# MeGaWiFi Application Programming Interface

Generated by Doxygen 1.8.13

# Contents

# Chapter 1

# Todo List

**Module lsd**

Implement UART RTS/CTS handshaking.

Current implementation uses polling. Unfortunately as the Genesis/ Megadrive does not have an interrupt pin on the cart, implementing more efficient data transmission techniques will be tricky.

Proper implementation of error handling.

**Module MegaWiFi**

Missing a lot of integrity checks, also module should track used channels, and is not currently doing it

# Chapter 2

# Module Index

## 2.1 Modules

Here is a list of all modules:

# Chapter 3

# Data Structure Index

## 3.1 Data Structures

Here are the data structures with brief descriptions:

# Chapter 4

# Module Documentation

## 4.1 16c550

Simple 16C550 UART chip driver.

### Modules

- UartRegs

  *16C550 UART registers*
- UartOuts

  *Output pins controlled by the MCR UART register.*
- UartIns

  *Input pins readed in the MSR UART register.*

### Data Structures

- struct UartShadow

  *Structure with the shadow registers.*

### Macros

- #define UART_BASE 0xA130C1

  *16C550 UART base address*
- #define UART_CLK 24000000LU

  *Clock applied to 16C550 chip. Currently using 24 MHz crystal.*
- #define UART_BR 1500000LU
- #define UART_TX_FIFO_LEN 16

  *Length of the TX FIFO in bytes.*
- #define DivWithRounding(dividend, divisor) ((((dividend)$*$2/(divisor))+1)/2)

  *Division with one bit rounding, useful for divisor calculations.*
- #define UART_DLM_VAL (DivWithRounding(UART_CLK, 16 $*$ UART_BR)>>8)

  *Value to load on the UART divisor, high byte.*
- #define UART_DLL_VAL (DivWithRounding(UART_CLK, 16 $*$ UART_BR) & 0xFF)

*Value to load on the UART divisor, low byte.*

- #define UartTxReady() (UART_LSR & 0x20)

  *Checks if UART transmit register/FIFO is ready. In FIFO mode, up to 16 characters can be loaded each time transmitter is ready.*
- #define UartRxReady() (UART_LSR & 0x01)

  *Checks if UART receive register/FIFO has data available.*
- #define UartPutc(c) do{UART_RHR = (c);}while(0);

  *Sends a character. Please make sure there is room in the transmit register/FIFO by calling UartRxReady() before using this function.*
- #define UartGetc() (UART_RHR)

  *Returns a received character. Please make sure data is available by calling UartRxReady() before using this function.*
- #define UartSet(reg, val) do{sh.reg = (val);UART_##reg = (val);}while(0)

  *Sets a value in IER, FCR, LCR or MCR register.*
- #define UartGet(reg) (sh.reg)

  *Gets value of IER, FCR, LCR or MCR register.*
- #define UartSetBits(reg, val)

  *Sets bits in IER, FCR, LCR or MCR register.*
- #define UartClrBits(reg, val)

  *Clears bits in IER, FCR, LCR or MCR register.*
- #define UartResetFifos() UartSetBits(FCR, 0x07)

  *Reset TX and RX FIFOs.*

## Functions

- void UartInit (void)

  *Initializes the driver. The baud rate is set to UART_BR, and the UART FIFOs are enabled. This function must be called before using any other API call.*

## Variables

- UartShadow sh

  *Uart shadow registers. Do NOT access directly!*

### 4.1.1 Detailed Description

Simple 16C550 UART chip driver.

**Author**

Jesus Alonso (doragasu)

**Date**

2016

### 4.1.2 Macro Definition Documentation

#### 4.1.2.1 UART_BR

```
#define UART_BR 1500000LU
```

Desired baud rate. Maximum achievable baudrate with 24 MHz crystal is 24000000/16 = 1.5 Mbps

#### 4.1.2.2 UartClrBits

```
#define UartClrBits(
                reg,
                val )
```

**Value:**

```
do{sh.reg &= ~(val);                    \
                                UART_##reg = sh.reg;}while(0)
```

Clears bits in IER, FCR, LCR or MCR register.

