCKET sockdet)
- TCP_SOCKET TCPClientOpen (char tcpaddr[], char tcpport[])
- TCP_SOCKET TCPServerOpen (char tcpport[])
- void TCPRead (TCP_SOCKET socktoread, char readch[], int rlen)
- void TCPpRead (TCP_SOCKET socktoread, char readch[], int rlen, int start)
- WORD TCPWrite (TCP_SOCKET socktowrite, char *writech, int wlen)
- NODE_INFO TCPRemote (TCP_SOCKET remotesock)
- void TCPClientClose (TCP_SOCKET Sockclose)
- void TCPServerClose (TCP_SOCKET Sockclose)
- BOOL TCPisConn (TCP_SOCKET sockconn)
- void TCPRxFlush (TCP_SOCKET sockflush)
- WORD TCPRxLen (TCP_SOCKET socklen)

2.12.1 Detailed Description

The TCP library contains all the command to manage the TCP sockets. If you need to use a TCP client or server inside your application, this is the correct section.

2.12.2 Function Documentation

2.12.2.1 void TCPClientClose (TCP_SOCKET *Sockclose*)

TCPClientClose - Closes the client socket specified by the handle.

Parameters

Sockclose	- The handle of the socket to close (the handle returned by the command TCPClientOpen).

Returns

None.

2.12.2.2 TCP_SOCKET TCPClientOpen (char *tcpaddr[]*, char *tcpport[]*)

TCPClientOpen - Creates a TCP client on specified IP address and port

Parameters

tcpaddr	- IP address of the remote server. Example: "192.168.1.100" (the char array must be NULL terminated).
tcpport	- Port of the remote server to connect. Example: "1234" (the char array must be NULL terminated).

Returns

- INVALID_SOCKET: the operation was failed. Maybe there are not available sockets.
- A TCP_SOCKET handle to the created socket. It must be used to access the socket in the program (read/write operations and close socket).

2.12.2.3 BOOL TCPisConn (TCP_SOCKET *sockconn*)

TCPisConn - Verifies the connection of a remote TCP device with the socket. It can be useful to catch an incoming new connection to a TCP server.

Parameters

sockconn	- The handle of the socket to control (the handle returned by the command TCPServerOpen).

Returns

TRUE - The remote connection is established.
FALSE - The remote connection is not established.

2.12.2.4 void TCPpRead (TCP_SOCKET *socktoread,* char *readch[ ],* int *rlen,* int *start*)

TCPpRead - Reads the specified number of characters from a TCP socket and puts them into the specified char array. **NOTE:** This function does flush the buffer!

Parameters

socktoread	- The handle of the socket to read (the handle returned by the command TCPClientOpen or TCPServerOpen).
readch	- The char array to fill with the read characters.
rlen	- The number of characters to read.
start	- The starting point to read.

Warning

The length of the array must be AT LEAST = rlen+1, because at the end of the operation the array it's automatically NULL terminated (is added the '\0' character).

Returns

None.

2.12.2.5 void TCPRead (TCP_SOCKET *socktoread,* char *readch[ ],* int *rlen*)

TCPRead - Reads the specified number of characters from a TCP socket and puts them into the specified char array. **NOTE:** This function flushes the buffer after reading!

Parameters

socktoread	- The handle of the socket to read (the handle returned by the command TCPClientOpen or TCPServerOpen).
readch	- The char array to fill with the read characters.
rlen	- The number of characters to read.

Warning

The length of the array must be AT LEAST = rlen+1, because at the end of the operation the array it's automatically NULL terminated (is added the '\0' character).

Returns

None.

2.12.2.6 NODE_INFO TCPRemote (TCP_SOCKET *remotesock*)

TCPRemote - Returns IPaddress and MAC of the remote device

Parameters

remotesock	- The socket of the TCP connection

Returns

NODE_INFO struct with IPAddr and MACAddr elements

2.12.2.7 void TCPRxFlush (TCP_SOCKET *sockflush*)

TCPRxFlush - Empty specified TCP socket RX Buffer.

Parameters

sockflush	- The handle of the socket to empty (the handle returned by the command TCPClientOpen or TCPServerOpen).

Returns

none.

2.12.2.8 WORD TCPRxLen (TCP_SOCKET *socklen*)

TCPRxLen - Verifies how many bytes can be read from the specified TCP socket.

Parameters

socklen	- The handle of the socket to control (the handle returned by the command TCPClientOpen or TCPServerOpen).

Returns

The number of bytes available to be read.

2.12.2.9 void TCPServerClose (TCP_SOCKET *Sockclose*)

TCPServerClose - Closes the server socket specified and destroys the handle. Any remote client connected with the server will be disconnected.

Parameters

Sockclose	- The handle of the socket to close (the handle returned by the command TCPServerOpen).

Returns

None.

2.12.2.10 void TCPServerDetach (TCP_SOCKET *sockdet*)

TCPServerDetach - Detaches the remote client from the server

Parameters

sockdet	- Socket of the server (the handle returned by the command TCPServerOpen).

Returns

None

2.12.2.11 TCP_SOCKET TCPServerOpen (char *tcpport[]*)

TCPServerOpen - Creates a TCP server on specified port

Parameters

tcpport	- Number of the port for the server. Example: "1234" (the array must be NULL terminated).

