

open*f*icus

OpenPicus - Flyport

Wed Apr 27 2011 17:38:46

Contents

1	OpenPicus	1
2	Module Index	3
2.1	Modules	3
3	Data Structure Index	5
3.1	Data Structures	5
4	File Index	7
4.1	File List	7
5	Module Documentation	9
5.1	Delays	9
5.1.1	Detailed Description	9
5.1.2	Delays	9
5.1.3	Function Documentation	9
5.1.3.1	DelayMs	9
5.1.3.2	DelayUs	10
5.1.3.3	vTaskDelay	10
5.1.3.4	vTaskSuspendAll	10
5.1.3.5	xTaskResumeAll	11
5.2	ARPLib stack	11
5.2.1	Detailed Description	11
5.2.2	ARP library	11
5.2.3	Function Documentation	11
5.2.3.1	ARPResolveMAC	11
5.3	FTPLib stack	12
5.3.1	Detailed Description	12
5.3.2	library	12
5.3.3	Function Documentation	12
5.3.3.1	FTPClientOpen	12
5.3.3.2	FTPClose	12
5.3.3.3	FTPIsConn	13
5.3.3.4	FTPRead	13
5.3.3.5	FTPRxLen	13
5.3.3.6	FTPWrite	14
5.4	ADC	14
5.4.1	Detailed Description	14
5.4.2	Function Documentation	14
5.4.2.1	ADCInit	14

	5.4.2.2	ADCVal	15
5.5		GPIOs	15
	5.5.1	Detailed Description	15
	5.5.2	Function Documentation	15
		5.5.2.1 IOButtonState	15
		5.5.2.2 IOGet	16
		5.5.2.3 IOInit	16
		5.5.2.4 IOPut	16
5.6		UART	17
	5.6.1	Detailed Description	17
	5.6.2	Function Documentation	17
		5.6.2.1 UARTBufferSize	17
		5.6.2.2 UARTFlush	17
		5.6.2.3 UARTInit	18
		5.6.2.4 UARTOff	18
		5.6.2.5 UARTOn	18
		5.6.2.6 UARTRead	18
		5.6.2.7 UARTWrite	19
		5.6.2.8 UARTWriteCh	19
5.7		PWM	19
	5.7.1	Detailed Description	20
	5.7.2	Function Documentation	20
		5.7.2.1 PWMDuty	20
		5.7.2.2 PWMInit	20
		5.7.2.3 PWMOff	20
		5.7.2.4 PWMon	21
5.8		I2C	21
	5.8.1	Detailed Description	21
	5.8.2	Function Documentation	21
		5.8.2.1 I2CInit	21
		5.8.2.2 I2CRead	22
		5.8.2.3 I2CRestart	22
		5.8.2.4 I2CStart	22
		5.8.2.5 I2CStop	22
		5.8.2.6 I2CWrite	23
5.9		SMTPlib stack	23
	5.9.1	Detailed Description	23
	5.9.2	Function Documentation	23
		5.9.2.1 SMTPBusy	23
		5.9.2.2 SMTPSend	24
		5.9.2.3 SMTPSetMsg	24
		5.9.2.4 SMTPSetServer	24
		5.9.2.5 SMTPStart	25
		5.9.2.6 SMTPStop	25
5.10		TCPLib stack	25
	5.10.1	Detailed Description	26
	5.10.2	library	26
	5.10.3	Function Documentation	26
		5.10.3.1 TCPClientClose	26
		5.10.3.2 TCPClientOpen	26

5.10.3.3	TCPisConn	27
5.10.3.4	TCPRead	27
5.10.3.5	TCPRxLen	27
5.10.3.6	TCPServerClose	28
5.10.3.7	TCPServerDetach	28
5.10.3.8	TCPServerOpen	28
5.10.3.9	TCPWrite	28
5.11	UDPLib stack	29
5.11.1	Detailed Description	29
5.11.2	UDP library	29
5.11.3	Function Documentation	29
5.11.3.1	UDPBroadcastOpen	29
5.11.3.2	UDPClientClose	30
5.11.3.3	UDPClientOpen	30
5.11.3.4	UDPRead	30
5.11.3.5	UDPRxLen	30
5.11.3.6	UDPServerClose	31
5.11.3.7	UDPServerOpen	31
5.11.3.8	UDPWrite	31
5.12	WiFi	31
5.12.1	Detailed Description	32
5.12.2	Function Documentation	32
5.12.2.1	WFConnect	32
5.12.2.2	WFCustomDelete	32
5.12.2.3	WFCustomExist	33
5.12.2.4	WFCustomLoad	33
5.12.2.5	WFCustomSave	33
5.12.2.6	WFDisconnect	34
5.12.2.7	WFHibernate	34
5.12.2.8	WFPsPollEnable	34
5.12.2.9	WFScan	34
5.12.2.10	WFScanList	35
5.12.2.11	WFSetParam	35
5.12.2.12	WFSetSecurity	36
5.12.2.13	WFStopConnecting	37
5.13	System	37
5.14	Net	37
5.15	Hardware	38
6	Data Structure Documentation	39
6.1	tWFNetwork Struct Reference	39
7	File Documentation	41
7.1	C:/openpicus/X Release 2.0/Flyport libs/ARPlib.c File Reference	41
7.1.1	Detailed Description	41
7.2	C:/openpicus/X Release 2.0/Flyport libs/FTPlib.c File Reference	41
7.2.1	Detailed Description	41
7.3	C:/openpicus/X Release 2.0/Flyport libs/HWlib.c File Reference	42
7.3.1	Detailed Description	42
7.4	C:/openpicus/X Release 2.0/Flyport libs/SMTPlib.c File Reference	42

