IoT Embedded Systems

Alexandre Silva v1.0

Generated by Doxygen 1.8.16

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

File Index

2.1 File List

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

Chapter 3

Module Documentation

3.1 MQTT

This package provides the ability to interact with remote servers through the MQTT protocol.

Macros

- #define NETWORK_MQTT_REFRESH_PERIOD_MS 66
- #define NETWORK_MQTT_TASK_STACK_SIZE configMINIMAL_STACK_SIZE ∗ 9
- #define NETWORK_MQTT_QUEUES_SIZE 10
- #define NETWORK_MQTT_ADDRESS_LENGTH 64
- #define NETWORK_MQTT_DEVICE_TOKEN_LENGTH 64
- #define MQTT_NETWORK_CONNECT_TIMEOUT 7000

Functions

- bool netMQTT_Init ()

  Initializes the netMQTT API.
- bool netMQTT_Ready ()

  Checks if the MQTT service is connected to the server and ready to be used.
- void netMQTT_Config (char ∗address, char ∗token, unsigned long port, unsigned long keepalive)

  Configures the credentials to be used in the MQTT protocol.
- void netMQTT_Start ()

  Starts the MQTT task.
- void netMQTT_Stop ()

  Stops the MQTT task.
- bool netMQTT_Publish (char ∗key, char ∗value)

  Publishes data to the previously configured MQTT server.

3.1.1 Detailed Description

This package provides the ability to interact with remote servers through the MQTT protocol.

3.1.2 Macro Definition Documentation

3.1.2.1 MQTT_NETWORK_CONNECT_TIMEOUT

```
#define MQTT_NETWORK_CONNECT_TIMEOUT 7000
```

Maximum delay to wait at boot, in milliseconds, for WiFi connection to be estabilished.

3.1.2.2 NETWORK_MQTT_ADDRESS_LENGTH

```
#define NETWORK_MQTT_ADDRESS_LENGTH 64
```

Maximum length for the MQTT address length.

3.1.2.3 NETWORK_MQTT_DEVICE_TOKEN_LENGTH

```
#define NETWORK_MQTT_DEVICE_TOKEN_LENGTH 64
```

Maximum length for the MQTT address length.

3.1.2.4 NETWORK_MQTT_QUEUES_SIZE

```
#define NETWORK_MQTT_QUEUES_SIZE 10
```

Size for the MQTT queues.

3.1.2.5 NETWORK_MQTT_REFRESH_PERIOD_MS

```
#define NETWORK_MQTT_REFRESH_PERIOD_MS 66
```

Refresh period for the MQTT task, in milliseconds.

3.1.2.6 NETWORK_MQTT_TASK_STACK_SIZE

```
#define NETWORK_MQTT_TASK_STACK_SIZE configMINIMAL_STACK_SIZE * 9
```

Stack size for the MQTT task.

3.1.3 Function Documentation

3.1.3.1 netMQTT_Config()

```
void netMQTT_Config (
            char * address,
            char * token,
            unsigned long port,
            unsigned long keepalive )
```

Configures the credentials to be used in the MQTT protocol.

Parameters

address	Address of the remote server to connect to.
token	Access token to access the remote server.
port	Port to be used to connect the remote server.
keepalive	Keepalive time, in seconds, to be used in the connection.

Returns

True if ready, false otherwise.

3.1.3.2 netMQTT_Init()

```
bool netMQTT_Init ( )
```

Initializes the netMQTT API.

Returns

True if successful, false otherwise.

Note

This function must be called prior to any other netMQTT functions.

3.1.3.3 netMQTT_Publish()

```
bool netMQTT_Publish (
            char * key,
            char * value )
```

Publishes data to the previously configured MQTT server.

Returns

True if successful, false otherwise.

3.1.3.4 netMQTT_Ready()

```
bool netMQTT_Ready ( )
```

Checks if the MQTT service is connected to the server and ready to be used.

Returns

True if ready, false otherwise.

3.1.3.5 netMQTT_Start()

```
void netMQTT_Start ( )
```

Starts the MQTT task.

Returns

None.

3.1.3.6 netMQTT_Stop()

```
void netMQTT_Stop ( )
```

Stops the MQTT task.

Returns

None.

3.2 NTP

This package provides the ability to interact with remote servers through the NTP protocol.