**Parameters**

| | | |
|---|---|---|
| in | *reg* | Register to modify (IER, FCR, LCR or MCR). |
| in | *val* | Bits set in val, will be cleared in reg register. |

#### 4.1.2.3 UartGet

```
#define UartGet(
                reg ) (sh.reg)
```

Gets value of IER, FCR, LCR or MCR register.

**Parameters**

| | | |
|---|---|---|
| in | *reg* | Register to read (IER, FCR, LCR or MCR). |

**Returns**

The value of the requested register.

#### 4.1.2.4 UartGetc

```
#define UartGetc( ) (UART_RHR)
```

Returns a received character. Please make sure data is available by calling UartRxReady() before using this function.

**Returns**

Received character.

### 4.1.2.5 UartPutc

```
#define UartPutc(
             c ) do{UART_RHR = (c);}while(0);
```

Sends a character. Please make sure there is room in the transmit register/FIFO by calling UartRxReady() before using this function.

**Returns**

Received character.

### 4.1.2.6 UartRxReady

```
#define UartRxReady( ) (UART_LSR & 0x01)
```

Checks if UART receive register/FIFO has data available.

**Returns**

TRUE if at least 1 byte is available, FALSE otherwise.

### 4.1.2.7 UartSet

```
#define UartSet(
             reg,
             val ) do{sh.reg = (val);UART_##reg = (val);}while(0)
```

Sets a value in IER, FCR, LCR or MCR register.

**Parameters**

| | | |
|---|---|---|
| in | *reg* | Register to modify (IER, FCR, LCR or MCR). |
| in | *val* | Value to set in IER, FCR, LCR or MCR register. |

#### 4.1.2.8 UartSetBits

```
#define UartSetBits(
              reg,
              val )
```

**Value:**

```
do{sh.reg |= (val);                              \
                             UART_##reg = sh.reg;}while(0)
```

Sets bits in IER, FCR, LCR or MCR register.

**Parameters**

| | | |
|---|---|---|
| in | *reg* | Register to modify (IER, FCR, LCR or MCR). |
| in | *val* | Bits set in val, will be set in reg register. |

#### 4.1.2.9 UartTxReady

```
#define UartTxReady( ) (UART_LSR & 0x20)
```

Checks if UART transmit register/FIFO is ready. In FIFO mode, up to 16 characters can be loaded each time transmitter is ready.

**Returns**

TRUE if transmitter is ready, FALSE otherwise.

## 4.2 lsd

Local Symmetric Data-link. Implements an extremely simple protocol to link two full-duplex devices, multiplexing the data link.

### Modules

- ReturnCodes

  *OK/Error codes returned by several functions.*

### Macros

- #define LSD_OVERHEAD 4

  *LSD frame overhead in bytes.*
- #define LSD_UART 0

  *Uart used for LSD.*
- #define LSD_STX_ETX 0x7E

  *Start/end of transmission character.*
- #define LSD_MAX_CH 4

  *Maximum number of available simultaneous channels.*
- #define LSD_RECV_PRIO 2

  *Receive task priority.*
- #define LSD_MAX_LEN 4095

  *Maximum data payload length.*

### Functions

- void LsdInit (void)
- int LsdChEnable (uint8_t ch)
- int LsdChDisable (uint8_t ch)
- int LsdSend (uint8_t ∗data, uint16_t len, uint8_t ch, uint32_t maxLoopCnt)
- int LsdSplitStart (uint8_t ∗data, uint16_t len, uint16_t total, uint8_t ch, uint32_t maxLoopCnt)
- int LsdSplitNext (uint8_t ∗data, uint16_t len, uint32_t maxLoopCnt)
- int LsdSplitEnd (uint8_t ∗data, uint16_t len, uint32_t maxLoopCnt)
- int LsdRecv (uint8_t ∗buf, uint16_t ∗maxLen, uint32_t maxLoopCnt)

### 4.2.1 Detailed Description

Local Symmetric Data-link. Implements an extremely simple protocol to link two full-duplex devices, multiplexing the data link.