7.4.1	Detailed Description	43
7.5	C:/openpicus/X Release 2.0/Flyport libs/TCPIib.c File Reference	43
7.5.1	Detailed Description	43
7.6	C:/openpicus/X Release 2.0/Flyport libs/UDPIib.c File Reference	43
7.6.1	Detailed Description	44
7.7	C:/openpicus/X Release 2.0/Flyport libs/WFlib.c File Reference	44
7.7.1	Detailed Description	44

Chapter 1

OpenPicus

OpenPicus is open source Hardware and Software wireless project to enable Smart Sensors and Internet of Things.

- **Hardware platform:** it's modular, the Modules are PICUSes while the Carrier Boards are their NESTs
- **Wireless:** Wi-Fi or Bluetooth. You have full control of the Stack and power down modes.
- **Software Framework:** your Apps can control the functions of the Protocol Stack, but you don't need to be an expert of it.
- **Development tool:** free IDE is ready to let you start development immediately.
- **Serial Bootloader:** Brutus loaded on modules, you don't need a programmer

Flyport is the first OpenPicus device, embedding the TCP/IP and WiFi stack for wireless communication with 802.11 devices. The Flyport also includes a real time operating system (FreeRTOS - www.freertos.org) to manage easily the TCP/IP stack and the user application in two different tasks.



Authors

Gabriele Allegria

Claudio Carnevali

Special thanks to Andrea Seraghiti for the support in development.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

System	37
Delays	9
Net	37
ARPLib stack	11
FTPLib stack	12
SMTPlib stack	23
TCPLib stack	25
UDPLib stack	29
WiFi	31
Hardware	38
ADC	14
GPIOs	15
UART	17
PWM	19
I2C	21

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

tWFNetwork	39
--------------------------------------	----

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

C:/openpicus/X Release 2.0/Flyport libs/ARPlib.c (ARP wrapper for FreeRTOS)	41
C:/openpicus/X Release 2.0/Flyport libs/FTPlib.c (FTP wrapper for FreeRTOS)	41
C:/openpicus/X Release 2.0/Flyport libs/HWlib.c (Hardware library to manage the analog and digital IOs and UART)	42
C:/openpicus/X Release 2.0/Flyport libs/SMTPlib.c (SMTP wrapper for FreeRTOS)	42
C:/openpicus/X Release 2.0/Flyport libs/TCPlib.c (TCP wrapper for FreeRTOS)	43
C:/openpicus/X Release 2.0/Flyport libs/UDPlib.c (UDP wrapper for FreeRTOS)	43
C:/openpicus/X Release 2.0/Flyport libs/WFlib.c (Function to manage to Wifi module and to change at runtime the network settings)	44
C:/openpicus/X Release 2.0/Flyport libs/Include/ARPlib.h	??
C:/openpicus/X Release 2.0/Flyport libs/Include/FTPlib.h	??
C:/openpicus/X Release 2.0/Flyport libs/Include/HWlib.h	??
C:/openpicus/X Release 2.0/Flyport libs/Include/HWmap.h	??
C:/openpicus/X Release 2.0/Flyport libs/Include/SMTPlib.h	??
C:/openpicus/X Release 2.0/Flyport libs/Include/TCPlib.h	??
C:/openpicus/X Release 2.0/Flyport libs/Include/UDPlib.h	??
C:/openpicus/X Release 2.0/Flyport libs/Include/WFlib.h	??
C:/openpicus/X Release 2.0/Flyport/HTTPPrint.h	??
C:/openpicus/X Release 2.0/Flyport/taskFlyport.h	??
C:/openpicus/X Release 2.0/Flyport/taskTCPIP.h	??
C:/openpicus/X Release 2.0/Flyport/TCPIPConfig.h	??
C:/openpicus/X Release 2.0/Flyport/WF_Config.h	??
C:/openpicus/X Release 2.0/Flyport/WF_Events.h	??

Chapter 5

Module Documentation

5.1 Delays

Functions

- void [vTaskDelay](#) (int delay)
- void [DelayMs](#) (WORD ms)
- void [DelayUs](#) (WORD ms)
- void [vTaskSuspendAll](#) ()
- void [xTaskResumeAll](#) ()

5.1.1 Detailed Description

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

5.1.2 Delays

5.1.3 Function Documentation

5.1.3.1 void [DelayMs](#) (WORD ms)

[DelayMs](#) - Introduces a delay inside the execution of the firmware.

Parameters

<i>delay</i>	- 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.

5.1.3.2 void DelayUs (WORD ms)

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

Parameters

<i>delay</i>	- 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.