Macros

- #define NETWORK_NTP_QUEUES_SIZE 10
- #define NETWORK_NTP_TASK_STACK_SIZE configMINIMAL_STACK_SIZE ∗ 6

Functions

- bool netNTP_Init ()

  Initializes the netNTP API.
- time_t netNTP_GetTime (int hour_adjust)

  Retrieves the number of seconds elapsed since the year 1900 from an NTP remote server.

3.2.1 Detailed Description

This package provides the ability to interact with remote servers through the NTP protocol.

3.2.2 Macro Definition Documentation

3.2.2.1 NETWORK_NTP_QUEUES_SIZE

```
#define NETWORK_NTP_QUEUES_SIZE 10
```

Size for the NTP queues.

3.2.2.2 NETWORK_NTP_TASK_STACK_SIZE

```
#define NETWORK_NTP_TASK_STACK_SIZE configMINIMAL_STACK_SIZE * 6
```

Stack size for the NTP task.

3.2.3 Function Documentation

3.2.3.1 netNTP_GetTime()

```
time_t netNTP_GetTime (
            int hour_adjust )
```

Retrieves the number of seconds elapsed since the year 1900 from an NTP remote server.

Parameters

hour_adjust	Number of hours to add or subtract in order to get the desired time.

Returns

Number of seconds elapsed since 1900 if successful, 0 otherwise.

3.2.3.2 netNTP_Init()

```
bool netNTP_Init ( )
```

Initializes the netNTP API.

Returns

True if successful, false otherwise.

Note

This function must be called prior to any other netNTP functions.

3.3 Transport layer related functions

This package provides the ability to interact with remote servers through transport layer protocols such as TCP and UDP.

Macros

- #define NETWORK_UDP_CONNECT_TIMEOUT 10000
- #define NETWORK_TCP_CONNECT_TIMEOUT 10000
- #define NETWORK_RECEIVE_TIMEOUT 20000
- #define NETWORK_SEND_TIMEOUT 20000

Functions

- bool netUDP_Start (const char ∗host, int port)

  Estabilishes a UDP connection with a remote server.
- bool netUDP_Close ()

  Closes an ongoing UDP connection.
- bool netTCP_Start (const char ∗host, const unsigned short int port, const unsigned short int keepalive)

  Estabilishes a TCP connection with a remote server.
- bool netTCP_Close ()

  Closes an ongoing TCP connection.
- int netTRANSPORT_Send (unsigned char ∗address, unsigned int bytes)

  Sends an ammount of data to the desired address through an ongoing transport layer connection.
- int netTRANSPORT_Recv (unsigned char ∗address, unsigned int maxbytes)

  Receives an ammount of data to the desired address through an ongoing transport layer connection.

3.3.1 Detailed Description

This package provides the ability to interact with remote servers through transport layer protocols such as TCP and UDP.

3.3.2 Macro Definition Documentation

3.3.2.1 NETWORK_RECEIVE_TIMEOUT

```
#define NETWORK_RECEIVE_TIMEOUT 20000
```

Timeout to be used in the netTRANSPORT_Recv function, in milliseconds.

3.3.2.2 NETWORK_SEND_TIMEOUT

```
#define NETWORK_SEND_TIMEOUT 20000
```

Timeout to be used in the netTRANSPORT_Send function, in milliseconds.

3.3.2.3 NETWORK_TCP_CONNECT_TIMEOUT

```
#define NETWORK_TCP_CONNECT_TIMEOUT 10000
```

Timeout to be used in the netTCP_Start function, in milliseconds.

3.3.2.4 NETWORK_UDP_CONNECT_TIMEOUT

```
#define NETWORK_UDP_CONNECT_TIMEOUT 10000
```

Timeout to be used in the netUDP_Start function, in milliseconds.

3.3.3 Function Documentation

3.3.3.1 netTCP_Close()

```
bool netTCP_Close ( )
```

Closes an ongoing TCP connection.

Returns

True if successful, false otherwise.

3.3.3.2 netTCP_Start()

```
bool netTCP_Start (
             const char * host,
             const unsigned short int port,
             const unsigned short int keepalive )
```

Estabilishes a TCP connection with a remote server.

Parameters

host	Host address for the connection.
port	Host port for the connection.
keepalive	Keepalive time, in seconds, to be used in the connection.

Returns

True if successful, false otherwise.

3.3.3.3 netTRANSPORT_Recv()

```
int netTRANSPORT_Recv (
            unsigned char * address,
            unsigned int maxbytes )
```

Receives an ammount of data to the desired address through an ongoing transport layer connection.