#### Author

Jesus Alonso (doragasu)

#### Date

2016

**Todo** Implement UART RTS/CTS handshaking.

Current implementation uses polling. Unfortunately as the Genesis/ Megadrive does not have an interrupt pin on the cart, implementing more efficient data transmission techniques will be tricky.

Proper implementation of error handling.

### 4.2.2 Function Documentation

#### 4.2.2.1 LsdChDisable()

```
int LsdChDisable (
            uint8_t ch )
```

Disables a channel to stop reception and prohibit sending data.

**Parameters**

| | | |
|---|---|---|
| in | *ch* | Channel number. |

**Returns**

A pointer to an empty TX buffer, or NULL if no buffer is available.

#### 4.2.2.2 LsdChEnable()

```
int LsdChEnable (
            uint8_t ch )
```

Enables a channel to start reception and be able to send data.

**Parameters**

| | | |
|---|---|---|
| in | *ch* | Channel number. |

**Returns**

A pointer to an empty TX buffer, or NULL if no buffer is available.

#### 4.2.2.3 LsdInit()

```
void LsdInit (
            void )
```

Module initialization. Call this function before any other one in this module.

#### 4.2.2.4 LsdRecv()

```
int LsdRecv (
            uint8_t * buf,
            uint16_t * maxLen,
            uint32_t maxLoopCnt )
```

Receives a frame using LSD protocol.

**Parameters**

| | | |
|---|---|---|
| `out` | *buf* | Buffer that will hold the received data. |
| `in,out` | *maxLen* | When calling the function, the variable pointed by maxLen, must hold the maximum number of bytes buf can store. On return, the variable is updated to the number of bytes received. |
| `in` | *maxLoopCnt* | Maximum number of loops trying to read data. |

**Returns**

On success, the number of the channel in which data has been received. On failure, a negative number.

#### 4.2.2.5 LsdSend()

```
int LsdSend (
            uint8_t * data,
            uint16_t len,
            uint8_t ch,
            uint32_t maxLoopCnt )
```

Sends data through a previously enabled channel.

**Parameters**

| | | |
|---|---|---|
| `in` | *data* | Buffer to send. |
| `in` | *len* | Length of the buffer to send. |
| `in` | *ch* | Channel number to use. |
| `in` | *maxLoopCnt* | Maximum number of loops trying to write data. |

**Returns**

-1 if there was an error, or the number of characterse sent otherwise. Note returned value might be 0 if no characters were sent due to maxLoopCnt value reached (timeout).

**Note**

maxLoopCnt value is only used for the wait before starting sending the frame header. For sending the data payload and the ETX, UINT32_MAX value is used for loop counts. If tighter control of the timing is necessary, frame must be sent using split functions.

#### 4.2.2.6 LsdSplitEnd()

```
int LsdSplitEnd (
            uint8_t * data,
            uint16_t len,
            uint32_t maxLoopCnt )
```

Appends (sends) additional data to a frame previously started by an LsdSplitStart() call, and finally ends the frame.

**Parameters**

| | | |
|---|---|---|
| in | *data* | Buffer to send. |
| in | *len* | Length of the data buffer to send. |
| in | *maxLoopCnt* | Maximum number of loops trying to write data. |

**Returns**

-1 if there was an error, or the number of characterse sent otherwise.

#### 4.2.2.7 LsdSplitNext()

```
int LsdSplitNext (
            uint8_t * data,
            uint16_t len,
            uint32_t maxLoopCnt )
```

Appends (sends) additional data to a frame previously started by an LsdSplitStart() call.

**Parameters**

| | | |
|---|---|---|
| in | *data* | Buffer to send. |
| in | *len* | Length of the data buffer to send. |
| in | *maxLoopCnt* | Maximum number of loops trying to write data. |

**Returns**

-1 if there was an error, or the number of characterse sent otherwise.

#### 4.2.2.8 LsdSplitStart()

```
int LsdSplitStart (
            uint8_t * data,
            uint16_t len,
            uint16_t total,
            uint8_t ch,
            uint32_t maxLoopCnt )
```

Starts sending data through a previously enabled channel. Once started, you can send more additional data inside of the frame by issuing as many LsdSplitNext() calls as needed, and end the frame by calling LsdSplitEnd().