5.1.3.3 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

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

Returns

None

5.1.3.4 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

<i>None</i>	
-------------	--

Returns

None

Warning

when inside the critical 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.

5.1.3.5 void xTaskResumeAll ()

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

Parameters

<i>None</i>

Returns

None

5.2 ARPIib stack**Functions**

- BYTE [ARPResolveMAC](#) (char ipaddr[])

5.2.1 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.

5.2.2 ARP library**5.2.3 Function Documentation****5.2.3.1 BYTE ARPResolveMAC (char ipaddr[])**

ARPResolveMAC - Force an arp request for specific IP address

Parameters

<i>ipaddr</i>	IP address
---------------	------------

5.3 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 [FTPRLen](#) (TCP_SOCKET ftpsocklen)

5.3.1 Detailed Description

FTP provides Internet communication abilities.

5.3.2 library

5.3.3 Function Documentation

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

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

Parameters

<i>ftpaddr</i>	- IP address of the remote server. Example: "192.168.1.100" (the char array must be NULL terminated).
<i>ftpport</i>	- 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).

5.3.3.2 void FTPClose (TCP_SOCKET *Sockclose*)

FTPClose - Closes the client socket specified by the handle.

Parameters

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

Returns

None.

5.3.3.3 BOOL FTPisConn (TCP_SOCKET *sockcon*)

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

Parameters

<i>sockcon</i>	- 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.

5.3.3.4 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

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

5.3.3.5 WORD FTPRXLen (TCP_SOCKET *ftpsocklen*)

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

Parameters

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

Returns

The number of bytes available to be read.

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

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

Parameters

<i>ftpsockwr</i>	- The socket to which data is to be written (it's the handle returned by the command TCPClientOpen or TCPServerOpen).
<i>ftpstrtowr</i>	- Pointer to the array of characters to be written.
<i>ftpwlen</i>	- 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 connected.

5.4 ADC**Functions**

- void [ADCInit](#) ()
- int [ADCVal](#) (int ch)

5.4.1 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.

5.4.2 Function Documentation**5.4.2.1 void ADCInit ()**

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

Parameters

<i>None</i>

Returns

None

5.4.2.2 int ADCVal (int *ch*)

ADCVal - Reads the value of the analog channel specified.

Parameters

<i>ch</i>	- 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.

5.5 GPIOs

Functions

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

5.5.1 Detailed Description

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

5.5.2 Function Documentation

5.5.2.1 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

<i>io</i>	- the pin to read.
-----------	--------------------

Returns

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

5.5.2.2 int IOGet (int *io*)

IOGet - Reads the value of the specified pin.

Parameters

<i>io</i>	- the pin to read.
-----------	--------------------

Returns

the value read by the function.

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

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

Parameters

<i>io</i>	- specifies the pin.
<i>putval</i>	- specifies how the pin must be initialized. The valid parameters are the following: <ul style="list-style-type: none"> • 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.

Returns

None

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

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

Parameters

<i>io</i>	- specifies the ouput pin.
<i>putval</i>	- the value to assign to the pin: <ul style="list-style-type: none"> • on (or ON, or 1) high level (about 3.3V). • off (or OFF, or 0) low level (0V).

Returns

None

5.6 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)

5.6.1 Detailed Description

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

5.6.2 Function Documentation

5.6.2.1 int [UARTBufferSize](#) (int *port*)

[UARTBufferSize](#) - Returns the RX buffer size of the specified UART port.

Parameters

<i>port</i>	- the UART port to read.
-------------	--------------------------

Returns

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

5.6.2.2 void [UARTFlush](#) (int *port*)

[UARTFlush](#) - Flushes the buffer of the specified UART port.

Parameters

<i>port</i>	- the UART port to flush.
-------------	---------------------------

Returns

None

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

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

Parameters

<i>port</i>	- 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.
<i>baud</i>	- the desired baudrate.

Returns

None

5.6.2.4 void UARTOff (int *port*)

UARTOff - Turns off the specified UART port.

Parameters

<i>port</i>	- the UART port to turn off.
-------------	------------------------------

Returns

None

5.6.2.5 void UARTOn (int *port*)

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

Parameters

<i>port</i>	- the UART port to turn on.
-------------	-----------------------------

Returns

None

5.6.2.6 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

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

Returns

the report for the operation:

- $N > 0$: N characters correctly read.
- $N < 0$: N characters read, but buffer overflow detected.

Warning

The output of this function is changed between IDE 1.0 and IDE 2.0

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

UARTWrite - writes the specified string on the UART port.

Parameters

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

Returns

None

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

UARTWriteCh - writes a single character on the UART port.

Parameters

<i>port</i>	- the UART port to write to.
<i>chr</i>	- the char to write.

Returns

None

5.7 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)

5.7.1 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's possible to change the duty cycle of the PWM.

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

5.7.2 Function Documentation

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

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

Parameters

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

Returns

None

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

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

Parameters

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

Returns

None

5.7.2.3 void PWMOff (BYTE *pwm*)

PWMOff - turns off the specified PWM.