Parameters

address	Host address to receive data from.
maxbytes	Maximum ammount of bytes to be received.

Returns

Number of bytes received if successful, -1 otherwise.

3.3.3.4 netTRANSPORT_Send()

```
int netTRANSPORT_Send (
            unsigned char * address,
            unsigned int bytes )
```

Sends an ammount of data to the desired address through an ongoing transport layer connection.

Parameters

address	Host address to send data to.
bytes	Ammount of bytes to be sent.

Returns

Number of bytes sent if successful, -1 otherwise.

3.3.3.5 netUDP_Close()

```
bool netUDP_Close ( )
```

Closes an ongoing UDP connection.

Returns

True if successful, false otherwise.

3.3.3.6 netUDP_Start()

```
bool netUDP_Start (
            const char * host,
            int port )
```

Estabilishes a UDP connection with a remote server.

Parameters

host	Host address for the connection.
port	Host port for the connection.

Returns

True if successful, false otherwise.

3.4 WiFi related functions

This package provides the ability to interact with remote access points through WiFi protocols.

Macros

- #define NETWORK_REFRESH_PERIOD_MS 66
- #define NETWORK_WIFI_QUEUES_SIZE 10
- #define NETWORK_WIFI_STACK_SIZE configMINIMAL_STACK_SIZE ∗ 9
- #define NETWORK_MAX_SSID_LENGTH 32
- #define NETWORK_MAX_PASSWORD_LENGTH 64
- #define NETWORK_MAC_ADDRESS_LENGTH 18
- #define NETWORK_MAX_BSSID_LENGTH 18
- #define NETWORK_CONNECT_TIMEOUT 20000
- #define NETWORK_WIFI_SCAN_TIMEOUT 20000
- #define NETWORK_WIFI_CHECK_PERIOD 5000

Functions

- bool netWIFI_Init ()

  Initializes the netWIFI API.
- bool netWIFI_Connect (char ∗ssid, char ∗pass, bool makedefault)

  Estabilishes a WiFi connection with an access point.
- bool netWIFI_Disconnect ()

  Disconnects from a previously connected access point.
- void netWIFI_Check (bool ∗result)

  Checks whether a usable WiFi connection is estabilished.
- void netWIFI_Scan (int max_ap, int ∗nets_scanned)

  Scans the available access points and stores the list in memory.
- Network_t ∗ netWIFI_ScannedSSID (int index)

  Retrieves a scanned available access point previously stored in memory.

3.4.1 Detailed Description

This package provides the ability to interact with remote access points through WiFi protocols.

3.4.2 Macro Definition Documentation

3.4.2.1 NETWORK_CONNECT_TIMEOUT

```
#define NETWORK_CONNECT_TIMEOUT 20000
```

Timeout to be used in the netWIFI_Connect function, in milliseconds.

3.4.2.2 NETWORK_MAC_ADDRESS_LENGTH

```
#define NETWORK_MAC_ADDRESS_LENGTH 18
```

Maximum length for a WiFi mac address.

3.4.2.3 NETWORK_MAX_BSSID_LENGTH

```
#define NETWORK_MAX_BSSID_LENGTH 18
```

Maximum length for a WiFi BSSID.

3.4.2.4 NETWORK_MAX_PASSWORD_LENGTH

```
#define NETWORK_MAX_PASSWORD_LENGTH 64
```

Maximum length for a WiFi password.

3.4.2.5 NETWORK_MAX_SSID_LENGTH

```
#define NETWORK_MAX_SSID_LENGTH 32
```

Maximum length for a WiFi SSID.

3.4.2.6 NETWORK_REFRESH_PERIOD_MS

```
#define NETWORK_REFRESH_PERIOD_MS 66
```

Refresh period for the WiFi task, in milliseconds.

3.4.2.7 NETWORK_WIFI_CHECK_PERIOD

```
#define NETWORK_WIFI_CHECK_PERIOD 5000
```

Timeout to be used in the netWIFI_Check function, in milliseconds.

3.4.2.8 NETWORK_WIFI_QUEUES_SIZE

```
#define NETWORK_WIFI_QUEUES_SIZE 10
```

Size for the WiFi queues.

3.4.2.9 NETWORK_WIFI_SCAN_TIMEOUT

```
#define NETWORK_WIFI_SCAN_TIMEOUT 20000
```

Timeout to be used in the netWIFI_Scan function, in milliseconds.