**Parameters**

| | | |
|---|---|---|
| `in` | *data* | Buffer to send. |
| `in` | *len* | Length of the data buffer to send. |
| `in` | *total* | Total length of the data to send using a split frame. |
| `in` | *ch* | Channel number to use for sending. |
| `in` | *maxLoopCnt* | Maximum number of loops trying to write data. |

**Returns**

-1 if there was an error, or the number of characterse sent otherwise.

**Note**

maxLoopCnt is only used for the wait before starting sending the frame header. Optional data field is sent using UINT32_MAX as loop count.

## 4.3 megawifi

MeGaWiFi API implementation.

### Modules

- MwCtrlPins

  *Pins used to control WiFi module.*
- MwRetVals

  *Function return values.*

### Data Structures

- struct MwApData

  *Access Point data.*

### Macros

- #define MW_SSID_MAXLEN 32

  *Maximum SSID length (including '\0').*
- #define MW_PASS_MAXLEN 64

  *Maximum password length (including '\0').*
- #define MW_NTP_POOL_MAXLEN 80

  *Maximum length of an NTP pool URI (including '\0').*
- #define MW_NUM_AP_CFGS 3

  *Number of AP configurations stored to nvflash.*
- #define MW_NUM_DNS_SERVERS 2

  *Number of DSN servers supported per AP configuration.*
- #define MW_FSM_QUEUE_LEN 8

  *Length of the FSM queue.*
- #define MW_MAX_SOCK 3

  *Maximum number of simultaneous TCP connections.*
- #define MW_CTRL_CH 0

  *Control channel used for LSD protocol.*
- #define MW_DEF_MAX_LOOP_CNT UINT32_MAX
- #define MW_CMD_MIN_BUFLEN 104
- #define MwSend(ch, data, length)

  *Sends data through a socket, using a previously allocated channel.*
- #define MwModuleReset() do{UartSetBits(MCR, MW__RESET);}while(0)

  *Puts the WiFi module in reset state.*
- #define MwModuleStart() do{UartClrBits(MCR, MW__RESET);}while(0)

  *Releases the module from reset state.*

## Functions

- int MwInit (char ∗cmdBuf, uint16_t bufLen)

  *MwInit Module initialization. Must be called once before using any other function. It also initializes de UART.*
- int MwVersionGet (uint8_t ∗verMajor, uint8_t ∗verMinor, char ∗variant[ ])

  *Obtain module version numbers and string.*
- int MwDefaultCfgSet (void)

  *Set default module configuration.*
- int MwApCfgSet (uint8_t index, const char ssid[ ], const char pass[ ])

  *Set access point configuration (SSID and password).*
- int MwApCfgGet (uint8_t index, char ∗ssid[ ], char ∗pass[ ])

  *Gets access point configuration (SSID and password).*
- int MwIpCfgSet (uint8_t index, const MwIpCfg ∗ip)

  *Set IPv4 configuration.*
- int MwIpCfgGet (uint8_t index, MwIpCfg ∗∗ip)

  *Get IPv4 configuration.*
- int MwApScan (char ∗apData[ ])

  *Scan for access points.*
- int MwApFillNext (char apData[ ], uint16_t pos, MwApData ∗apd, uint16_t dataLen)

  *Parses received AP data and fills information of the AP at "pos". Useful to extract AP information from the data obtained by calling MwApScan() function.*
- int MwApJoin (uint8_t index)

  *Tries joining an AP. If successful, also configures IPv4.*
- int MwApLeave (void)

  *Leaves a previously joined AP.*
- int MwTcpConnect (uint8_t ch, char dstaddr[ ], char dstport[ ], char srcport[ ])

  *Tries establishing a TCP connection with specified server.*
- int MwTcpDisconnect (uint8_t ch)

  *Disconnects a TCP socket from specified channel.*
- int MwTcpBind (uint8_t ch, uint16_t port)

  *Binds a socket to a port, and listens to connections on the port. If a connection request is received, it will be automatically accepted.*
- int MwDataWait (uint32_t maxLoopCnt)