Parameters

<i>pwm</i>	- PWM number previously defined in PWMInit.
------------	---

Returns

None

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

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

Parameters

<i>io</i>	- pin to assign the PWM.
<i>pwm</i>	- PWM number previously defined in PWMInit.

Returns

None

5.8 I2C

Functions

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

5.8.1 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.

5.8.2 Function Documentation

5.8.2.1 void I2CInit (BYTE *I2CSpeed*)

I2CInit - Initializes the I2C module.

Parameters

<i>I2CSpeed</i>	- can use HIGH_SPEED for 404K or use LOW_SPEED for 100K.
-----------------	--

Returns

None

Warning

This function changed between IDE 1.0 and IDE 2.0. IDE 1.0 doesn't accept I2CSpeed parameter

5.8.2.2 BYTE I2CRead (BYTE *ack*)

I2CRead - Reads one byte from the data bus .

Parameters

<i>None</i>

Returns

None

5.8.2.3 void I2CRestart ()

I2CRestart - Sends a repeated start sequence on the bus .

Parameters

<i>None</i>

Returns

None

5.8.2.4 void I2CStart ()

I2CStart - Sends a start sequence on the bus.

Parameters

<i>None</i>

Returns

None

5.8.2.5 void I2CStop ()

I2CStop - Stops the transmissions on the bus.

Parameters

<i>None</i>

Returns

None

5.8.2.6 void I2CWrite (BYTE data)

I2CWrite - writes one byte on the bus.

Parameters

None

Returns

None

5.9 SMTPlib stack

Functions

- BOOL [SMTPStart](#) ()
- void [SMTPSetServer](#) (int servparam, char *paramfield)
- void [SMTPSetMsg](#) (int msgparam, char *mparamfield)
- BOOL [SMTPSend](#) ()
- BOOL [SMTPBusy](#) ()
- BOOL [SMTPStop](#) ()

5.9.1 Detailed Description

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

5.9.2 Function Documentation

5.9.2.1 BOOL SMTPBusy ()

SMTPBusy - Verifies if the SMTP client is busy performing 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.

5.9.2.2 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

<i>None</i>

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.

5.9.2.3 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

<i>msgparam</i>	<p>the parameter to set. Available list is the following:</p> <ul style="list-style-type: none"> • MSG_TO: the destination address. • MSG_CC: the cc address.</L1> • MSG_BCC: the bcc address.</L1> • MSG_FROM: the sender of the email message.</L1> • MSG_SUBJECT: the object of the email message.</L1> • MSG_BODY: the body of the email message.</L1>
<i>mparamfield</i>	<p>a string containing the associated value to the parameter. For example: if we want to send a message to <code>example@openpicus.com</code>. <code>SMTPSetMsg(MSG_TO , "example@openpicus.com")</code></p>

Returns

None.

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

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

Parameters

<i>servparam</i>	the parameter to set. Available list is the following: <ul style="list-style-type: none"> • SERVER_NAME: the name of the SMTP server. • SERVER_USER: the username for access to the server.</L1> • SERVER_PASS: the password associated to the username.</L1> • SERVER_PORT: the port of the SMTP server.</L1>
<i>paramfield</i>	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.

5.9.2.5 BOOL SMTPStart ()

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

Parameters

<i>None</i>

Returns

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

5.9.2.6 BOOL SMTPStop ()

SMTPStop - Closes the SMTP client. To use it again must restart it.

Parameters

<i>None</i>

Returns

TRUE: the client correctly closed.
FALSE: an error occurred closing the client.

5.10 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)
- WORD [TCPWrite](#) (TCP_SOCKET socktowrite, char *writtech, int wlen)
- void [TCPClientClose](#) (TCP_SOCKET Sockclose)
- void [TCPServerClose](#) (TCP_SOCKET Sockclose)
- BOOL [TCPisConn](#) (TCP_SOCKET sockconn)
- WORD [TCPRxLen](#) (TCP_SOCKET socklen)

5.10.1 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.

5.10.2 library

5.10.3 Function Documentation

5.10.3.1 void TCPClientClose (TCP_SOCKET *Sockclose*)

TCPClientClose - Closes the client socket specified by the handle.

Parameters

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

Returns

None.

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

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

Parameters

<i>tcpaddr</i>	- IP address of the remote server. Example: "192.168.1.100" (the char array must be NULL terminated).
<i>tcpport</i>	- 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).

5.10.3.3 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

<i>sockconn</i>	- 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.

5.10.3.4 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

Parameters

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

5.10.3.5 WORD TCPRxLen (TCP_SOCKET *socklen*)

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

Parameters

<i>socklen</i>	- 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.

5.10.3.6 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

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

Returns

None.

5.10.3.7 void TCPServerDetach (TCP_SOCKET *sockdet*)

TCPServerDetach - Detaches the remote client from the server

Parameters

<i>sockdet</i>	- Socket of the server (the handle returned by the command TCPServerOpen).
----------------	--

Returns

None

5.10.3.8 TCP_SOCKET TCPServerOpen (char *tcpport*[])

TCPServerOpen - Creates a TCP server on specified port

Parameters

<i>tcpport</i>	- 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).