3.4.2.10 NETWORK_WIFI_STACK_SIZE

```
#define NETWORK_WIFI_STACK_SIZE configMINIMAL_STACK_SIZE * 9
```

Stack size for the WiFi task.

3.4.3 Function Documentation

3.4.3.1 netWIFI_Check()

```
void netWIFI_Check (
            bool * result )
```

Checks whether a usable WiFi connection is estabilished.

Parameters

result	Address where the result of the operation will be written to: true if successful, false otherwise.

3.4.3.2 netWIFI_Connect()

```
bool netWIFI_Connect (
            char * ssid,
            char * pass,
            bool makedefault )
```

Estabilishes a WiFi connection with an access point.

Parameters

ssid	SSID of the access point.
pass	Password of the access point.
makedefault	Connects to the indicated AP every time the WiFi module boots.

Returns

True if successful, false otherwise.

3.4.3.3 netWIFI_Disconnect()

```
bool netWIFI_Disconnect ( )
```

Disconnects from a previously connected access point.

Returns

True if successful, false otherwise.

3.4.3.4 netWIFI_Init()

```
bool netWIFI_Init ( )
```

Initializes the netWIFI API.

Returns

True if successful, false otherwise.

Note

This function must be called prior to any other netWIFI functions.

3.4.3.5 netWIFI_Scan()

```
void netWIFI_Scan (
             int max_ap,
             int * nets_scanned )
```

Scans the available access points and stores the list in memory.

Parameters

max_ap	Maximum ammount of AP's to store.
nets_scanned	Address where the number of scanned AP's will be written to.

Returns

None.

3.4.3.6 netWIFI_ScannedSSID()

```
Network_t* netWIFI_ScannedSSID (
             int index )
```

Retrieves a scanned available access point previously stored in memory.

Parameters

index	Index of the scanned network stored in memory.

Returns

Scanned network if successfull, NULL otherwise.

Note

This function must be called after netWIFI_Scan, otherwise undefined behaviour is to be expected.

3.5 Menu interfacing

This package provides the means to easily interface menus through an LCD and a rotary encoder.

Macros

- #define UI_MENU_ENCODER_INVERT true
- #define UI_MENU_REFRESH_PERIOD_MS 33
- #define UI_MENU_BLINK_PERIOD_MS 250
- #define UI_MENU_LINES LCDText_LINES
- #define UI_MENU_COLUMNS LCDText_COLUMNS
- #define UI_MENU_SUCCESS_MESSAGE "success"
- #define UI_MENU_FAIL_MESSAGE "failure"
- #define UI_MENU_PROCESSING_MESSAGE "processing..."

Typedefs

- typedef void(∗ ItemHandler_t) (void ∗)
- typedef bool(∗ CommandFunction_t) (void ∗)

Functions

- Menu_t ∗ uiMENU_Generate (char ∗menu, char ∗∗item_names, ItemHandler_t ∗handlers, void ∗∗args, int nr_items)

  Initializes a menu object with the given parameters.
- void uiMENU_Execute (Menu_t ∗menu)

  Executes a previously generated menu.
- bool uiMENU_ExecuteCMD (char ∗message_title, CommandFunction_t function, void ∗args)

  Executes a given function with bool return type and displays the status of its execution.
- void uiMENU_InputData (char ∗message_title, char ∗input_data, int max_chars, char ∗char_table, char ∗output_data)

  Allows the input of data through a rotary encoder.
- int uiMENU_SelectOptions (char ∗menu_name, char ∗∗options, int nr_options, bool cancel)

  Allows the selection of a given ammount of items.
- bool uiMENU_Ask (char ∗question)

  Asks a binary answer question, returning the index of the selected answer.
- void uiMENU_Destroy (Menu_t ∗menu)

  Destroys a previously generated menu object.
- void uiENCODER_Handle (int ∗curr_state, int state_on_left, int state_on_right, int state_on_click, int state←_on_dclick)

  Handles current state based on the encoder interaction with the user.
- char ∗ uiMENU_FillString (char ∗string_to_fill, int col_number)

  Fills a string with empty spaces to prevent junk data from being displayed.
- char ∗ uiMENU_CenterString (char ∗string_to_center, int col_number)

  Centers and fills a string with empty spaces to prevent junk data from being displayed.
- char ∗ uiMENU_ItemizeString (char ∗string_to_itemize, int col_number)

  Centers fills a string with empty spaces and places arrows in the first and last locations of the passed string.
- void uiMENU_BlinkStr (char ∗str_to_blink, int line, int col, int length_to_blink, TickType_t ∗start_time, Tick←Type_t blink_period)

  Alternates between a string and empty spaces in the positions of the display, length and period given.
- void uiMENU_BlinkStrClear (char ∗str_to_blink, int line, int col, int length_to_blink)

  Ensures the last string displayed is the string given rather than an empty one.