  *Waits until data is received or loop timeout. If data is received, return the channel on which it has been.*
- int MwRecv (uint8_t ∗∗data, uint16_t ∗len, uint32_t maxLoopCnt)

  *Receive data.*
- MwMsgSysStat ∗ MwSysStatGet (void)

  *Get system status.*
- MwSockStat MwSockStatGet (uint8_t ch)

  *Get socket status.*
- int MwSntpCfgSet (char ∗servers[3], uint8_t upDelay, char timezone, char dst)

  *Configure SNTP parameters and timezone.*
- char ∗ MwDatetimeGet (uint32_t dtBin[2])

  *Get date and time.*
- int MwFlashSectorErase (uint16_t sect)

  *Erase a 4 KiB Flash sector. Every byte of an erased sector can be read as 0xFF.*
- int MwFlashWrite (uint32_t addr, uint8_t data[ ], uint16_t dataLen)

  *Write data to specified flash address.*
- uint8_t ∗ MwFlashRead (uint32_t addr, uint16_t dataLen)

  *Read data from specified flash address.*
- int MwCmdSend (MwCmd ∗cmd, uint32_t maxLoopCnt)

  *Send a command to the WiFi module.*
- int MwCmdReplyGet (MwCmd ∗rep, uint32_t maxLoopCnt)

  *Try obtaining a reply to a command.*

### 4.3.1 Detailed Description

MeGaWiFi API implementation.

**Author**

Jesus Alonso (doragasu)

**Date**

2015

**Note**

Module is not reentrant.

**Todo** Missing a lot of integrity checks, also module should track used channels, and is not currently doing it

### 4.3.2 Macro Definition Documentation

#### 4.3.2.1 MW_CMD_MIN_BUFLEN

```
#define MW_CMD_MIN_BUFLEN 104
```

Minimum command buffer length to be able to send all available commands with minimum data payload. This length might not guarantee that commands like MwSntpCfgSet() can be sent if payload length is big enough).

#### 4.3.2.2 MW_DEF_MAX_LOOP_CNT

```
#define MW_DEF_MAX_LOOP_CNT UINT32_MAX
```

Default value of maximum times to try completing a command before desisting

#### 4.3.2.3 MwSend

```
#define MwSend(
              ch,
              data,
              length )
```

**Value:**

```
LsdSend(data, length, ch, \
                                       MW_DEF_MAX_LOOP_CNT)
```

Sends data through a socket, using a previously allocated channel.

**Parameters**

| | | |
|---|---|---|
| in | *ch* | Channel used to send the data. |
| in | *data* | Data to send through channel. |
| in | *length* | Length in bytes of the data field. |

## 4.3.3 Function Documentation

### 4.3.3.1 MwApCfgGet()

```
int MwApCfgGet (
            uint8_t index,
            char * ssid[],
            char * pass[] )
```

Gets access point configuration (SSID and password).

**Parameters**

| | | |
|---|---|---|
| in | *index* | Index of the configuration to get. |
| out | *ssid* | String with the AP SSID got. |
| out | *pass* | String with the AP SSID got. |

**Returns**

MW_OK if configuration successfully got, MW_ERROR otherwise.

**Warning**

ssid is zero padded up to 32 bytes, and pass is zero padded up to 64 bytes. If ssid is 32 bytes, it will NOT be NULL terminated. Also if pass is 64 bytes, it will NOT be NULL terminated.

### 4.3.3.2 MwApCfgSet()

```
int MwApCfgSet (
            uint8_t index,
            const char ssid[],
            const char pass[] )
```

Set access point configuration (SSID and password).

**Parameters**

| | | |
|---|---|---|
| in | *index* | Index of the configuration to set. |
| in | *ssid* | String with the AP SSID to set. |
| in | *pass* | String with the AP SSID to set. |

**Returns**

MW_OK if configuration successfully set, MW_ERROR otherwise.

**Note**

Strings must be NULL terminated. Maximum SSID length is 32 bytes, maximum pass length is 64 bytes.

#### 4.3.3.3 MwApFillNext()

```
int MwApFillNext (
            char apData[],
            uint16_t pos,
            MwApData * apd,
            uint16_t dataLen )
```

Parses received AP data and fills information of the AP at "pos". Useful to extract AP information from the data obtained by calling MwApScan() function.