5.10.3.9 WORD TCPWrite (TCP_SOCKET *socktowrite*, char * *writetech*, int *wlen*)

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

Parameters

<i>socktowrite</i>	- The socket to which data is to be written (it's the handle returned by the command TCPClientOpen or TCPServerOpen).
<i>writtech</i>	- Pointer to the array of characters to be written.
<i>wlen</i>	- 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.

5.11 UDPLib stack**Functions**

- 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)
- int [UDPRead](#) (BYTE sock, char str2rd[], int lstr)
- WORD [UDPWrite](#) (BYTE sockwr, BYTE *str2wr, int lstr)

5.11.1 Detailed Description

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

5.11.2 UDP library**5.11.3 Function Documentation****5.11.3.1 BYTE UDPBroadcastOpen (char *udpport*[])**

UDPBroadcastOpen - Create a UDP broadcast on specified port

Parameters

<i>udpport</i>	Remote Port for UDP
----------------	---------------------

Returns

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

5.11.3.2 BYTE UDPClose (BYTE sock)

UDPClose - Closes UDP Client socket

Parameters

<i>sock</i>	UDP Socket number
-------------	-------------------

5.11.3.3 BYTE UDPOpen (char * udpaddr, char udpport[])

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

Parameters

<i>udpaddr</i>	IP address of server
<i>udpport</i>	Remote Port of UDP server

Returns

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

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

UDPRead - Reads a byte from the RX buffer

Parameters

<i>sock</i>	UDP socket number
<i>str2rd</i>	Buffer for data
<i>lstr</i>	length of string

Returns

The number of read characters from the specified UDP socket.

5.11.3.5 WORD UDPRxLen (BYTE sock)

UDPRxLen - Reads the length of the RX buffer

Parameters

<i>sock</i>	UDP socket number
-------------	-------------------

Returns

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

5.11.3.6 BYTE UDPServerClose (BYTE sock)

UDPServerClose - Closes UDP Server socket

Parameters

<i>sock</i>	UDP Socket number
-------------	-------------------

5.11.3.7 BYTE UDPServerOpen (char udpport[])

UDPServerOpen - Create a UDP server on specified port

Parameters

<i>udpport</i>	Local Port for UDP server
----------------	---------------------------

Returns

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

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

UDPWrite - Writes on the UDP socket

Parameters

<i>sockwr</i>	UDP socket number
<i>str2wr</i>	String to write
<i>lstr</i>	String length

Returns

The number of write characters to the specified UDP socket.

5.12 WiFi**Functions**

- void **WFGeneric** (int function)
- void **WFConnect** (int pconn)
- void **WFDisconnect** ()
- void **WFScan** ()
- void **WFCustomSave** ()
- void **WFCustomDelete** ()
- BOOL **WFCustomExist** ()
- void **WFCustomLoad** ()

- [tWFNetwork WFScanList](#) (int ntscn)
- void [WFSetParam](#) (int paramtoset, char *paramstring)
- void [WFStopConnecting](#) ()
- void [WFSetSecurity](#) (BYTE mode, char *keypass, BYTE keylen, BYTE keyind)
- void [WFHibernate](#) ()
- void [WFPsPollEnable](#) (BOOL ps_active)

5.12.1 Detailed Description

Introduction

The **WiFi library** contains all the functions to manage the WiFi stack. It's possible to connect or disconnect from a network, change the settings, and save them inside the flash 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 WiFi network. The Flyport module has two possible profile to use:

- **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

5.12.2 Function Documentation

5.12.2.1 void WFCConnect (int *pconn*)

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

Parameters

<i>pconn</i>	- Specifies the profile used to connect to the network. The following profile are available: WF_DEFAULT : uses the settings chosen 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

5.12.2.2 void WFCustomDelete ()

WFCustomDelete - Deletes the custom settings for the network profile WF_CUSTOM.

Parameters

<i>None</i>

Returns

None

5.12.2.3 BOOL WFCustomExist ()

WFCustomExist - Verifies if in memory is present some data for the WF_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.

Parameters

<i>None</i>

Returns

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

5.12.2.4 void WFCustomLoad ()

WFCustomLoad - Loads from the flash memory the previously set parameters for the WF_CUSTOM profile.

Parameters

<i>None</i>

Returns

None

5.12.2.5 void WFCustomSave ()

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

Parameters

<i>None</i>

Returns

None

5.12.2.6 void WFDDisconnect ()

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

Parameters

<i>None</i>

Returns

None

5.12.2.7 void WFHibernate ()

WFHibernate - Enables Hibernate mode on the MRF24WB0M, which effectively turns off the device for maximum power savings. At the moment to exit from hibernate state a Reset() is needed.

Parameters

<i>None</i>

Returns

None

5.12.2.8 void WFPsPollEnable (BOOL *ps_active*)

WFPsPollEnable - Enables or Disable Power-Save Pool at runtime

Parameters

<i>ps_active</i>	TRUE to enable Poll mode for longer battery life or FALSE to disable PS Poll mode, MRF24WB0M will stay active and not go sleep.
------------------	---

Returns

None

5.12.2.9 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

<i>None</i>

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.