3.5.1 Detailed Description

This package provides the means to easily interface menus through an LCD and a rotary encoder.

3.5.2 Macro Definition Documentation

3.5.2.1 UI_MENU_BLINK_PERIOD_MS

```
#define UI_MENU_BLINK_PERIOD_MS 250
```

Blinking period for the menu items, in milliseconds.

3.5.2.2 UI_MENU_COLUMNS

```
#define UI_MENU_COLUMNS LCDText_COLUMNS
```

Ammount of columns in the display used.

3.5.2.3 UI_MENU_ENCODER_INVERT

```
#define UI_MENU_ENCODER_INVERT true
```

Inverts the rotation values from the encoder, from left to right and vice-versa.

3.5.2.4 UI_MENU_FAIL_MESSAGE

```
#define UI_MENU_FAIL_MESSAGE "failure"
```

Message displayed upon failed execution of a given command.

3.5.2.5 UI_MENU_LINES

```
#define UI_MENU_LINES LCDText_LINES
```

Ammount of lines in the display used.

3.5.2.6 UI_MENU_PROCESSING_MESSAGE

```
#define UI_MENU_PROCESSING_MESSAGE "processing..."
```

Message to display while awaiting execution of a given command.

3.5.2.7 UI_MENU_REFRESH_PERIOD_MS

```
#define UI_MENU_REFRESH_PERIOD_MS 33
```

Refresh period for the menus, in milliseconds.

3.5.2.8 UI_MENU_SUCCESS_MESSAGE

```
#define UI_MENU_SUCCESS_MESSAGE "success"
```

Message displayed upon successful execution of a given command.

3.5.3 Typedef Documentation

3.5.3.1 CommandFunction_t

```
typedef bool(* CommandFunction_t) (void *)
```

Handler function type to be used as a command.

3.5.3.2 ItemHandler_t

```
typedef void(* ItemHandler_t) (void *)
```

Handler function type to be used in a menu.

3.5.4 Function Documentation

3.5.4.1 uiENCODER_Handle()

```
void uiENCODER_Handle (
             int * curr_state,
             int state_on_left,
             int state_on_right,
             int state_on_click,
             int state_on_dclick )
```

Handles current state based on the encoder interaction with the user.

Parameters

curr_state	Address of the current state.
state_on_left	State to be assigned to the current state upon left/counter-clockwise rotation of the encoder.
state_on_right	State to be assigned to the current state upon right/clockwise rotation of the encoder.
state_on_click	State to be assigned to the current state upon single button press of the encoder.
state_on_dclick	State to be assigned to the current state upon double rapid button press of the encoder.

Returns

None.

3.5.4.2 uiMENU_Ask()

```
bool uiMENU_Ask (
            char * question )
```

Asks a binary answer question, returning the index of the selected answer.

Parameters

question	Question to be asked.

Returns

Index of the option answer selected.

3.5.4.3 uiMENU_BlinkStr()

```
void uiMENU_BlinkStr (
            char * str_to_blink,
            int line,
            int col,
            int length_to_blink,
            TickType_t * start_time,
            TickType_t blink_period )
```

Alternates between a string and empty spaces in the positions of the display, length and period given.

Parameters

str_to_blink	String to be blinked.
line	Line position in the display of the string to be blinked.
col	Column position in the display of the string to be blinked.
length_to_blink	Length of the string to be blinked.
start_time	Starting time of a blinking phase, in ticks.
blink_period	Blinking period of the display.

Returns

None.

3.5.4.4 uiMENU_BlinkStrClear()

```
void uiMENU_BlinkStrClear (
             char * str_to_blink,
             int line,
             int col,
             int length_to_blink )
```

Ensures the last string displayed is the string given rather than an empty one.

Parameters

str_to_blink	String to be displayed.
line	Line position in the display of the string.
col	Column position in the display of the string.
length_to_blink	Length of the string.

Returns

None.