**Parameters**

| | | |
|---|---|---|
| in | *apData* | Access point data obtained from MwApScan(). |
| in | *pos* | Position at which to extract data. |
| out | *apd* | Pointer to the extracted data from an AP. |
| in | *dataLen* | Lenght of apData. |

**Returns**

Position of the next AP entry in apData, 0 if no more APs available or MW_ERROR if apData/pos combination is not valid.

**Note**

This functions executes locally, does not communicate with the WiFi module.

#### 4.3.3.4 MwApJoin()

```
int MwApJoin (
            uint8_t index )
```

Tries joining an AP. If successful, also configures IPv4.

**Parameters**

|  |  |  |
|---|---|---|
| `in` | *index* | Index of the configuration used to join the AP. |

**Returns**

MW_OK if AP joined successfully and ready to send/receive data, or MW_ERROR if AP join/IP configuration failed.

#### 4.3.3.5 MwApLeave()

```
int MwApLeave (
             void )
```

Leaves a previously joined AP.

**Returns**

MW_OK if AP successfully left, or MW_ERROR if operation failed.

#### 4.3.3.6 MwApScan()

```
int MwApScan (
             char * apData[] )
```

Scan for access points.

**Parameters**

|  |  |  |
|---|---|---|
| `out` | *apData* | Data of the found access points. Each entry has the format specified on the MwApData structure. |

**Returns**

Length in bytes of the output data if operation completes successfully, or MW_ERROR if scan fails.

#### 4.3.3.7 MwCmdReplyGet()

```
int MwCmdReplyGet (
             MwCmd * rep,
             uint32_t maxLoopCnt )
```

Try obtaining a reply to a command.

**Parameters**

| | | |
|---|---|---|
| out | *rep* | Pointer to MwRep structure, containing the reply to the command, if the call completed successfully. |
| in | *maxLoopCnt* | Maximum number of loops trying to read data. |

**Returns**

The channel on which the data has been received (0 if it was on the control channel). Lower than 0 if there was a reception error.

#### 4.3.3.8 MwCmdSend()

```
int MwCmdSend (
             MwCmd * cmd,
             uint32_t maxLoopCnt )
```

Send a command to the WiFi module.

**Parameters**

| | | |
|---|---|---|
| in | *cmd* | Pointer to the filled MwCmd command structure. |
| in | *maxLoopCnt* | Maximum number of loops trying to write command. |

**Returns**

0 if OK. Nonzero if error.

#### 4.3.3.9 MwDataWait()

```
int MwDataWait (
             uint32_t maxLoopCnt )
```

Waits until data is received or loop timeout. If data is received, return the channel on which it has been.

**Parameters**

| | | |
|---|---|---|
| in | *maxLoopCnt* | Maximum number of loop tries before desisting from waiting. Set to 0 avoid waiting if no data is available. |

**Returns**

Channel in which data has been received, or MW_ERROR if an error has occurred.

**Note**

If data has been received on control channel, 0 will be returned.

#### 4.3.3.10 MwDatetimeGet()

```
char* MwDatetimeGet (
            uint32_t dtBin[2] )
```

Get date and time.

**Parameters**

| | | |
|---|---|---|
| out | *dtBin* | Date and time in seconds since Epoch. If set to NULL, this info is not filled (but return value will still be properly set). |

**Returns**

A string with the date and time in textual format, e.g.: "Thu Mar 3 12:26:51 2016”.

#### 4.3.3.11 MwDefaultCfgSet()

```
int MwDefaultCfgSet (
            void )
```

Set default module configuration.

**Returns**

MW_OK if configuration successfully reset, MW_ERROR otherwise.

**Note**

For this command to take effect, it must be followed by a module reset.

#### 4.3.3.12 MwFlashRead()

```
uint8_t* MwFlashRead (
            uint32_t addr,
            uint16_t dataLen )
```

Read data from specified flash address.

**Parameters**

| | | |
|---|---|---|
| in | *addr* | Address from which data will be read. |
| in | *dataLen* | Number of bytes to read from addr. |

**Returns**

Pointer to read data on success, or NULL if command failed.