5.12.2.10 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

<i>ntscn</i>	- 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`.

5.12.2.11 void WFSetParam (int *paramtoiset*, char * *paramstring*)

`WFSetParam` - With this command is possible to change any network parameter of the `WF_CUSTOM` profile.

Parameters

<i>paramtoiset</i>	- the parameter to change.
<i>paramstring</i>	- value of the parameter.

Returns

None.

Syntax of the command to set the parameters:

- **IP address of the device:** WFSiParam(MY_IP_ADDR , string with the IP address, for example "192.168.1.100")
- **Primary DNS server:** WFSiParam(PRIMARY_DNS , string with the IP address, for example "192.168.1.1")
- **Secondary DNS server:** WFSiParam(SECONDARY_DNS , string with the IP address, for example "192.168.1.1")
- **Default gateway:** WFSiParam(MY_GATEWAY , string with the IP address, for example "192.168.1.1")
- **Subnet mask:** WFSiParam(SUBNET_MASK , string with the subnet mask, for example "255.255.255.0")
- **Netbios name:** WFSiParam(NETBIOS_NAME , string with the Netbios name, for example "Flyport")
- **SSID:** WFSiParam(SSID_NAME , string with the SSID name, for example "FlyportNet")
- **DHCP client enabled or not:** WFSiParam(DHCP_ENABLE , ENABLED or DISABLED)
- **Network type (adhoc or infrastructure):** WFSiParam(NETWORK_TYPE , ADHOC or INFRASTRUCTURE)

5.12.2.12 void WFSiParamSecurity (BYTE mode, char * keypass, BYTE keylen, BYTE keyind)

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

Parameters

<i>mode</i>	<p>- the security mode. Valid security mode are the following:</p> <ul style="list-style-type: none"> • 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.
-------------	--

<i>keypass</i>	- the key or passphrase for the network. A key must be specified also for open connections (you can put a blank string, like "").
<i>keylen</i>	- length of the key/passphrase. Must be specified also for open connections (can be 0).
<i>keyind</i>	- 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.

5.12.2.13 void WFStopConnecting ()

WFStopConnecting - When the command WFCConnect is launched, 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 issue the command WFStopConnecting.

Parameters

<i>None</i>

Returns

None

5.13 System**Modules**

- [Delays](#)

5.14 Net**Modules**

- [ARPlib stack](#)
- [FTPlib stack](#)
- [SMTPlib stack](#)
- [TCPlib stack](#)
- [UDPlib stack](#)
- [WiFi](#)

5.15 Hardware

Modules

- [ADC](#)
- [GPIOs](#)
- [UART](#)
- [PWM](#)
- [I2C](#)

Functions

- void `HWInit` (int conf)

Chapter 6

Data Structure Documentation

6.1 tWFNetwork Struct Reference

Data Fields

- BYTE **ssid** [WF_BSSID_LENGTH]
- char **ssid** [WF_MAX_SSID_LENGTH+1]
- UINT8 **channel**
- UINT8 **signal**
- BYTE **security**
- BYTE **type**
- UINT8 **beacon**
- UINT8 **preamble**

The documentation for this struct was generated from the following file:

- C:/openpicus/X Release 2.0/Flyport libs/Include/WFlib.h

Chapter 7

File Documentation

7.1 C:/openpicus/X Release 2.0/Flyport libs/ARPLib.c File Reference

ARP wrapper for FreeRTOS.

Functions

- BYTE [ARPResolveMAC](#) (char ipaddr[])

7.1.1 Detailed Description

ARP wrapper for FreeRTOS.

7.2 C:/openpicus/X Release 2.0/Flyport libs/FTPLib.c File Reference

FTP wrapper for FreeRTOS.

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)

7.2.1 Detailed Description

FTP wrapper for FreeRTOS.

7.3 C:/openpicus/X Release 2.0/Flyport libs/HWlib.c File Reference

Hardware library to manage the analog and digital IOs and UART.

Functions

- void [ADCInit](#) ()
- int [ADCVal](#) (int ch)
- void [IOPut](#) (int io, int putval)
- void [IOInit](#) (int io, int putval)
- int [IOGet](#) (int io)
- int [IOButtonState](#) (int io)
- 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)
- void [PWMinit](#) (BYTE pwm, float freq, float dutyc)
- void [PWMon](#) (BYTE io, BYTE pwm)
- void [PWMDuty](#) (float duty, BYTE pwm)
- void [PWMOff](#) (BYTE pwm)
- void [I2CInit](#) (BYTE I2CSpeed)
- void [I2CStart](#) ()
- void [I2CRestart](#) ()
- void [I2CStop](#) ()
- void [I2CWrite](#) (BYTE data)
- BYTE [I2CRead](#) (BYTE ack)
- void [HWInit](#) (int conf)

7.3.1 Detailed Description

Hardware library to manage the analog and digital IOs and UART.

7.4 C:/openpicus/X Release 2.0/Flyport libs/SMTPlib.c File Reference

SMTP wrapper for FreeRTOS.