3.5.4.5 uiMENU_CenterString()

```
char* uiMENU_CenterString (
             char * string_to_center,
             int col_number )
```

Centers and fills a string with empty spaces to prevent junk data from being displayed.

Parameters

string_to_center	String to be filled.
col_number	Number of collumns of the display being used.

Returns

Generated string, dynamically allocated.

3.5.4.6 uiMENU_Destroy()

```
void uiMENU_Destroy (
             Menu_t * menu )
```

Destroys a previously generated menu object.

Parameters

menu	Menu object to be destroyed.

Returns

None.

3.5.4.7 uiMENU_Execute()

```
void uiMENU_Execute (
            Menu_t * menu )
```

Executes a previously generated menu.

Parameters

menu	Menu object to be executed.

Returns

None.

Note

uiMENU_Generate should be called prior to this function to in order to generate the menu object.

3.5.4.8 uiMENU_ExecuteCMD()

```
bool uiMENU_ExecuteCMD (
            char * message_title,
            CommandFunction_t function,
            void * args )
```

Executes a given function with bool return type and displays the status of its execution.

Parameters

message_title	Name of the command to be executed.
function	Handler function of the command to be executed.
args	Argument to the passed handler function.

Returns

True if the command was executed successfully, false otherwise.

Note

uiMENU_Generate should be called prior to this function to in order to generate the menu object.

3.5.4.9 uiMENU_FillString()

```
char* uiMENU_FillString (
             char * string_to_fill,
             int col_number )
```

Fills a string with empty spaces to prevent junk data from being displayed.

Parameters

string_to←_fill	String to be filled.
col_number	Number of collumns of the display being used.

Returns

Generated string, dynamically allocated.

3.5.4.10 uiMENU_Generate()

```
Menu_t* uiMENU_Generate (
             char * menu,
             char ** item_names,
             ItemHandler_t * handlers,
             void ** args,
             int nr_items )
```

Initializes a menu object with the given parameters.

Parameters

menu	Menu name to be displayed.
item_names	Names of the items to be displayed.
handlers	Handler functions of the items.
args	Arguments to be sent to item handler functions.
nr_items	Number of items of the menu.

Returns

Menu object if successful, NULL otherwise.

3.5.4.11 uiMENU_InputData()

```
void uiMENU_InputData (
             char * message_title,
             char * input_data,
             int max_chars,
             char * char_table,
             char * output_data )
```

Allows the input of data through a rotary encoder.

Parameters

message_title	Name of the input prompt.
input_data	Input data to display initially, allowing it to be changed. NULL should be passed to start with no data.
max_chars	Maximum ammount of input characters.
char_table	String containing the allowable characters to be changed.
output_data	Modified data after the input prompt.

Returns

None.

3.5.4.12 uiMENU_ItemizeString()

```
char* uiMENU_ItemizeString (
             char * string_to_itemize,
             int col_number )
```

Centers fills a string with empty spaces and places arrows in the first and last locations of the passed string.

Parameters

string_to_itemize	String to be itemized.
col_number	Number of collumns of the display being used.

Returns

Generated string, dynamically allocated.

3.5.4.13 uiMENU_SelectOptions()

```
int uiMENU_SelectOptions (
             char * menu_name,
             char ** options,
             int nr_options,
             bool cancel )
```

Allows the selection of a given ammount of items.

Parameters

menu_name	Name of selection menu.
options	Names of the items to be displayed.
nr_options	Ammount of options to be given.
cancel	Allows the non selection of any of the given menus.

Returns

Index of the option selected, -1 if an enabled cancellation ocurred.

3.6 FreeRTOS-Network-UI

Modules

- MQTT

  This package provides the ability to interact with remote servers through the MQTT protocol.
- NTP

  This package provides the ability to interact with remote servers through the NTP protocol.
- Transport layer related functions

  This package provides the ability to interact with remote servers through transport layer protocols such as TCP and UDP.
- WiFi related functions

  This package provides the ability to interact with remote access points through WiFi protocols.
- Menu interfacing

  This package provides the means to easily interface menus through an LCD and a rotary encoder.