#### 4.3.3.13 MwFlashSectorErase()

```
int MwFlashSectorErase (
            uint16_t sect )
```

Erase a 4 KiB Flash sector. Every byte of an erased sector can be read as 0xFF.

**Parameters**

| | | |
|---|---|---|
| in | *sect* | Sector number to erase. |

**Returns**

MW_OK if success, MW_ERROR if sector could not be erased.

#### 4.3.3.14 MwFlashWrite()

```
int MwFlashWrite (
            uint32_t addr,
            uint8_t data[],
            uint16_t dataLen )
```

Write data to specified flash address.

**Parameters**

| | | |
|---|---|---|
| in | *addr* | Address to which data will be written. |
| in | *data* | Data to be written to flash chip. |
| in | *dataLen* | Length in bytes of data field. |

**Returns**

MW_OK if success, MW_ERROR if data could not be written.

#### 4.3.3.15 MwInit()

```
int MwInit (
            char * cmdBuf,
            uint16_t bufLen )
```

MwInit Module initialization. Must be called once before using any other function. It also initializes de UART.

**Parameters**

| | | |
|---|---|---|
| in | *cmdBuf* | Pointer to the buffer used to send and receive commands. |
| in | *bufLen* | Length of cmdBuf in bytes. |

**Returns**

0 if Initialization successful, lower than 0 otherwise.

#### 4.3.3.16 MwIpCfgGet()

```
int MwIpCfgGet (
            uint8_t index,
            MwIpCfg ** ip )
```

Get IPv4 configuration.

**Parameters**

| | | |
|---|---|---|
| in | *index* | Index of the configuration to get. |
| out | *ip* | Double pointer to MwIpCfg structure, with IP conf. |

**Returns**

MW_OK if configuration successfully got, MW_ERROR otherwise.

#### 4.3.3.17 MwIpCfgSet()

```
int MwIpCfgSet (
            uint8_t index,
            const MwIpCfg * ip )
```

Set IPv4 configuration.

**Parameters**

| | | |
|---|---|---|
| in | *index* | Index of the configuration to set. |
| in | *ip* | Pointer to the MwIpCfg structure, with IP configuration. |

**Returns**

MW_OK if configuration successfully set, MW_ERROR otherwise.

**Note**

Strings must be NULL terminated. Maximum SSID length is 32 bytes, maximum pass length is 64 bytes.

#### 4.3.3.18 MwRecv()

```
int MwRecv (
            uint8_t ** data,
            uint16_t * len,
            uint32_t maxLoopCnt )
```

Receive data.

**Parameters**

| | | |
|---|---|---|
| out | *data* | Double pointer to received data. |
| out | *len* | Length of the received data. |
| in | *maxLoopCnt* | Maximum number of iterations to try before giving up. Set to 0 to avoid waiting if no data available. |

**Returns**

On success, channel on which data has been received, or MW_ERROR if no data was received.

#### 4.3.3.19 MwSntpCfgSet()

```
int MwSntpCfgSet (
            char * servers[3],
            uint8_t upDelay,
            char timezone,
            char dst )
```

Configure SNTP parameters and timezone.

**Parameters**

| | | |
|---|---|---|
| in | *servers* | Array of up to three NTP servers. If less than three servers are desired, unused entries must be empty. |
| in | *upDelay* | Update delay in seconds. Minimum value is 15. |
| in | *timezone* | Time zone information (from -11 to 13). |
| in | *dst* | Daylight saving. Set to 1 to apply 1 hour offset. |

**Returns**

MW_OK on success, MW_ERROR if command fails.

#### 4.3.3.20 MwSockStatGet()

```
MwSockStat MwSockStatGet (
              uint8_t ch )
```

Get socket status.

**Parameters**

| | | |
|---|---|---|
| in | *ch* | Channel associated to the socket asked for status. |

**Returns**

Socket status data on success, or MW_ERROR on error.

#### 4.3.3.21 MwSysStatGet()

```
MwMsgSysStat* MwSysStatGet (
              void )
```

Get system status.