Functions

- BOOL [SMTPStart](#) ()
- void [SMTPSetServer](#) (int servparam, char *paramfield)
- void [SMTPSetMsg](#) (int msgparam, char *mparamfield)
- BOOL [SMTPSend](#) ()
- BOOL [SMTPBusy](#) ()
- BOOL [SMTPStop](#) ()

7.4.1 Detailed Description

SMTP wrapper for FreeRTOS.

7.5 C:/openpicus/X Release 2.0/Flyport libs/TCPLib.c File Reference

TCP wrapper for FreeRTOS.

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)
- WORD [TCPWrite](#) (TCP_SOCKET socktowrite, char *writech, int wlen)
- void [TCPClientClose](#) (TCP_SOCKET Sockclose)
- void [TCPServerClose](#) (TCP_SOCKET Sockclose)
- BOOL [TCPisConn](#) (TCP_SOCKET sockconn)
- WORD [TCPRxLen](#) (TCP_SOCKET socklen)

7.5.1 Detailed Description

TCP wrapper for FreeRTOS.

7.6 C:/openpicus/X Release 2.0/Flyport libs/UDPLib.c File Reference

UDP wrapper for FreeRTOS.

Functions

- 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)
- int [UDPRead](#) (BYTE sock, char str2rd[], int lstr)
- WORD [UDPWrite](#) (BYTE sockwr, BYTE *str2wr, int lstr)

7.6.1 Detailed Description

UDP wrapper for FreeRTOS.

7.7 C:/openpicus/X Release 2.0/Flyport libs/WFlib.c File Reference

Function to manage to Wifi module and to change at runtime the network settings.

Functions

- void [WFGeneric](#) (int function)
- void [WFConnect](#) (int pconn)
- void [WFDisconnect](#) ()
- void [WFScan](#) ()
- void [WFCustomSave](#) ()
- void [WFCustomDelete](#) ()
- BOOL [WFCustomExist](#) ()
- void [WFCustomLoad](#) ()
- [tWFNetwork WFScanList](#) (int ntscn)
- void [WFSetParam](#) (int paramtoiset, char *paramstring)
- void [WFStopConnecting](#) ()
- void [WFSetSecurity](#) (BYTE mode, char *keypass, BYTE keylen, BYTE keyind)
- void [WFHibernate](#) ()
- void [WFPsPollEnable](#) (BOOL ps_active)

7.7.1 Detailed Description

Function to manage to Wifi module and to change at runtime the network settings.

Index

ADC, [14](#)
 ADCInit, [14](#)
 ADCVal, [14](#)
ADCInit
 ADC, [14](#)
ADCVal
 ADC, [14](#)
ARPlib
 ARPResolveMAC, [11](#)
ARPlib stack, [11](#)
ARPResolveMAC
 ARPlib, [11](#)

C:/openpicus/X Release 2.0/Flyport libs/ARPlib.c, FTPlib, [13](#)
 [41](#)
C:/openpicus/X Release 2.0/Flyport libs/FTP-
 Plib.c, [41](#)
C:/openpicus/X Release 2.0/Flyport libs/H-
 Wlib.c, [42](#)
C:/openpicus/X Release 2.0/Flyport libs/SMTGPIOs, [15](#)
 Plib.c, [42](#)
C:/openpicus/X Release 2.0/Flyport libs/TC-
 Plib.c, [43](#)
C:/openpicus/X Release 2.0/Flyport libs/UD-
 Plib.c, [43](#)
C:/openpicus/X Release 2.0/Flyport libs/WFIlibHardware, [38](#)
 [44](#)

DelayMs
 Delays, [9](#)
Delays, [9](#)
 DelayMs, [9](#)
 DelayUs, [10](#)
 vTaskDelay, [10](#)
 vTaskSuspendAll, [10](#)
 xTaskResumeAll, [11](#)
DelayUs
 Delays, [10](#)

FTPClientOpen
 FTPlib, [12](#)

FTPClose
 FTPlib, [12](#)
FTPisConn
 FTPlib, [13](#)
FTPlib
 FTPClientOpen, [12](#)
 FTPClose, [12](#)
 FTPisConn, [13](#)
 FTPRead, [13](#)
 FTPRxLen, [13](#)
 FTPWrite, [14](#)
FTPlib stack, [12](#)
FTPRead
 FTPRxLen
 FTPlib, [13](#)
 FTPWrite
 FTPlib, [14](#)

IOButtonState, [15](#)
IOGet, [15](#)
IOInit, [16](#)
IOPut, [16](#)

I2C, [21](#)
 I2CInit, [21](#)
 I2CRead, [21](#)
 I2CRestart, [22](#)
 I2CStart, [22](#)
 I2CStop, [22](#)
 I2CWrite, [22](#)
I2CInit
 I2C, [21](#)
I2CRead
 I2C, [21](#)
I2CRestart
 I2C, [22](#)
I2CStart