3.6.1 Detailed Description

Chapter 4

File Documentation

4.1 C:/Users/alext/Desktop/SE/Workspace/FreeRTOS-Network-U↩ I/inc/netmqtt.h File Reference

Contains the netMQTT API.

```
#include <stdbool.h>
#include <time.h>
```

Macros

- #define NETWORK_MQTT_REFRESH_PERIOD_MS 66
- #define NETWORK_MQTT_TASK_STACK_SIZE configMINIMAL_STACK_SIZE ∗ 9
- #define NETWORK_MQTT_QUEUES_SIZE 10
- #define NETWORK_MQTT_ADDRESS_LENGTH 64
- #define NETWORK_MQTT_DEVICE_TOKEN_LENGTH 64
- #define MQTT_NETWORK_CONNECT_TIMEOUT 7000

Functions

- bool netMQTT_Init ()

  Initializes the netMQTT API.
- bool netMQTT_Ready ()

  Checks if the MQTT service is connected to the server and ready to be used.
- void netMQTT_Config (char ∗address, char ∗token, unsigned long port, unsigned long keepalive)

  Configures the credentials to be used in the MQTT protocol.
- void netMQTT_Start ()

  Starts the MQTT task.
- void netMQTT_Stop ()

  Stops the MQTT task.
- bool netMQTT_Publish (char ∗key, char ∗value)

  Publishes data to the previously configured MQTT server.

4.1.1 Detailed Description

Contains the netMQTT API.

Version

1.0

Date

12 jul 2023

Author

Alexandre Silva

4.2 C:/Users/alext/Desktop/SE/Workspace/FreeRTOS-Network-U↩ I/inc/netntp.h File Reference

Contains the netNTP API.

```
#include <time.h>
#include <stdbool.h>
```

Macros

- #define NETWORK_NTP_QUEUES_SIZE 10
- #define NETWORK_NTP_TASK_STACK_SIZE configMINIMAL_STACK_SIZE ∗ 6

Functions

- bool netNTP_Init ()

  Initializes the netNTP API.
- time_t netNTP_GetTime (int hour_adjust)

  Retrieves the number of seconds elapsed since the year 1900 from an NTP remote server.

4.2.1 Detailed Description

Contains the netNTP API.

Version

1.0

Date

12 jul 2023

Author

Alexandre Silva

Copyright(C) 2023, Alexandre Silva All rights reserved.

Software that is described herein is for illustrative purposes only which provides customers with programming information regarding the products. This software is supplied "AS IS" without any warranties.

4.3 C:/Users/alext/Desktop/SE/Workspace/FreeRTOS-Network-U↩ I/inc/nettransport.h File Reference

Contains the netTRANSPORT API.

```
#include <stdbool.h>
```

Macros

- #define NETWORK_UDP_CONNECT_TIMEOUT 10000
- #define NETWORK_TCP_CONNECT_TIMEOUT 10000
- #define NETWORK_RECEIVE_TIMEOUT 20000
- #define NETWORK_SEND_TIMEOUT 20000

Functions

- bool netUDP_Start (const char ∗host, int port)

  Estabilishes a UDP connection with a remote server.
- bool netUDP_Close ()

  Closes an ongoing UDP connection.
- bool netTCP_Start (const char ∗host, const unsigned short int port, const unsigned short int keepalive)

  Estabilishes a TCP connection with a remote server.
- bool netTCP_Close ()

  Closes an ongoing TCP connection.
- int netTRANSPORT_Send (unsigned char ∗address, unsigned int bytes)

  Sends an ammount of data to the desired address through an ongoing transport layer connection.
- int netTRANSPORT_Recv (unsigned char ∗address, unsigned int maxbytes)

  Receives an ammount of data to the desired address through an ongoing transport layer connection.

4.3.1 Detailed Description

Contains the netTRANSPORT API.

Version

1.0

Date

12 jul 2023

Author

Alexandre Silva

Copyright(C) 2023, Alexandre Silva All rights reserved.

Software that is described herein is for illustrative purposes only which provides customers with programming information regarding the products. This software is supplied "AS IS" without any warranties.

4.4 C:/Users/alext/Desktop/SE/Workspace/FreeRTOS-Network-U↩ I/inc/netwifi.h File Reference

Contains the netWIFI API.

```
#include <stdbool.h>
```

Macros

- #define NETWORK_REFRESH_PERIOD_MS 66
- #define NETWORK_WIFI_QUEUES_SIZE 10
- #define NETWORK_WIFI_STACK_SIZE configMINIMAL_STACK_SIZE $*$ 9
- #define NETWORK_MAX_SSID_LENGTH 32
- #define NETWORK_MAX_PASSWORD_LENGTH 64
- #define NETWORK_MAC_ADDRESS_LENGTH 18
- #define NETWORK_MAX_BSSID_LENGTH 18
- #define NETWORK_CONNECT_TIMEOUT 20000
- #define NETWORK_WIFI_SCAN_TIMEOUT 20000
- #define NETWORK_WIFI_CHECK_PERIOD 5000