Returns

- INVALID_SOCKET: the operation was failed. Maybe there are not available sockets.
- A TCP_SOCKET handle to the created socket. It must be used to access the socket in the program (read/write operations and close socket).

2.12.2.12 WORD TCPWrite (TCP_SOCKET *socktowrite,* char * *writech,* int *wlen*)

TCPWrite - Writes an array of characters on the specified socket.

Parameters

socktowrite	- The socket to which data is to be written (it's the handle returned by the command TCPClient-Open or TCPServerOpen).
writech	- Pointer to the array of characters to be written.
wlen	- The number of characters to write.

Returns

The number of bytes written to the socket. If less than len, the buffer became full or the socket is not conected.

2.13 UDPlib stack

Functions

- WORD UDPLocalPort (BYTE udplocalsocket)
- BYTE UDPServerOpen (char udpport[])
- BYTE UDPClientOpen (char *udpaddr, char udpport[])
- BYTE UDPBroadcastOpen (char udpport[])
- BYTE UDPServerClose (BYTE sock)
- BYTE UDPClientClose (BYTE sock)
- WORD UDPRxLen (BYTE sock)
- void UDPRxFlush (BYTE sock)
- BOOL UDPRxOver (BYTE sock)
- int UDPRead (BYTE sock, char str2rd[], int lstr)
- int UDPpRead (BYTE sock, char str2rd[], int lstr, int start)
- WORD UDPWrite (BYTE sockwr, BYTE *str2wr, int lstr)

2.13.1 Detailed Description

UDP provides the commands to manage UDP connections. Flyport supports the creation of two different UDP socket, with their own RX buffer.

2.13.2 Function Documentation

2.13.2.1 BYTE UDPBroadcastOpen (char *udpport[]*)

UDPBroadcastOpen - Create a UDP broadcast on specified port

Parameters

udpport	Remote Port for UDP

Returns

The number of current socket or 0 if an error occured during the opening of the socket.

2.13.2.2 BYTE UDPClientClose (BYTE *sock*)

UDPClientClose - Closes UDP Client socket

Parameters

sock	UDP Socket number

2.13.2.3 BYTE UDPClientOpen (char * *udpaddr,* char *udpport[]*)

UDPClientOpen - Create a UDP client on specified port, try more time for arp request.

Parameters

udpaddr	IP address of server
udpport	Remote Port of UDP server

Returns

The number of current socket or 0 if an error occured during the opening of the socket.

2.13.2.4 WORD UDPLocalPort (BYTE *udplocalsocket*)

UDPLocalPort - Returns the local port of the specified UDP socket. It can be useful when using a socket as client.

Parameters

udplocalsocket	the number of the UDP socket.

Returns

The number of current socket, or 0 if an error occured during the opening of the socket.

2.13.2.5 int UDPpRead (BYTE *sock,* char *str2rd[],* int *lstr,* int *start*)

UDPpRead - Reads lstr bytes from the RX buffer without clear it

Parameters

sock	UDP socket number
str2rd	Buffer for data
lstr	lenght of string
start	point of start reading (in byte)

Returns

The number of read characters from the specified UDP socket.

2.13.2.6 int UDPRead (BYTE *sock,* char *str2rd[],* int *lstr*)

UDPRead - Reads lstr bytes from the RX buffer

Parameters

sock	UDP socket number
str2rd	Buffer for data
lstr	lenght of string

Returns

The number of read characters from the specified UDP socket.

2.13.2.7 void UDPRxFlush (BYTE *sock*)

UDPRxFlush - Empty the RX buffer

Parameters

sock	UDP socket number

Returns

none

2.13.2.8 WORD UDPRxLen (BYTE *sock*)

UDPRxLen - Reads the length of the RX buffer

Parameters

sock	UDP socket number

Returns

The number of char that can be read from the UDP buffer.

2.13.2.9 BOOL UDPRxOver (BYTE *sock*)

UDPRxOver - Checks if a overflow was reached in UDP RX buffer, and clear the flag.

Parameters

sock	UDP socket number

Returns

0 = no overflow, 1 = overflow reached

2.13.2.10 BYTE UDPServerClose (BYTE *sock*)

UDPServerClose - Closes UDP Server socket

Parameters

sock	UDP Socket number

2.13.2.11 BYTE UDPServerOpen (char *udpport[]*)

UDPServerOpen - Create a UDP server on specified port

Parameters

udpport	Local Port for UDP server

Returns

The number of current socket, or 0 if an error occured during the opening of the socket.

2.13.2.12 WORD UDPWrite (BYTE *sockwr,* BYTE * *str2wr,* int *lstr*)

UDPWrite - Writes on the UDP socket

Parameters

sockwr	UDP socket number
str2wr	String to write
lstr	String lenght

Returns

The number of write characters to the specified UDP socket.

2.14 System

Modules

- Delays

2.15 Net

Modules

- ARPlib stack
- FTPlib stack
- NETlib stack
- SMTPlib stack
- TCPlib stack
- UDPlib stack

2.16 Hardware

Modules

- ADC
- GPIOs
- UART
- PWM
- I2C
- Interrupts

Index