- I2C, [22](#)
- I2CStop
 - I2C, [22](#)
- I2CWrite
 - I2C, [22](#)
- IOButtonState
 - GPIOs, [15](#)
- IOGet
 - GPIOs, [15](#)
- IOInit
 - GPIOs, [16](#)
- IOPut
 - GPIOs, [16](#)
- Net, [37](#)
- PWM, [19](#)
 - PWMDuty, [20](#)
 - PWMInit, [20](#)
 - PWMOff, [20](#)
 - PWMon, [20](#)
- PWMDuty
 - PWM, [20](#)
- PWMInit
 - PWM, [20](#)
- PWMOff
 - PWM, [20](#)
- PWMon
 - PWM, [20](#)
- SMTPBusy
 - SMTPLib, [23](#)
- SMTPLib
 - SMTPBusy, [23](#)
 - SMTPSend, [23](#)
 - SMTPSetMsg, [24](#)
 - SMTPSetServer, [24](#)
 - SMTPStart, [25](#)
 - SMTPStop, [25](#)
- SMTPLib stack, [23](#)
- SMTPSend
 - SMTPLib, [23](#)
- SMTPSetMsg
 - SMTPLib, [24](#)
- SMTPSetServer
 - SMTPLib, [24](#)
- SMTPStart
 - SMTPLib, [25](#)
- SMTPStop
 - SMTPLib, [25](#)
- System, [37](#)
- TCPClientClose
 - TCPLib, [26](#)
- TCPClientOpen
 - TCPLib, [26](#)
- TCPisConn
 - TCPLib, [27](#)
- TCPLib
 - TCPClientClose, [26](#)
 - TCPClientOpen, [26](#)
 - TCPisConn, [27](#)
 - TCPRead, [27](#)
 - TCPRxLen, [27](#)
 - TCPServerClose, [27](#)
 - TCPServerDetach, [28](#)
 - TCPServerOpen, [28](#)
 - TCPWrite, [28](#)
- TCPLib stack, [25](#)
- TCPRead
 - TCPLib, [27](#)
- TCPRxLen
 - TCPLib, [27](#)
- TCPServerClose
 - TCPLib, [27](#)
- TCPServerDetach
 - TCPLib, [28](#)
- TCPServerOpen
 - TCPLib, [28](#)
- TCPWrite
 - TCPLib, [28](#)
- tWFNetwork, [39](#)
- UART, [17](#)
 - UARTBufferSize, [17](#)
 - UARTFlush, [17](#)
 - UARTInit, [17](#)
 - UARTOff, [18](#)
 - UARTOn, [18](#)
 - UARTRead, [18](#)
 - UARTWrite, [19](#)
 - UARTWriteCh, [19](#)
- UARTBufferSize
 - UART, [17](#)
- UARTFlush
 - UART, [17](#)
- UARTInit
 - UART, [17](#)
- UARTOff
 - UART, [18](#)

- UARTOn
 - UART, [18](#)
- UARTRead
 - UART, [18](#)
- UARTWrite
 - UART, [19](#)
- UARTWriteCh
 - UART, [19](#)
- UDPBroadcastOpen
 - UDPLib, [29](#)
- UDPClientClose
 - UDPLib, [29](#)
- UDPClientOpen
 - UDPLib, [30](#)
- UDPLib
 - UDPBroadcastOpen, [29](#)
 - UDPClientClose, [29](#)
 - UDPClientOpen, [30](#)
 - UDPRead, [30](#)
 - UDPRxLen, [30](#)
 - UDPServerClose, [30](#)
 - UDPServerOpen, [31](#)
 - UDPWrite, [31](#)
- UDPLib stack, [29](#)
- UDPRead
 - UDPLib, [30](#)
- UDPRxLen
 - UDPLib, [30](#)
- UDPServerClose
 - UDPLib, [30](#)
- UDPServerOpen
 - UDPLib, [31](#)
- UDPWrite
 - UDPLib, [31](#)
- vTaskDelay
 - Delays, [10](#)
- vTaskSuspendAll
 - Delays, [10](#)
- WFCConnect
 - WiFi, [32](#)
- WFCustomDelete
 - WiFi, [32](#)
- WFCustomExist
 - WiFi, [33](#)
- WFCustomLoad
 - WiFi, [33](#)
- WFCustomSave
 - WiFi, [33](#)
- WFDDisconnect
 - WiFi, [34](#)
- WFHibernate
 - WiFi, [34](#)
- WFPsPollEnable
 - WiFi, [34](#)
- WFScan
 - WiFi, [34](#)
- WFScanList
 - WiFi, [35](#)
- WFSetParam
 - WiFi, [35](#)
- WFSetSecurity
 - WiFi, [36](#)
- WFStopConnecting
 - WiFi, [37](#)
- WiFi, [31](#)
 - WFCConnect, [32](#)
 - WFCustomDelete, [32](#)
 - WFCustomExist, [33](#)
 - WFCustomLoad, [33](#)
 - WFCustomSave, [33](#)
 - WFDDisconnect, [34](#)
 - WFHibernate, [34](#)
 - WFPsPollEnable, [34](#)
 - WFScan, [34](#)
 - WFScanList, [35](#)
 - WFSetParam, [35](#)
 - WFSetSecurity, [36](#)
 - WFStopConnecting, [37](#)
- xTaskResumeAll
 - Delays, [11](#)