Functions

- bool netWIFI_Init ()

  Initializes the netWIFI API.
- bool netWIFI_Connect (char ∗ssid, char ∗pass, bool makedefault)

  Estabilishes a WiFi connection with an access point.
- bool netWIFI_Disconnect ()

  Disconnects from a previously connected access point.
- void netWIFI_Check (bool ∗result)

  Checks whether a usable WiFi connection is estabilished.
- void netWIFI_Scan (int max_ap, int ∗nets_scanned)

  Scans the available access points and stores the list in memory.
- Network_t ∗ netWIFI_ScannedSSID (int index)

  Retrieves a scanned available access point previously stored in memory.

4.4.1 Detailed Description

Contains the netWIFI API.

Version

1.0

Date

12 jul 2023

Author

Alexandre Silva

Copyright(C) 2023, Alexandre Silva All rights reserved.

Software that is described herein is for illustrative purposes only which provides customers with programming information regarding the products. This software is supplied "AS IS" without any warranties.

4.5 C:/Users/alext/Desktop/SE/Workspace/FreeRTOS-Network-U↩ I/inc/uimenu.h File Reference

Contains the uiMENU API.

```
#include <FreeRTOS.h>
#include <time.h>
#include <rtoslcd.h>
```

Macros

- #define UI_MENU_ENCODER_INVERT true
- #define UI_MENU_REFRESH_PERIOD_MS 33
- #define UI_MENU_BLINK_PERIOD_MS 250
- #define UI_MENU_LINES LCDText_LINES
- #define UI_MENU_COLUMNS LCDText_COLUMNS
- #define UI_MENU_SUCCESS_MESSAGE "success"
- #define UI_MENU_FAIL_MESSAGE "failure"
- #define UI_MENU_PROCESSING_MESSAGE "processing..."

Typedefs

- typedef void(∗ ItemHandler_t) (void ∗)
- typedef bool(∗ CommandFunction_t) (void ∗)

Functions

- Menu_t ∗ uiMENU_Generate (char ∗menu, char ∗∗item_names, ItemHandler_t ∗handlers, void ∗∗args, int nr_items)

  Initializes a menu object with the given parameters.
- void uiMENU_Execute (Menu_t ∗menu)

  Executes a previously generated menu.
- bool uiMENU_ExecuteCMD (char ∗message_title, CommandFunction_t function, void ∗args)

  Executes a given function with bool return type and displays the status of its execution.
- void uiMENU_InputData (char ∗message_title, char ∗input_data, int max_chars, char ∗char_table, char ∗output_data)

  Allows the input of data through a rotary encoder.
- int uiMENU_SelectOptions (char ∗menu_name, char ∗∗options, int nr_options, bool cancel)

  Allows the selection of a given ammount of items.
- bool uiMENU_Ask (char ∗question)

  Asks a binary answer question, returning the index of the selected answer.
- void uiMENU_Destroy (Menu_t ∗menu)

  Destroys a previously generated menu object.
- void uiENCODER_Handle (int ∗curr_state, int state_on_left, int state_on_right, int state_on_click, int state↩_on_dclick)

  Handles current state based on the encoder interaction with the user.
- char ∗ uiMENU_FillString (char ∗string_to_fill, int col_number)

  Fills a string with empty spaces to prevent junk data from being displayed.
- char ∗ uiMENU_CenterString (char ∗string_to_center, int col_number)

  Centers and fills a string with empty spaces to prevent junk data from being displayed.
- char ∗ uiMENU_ItemizeString (char ∗string_to_itemize, int col_number)

  Centers fills a string with empty spaces and places arrows in the first and last locations of the passed string.
- void uiMENU_BlinkStr (char ∗str_to_blink, int line, int col, int length_to_blink, TickType_t ∗start_time, Tick↩Type_t blink_period)

  Alternates between a string and empty spaces in the positions of the display, length and period given.
- void uiMENU_BlinkStrClear (char ∗str_to_blink, int line, int col, int length_to_blink)

  Ensures the last string displayed is the string given rather than an empty one.

4.5.1 Detailed Description

Contains the uiMENU API.

Version

1.0

Date

12 jul 2023

Author

Alexandre Silva