**Returns**

Pointer to system status structure on success, or NULL on error.

#### 4.3.3.22 MwTcpBind()

```
int MwTcpBind (
              uint8_t ch,
              uint16_t port )
```

Binds a socket to a port, and listens to connections on the port. If a connection request is received, it will be automatically accepted.

**Parameters**

| | | |
|---|---|---|
| in | *ch* | Channel associated to the socket bound t port. |
| in | *port* | Port number to which the socket will be bound. |

**Returns**

MW_OK if socket successfully bound, or MW_ERROR if command failed.

#### 4.3.3.23 MwTcpConnect()

```
int MwTcpConnect (
            uint8_t ch,
            char dstaddr[],
            char dstport[],
            char srcport[] )
```

Tries establishing a TCP connection with specified server.

**Parameters**

| | | |
|---|---|---|
| in | *ch* | Channel used for the connection. |
| in | *dstaddr* | Address (IP or DNS entry) of the server. |
| in | *dstport* | Port in which server is listening. |
| in | *srcport* | Port from which try establishing connection. Set to 0 or empty string for automatic port allocation. |

**Returns**

MW_OK if connection successfully established, or MW_ERROR if connection failed.

#### 4.3.3.24 MwTcpDisconnect()

```
int MwTcpDisconnect (
            uint8_t ch )
```

Disconnects a TCP socket from specified channel.

**Parameters**

| | | |
|---|---|---|
| in | *ch* | Channel associated to the socket to disconnect. |

**Returns**

MW_OK if socket successfully disconnected, or MW_ERROR if command failed.

#### 4.3.3.25 MwVersionGet()

```
int MwVersionGet (
            uint8_t * verMajor,
            uint8_t * verMinor,
            char * variant[] )
```

Obtain module version numbers and string.

**Parameters**

| | | |
|---|---|---|
| out | *verMajor* | Pointer to Major version number. |
| out | *verMinor* | Pointer to Minor version number. |
| out | *variant* | String with firmware variant ("std" for standard). |

**Returns**

MW_OK if completed successfully, MW_ERROR otherwise.

## 4.4 mwmsg

MeGaWiFi command message definitions. Contains the definition of the command codes and the data structures conforming the command message queries and responses.

### Modules

- MwCmds

  *Supported commands.*

### Data Structures

- struct MwMsgInAddr

  *TCP/UDP address message.*
- struct MwIpCfg

  *IP configuration parameters.*
- struct MwMsgApCfg

  *AP configuration message.*
- struct MwMsgIpCfg

  *IP configuration message.*
- struct MwMsgSntpCfg

  *SNTP and timezone configuration.*
- struct MwMsgDateTime

  *Date and time message.*
- struct MwMsgFlashData

  *Flash memory address and data.*
- struct MwMsgFlashRange

  *Flash memory block.*
- struct MwMsgBind

  *Bind message data.*
- union MwMsgSysStat

  *System status.*
- struct MwCmd

  *Command sent to system FSM.*

### Macros

- #define MW_MSG_MAX_BUFLEN 512

  *Maximum buffer length (bytes)*
- #define MW_CMD_HEADLEN (2 ∗ sizeof(uint16_t))

  *Command header length (command code and data length fields).*
- #define MW_CMD_MAX_BUFLEN (MW_MSG_MAX_BUFLEN - MW_CMD_HEADLEN)

  *Maximum data length contained inside command buffer.*
- #define MW_SSID_MAXLEN 32

  *Maximum SSID length (including '\0').*
- #define MW_PASS_MAXLEN 64

  *Maximum password length (including '\0').*

### Enumerations

- enum MwState {
  MW_ST_INIT = 0, MW_ST_IDLE, MW_ST_AP_JOIN, MW_ST_SCAN,
  MW_ST_READY, MW_ST_TRANSPARENT, MW_ST_MAX }

  *MwState Possible states of the system state machine.*
- enum MwSockStat { MW_SOCK_NONE = 0, MW_SOCK_TCP_LISTEN, MW_SOCK_TCP_EST, MW_SO↩
  CK_UDP_READY }

  *Socket status.*

## 4.4.1 Detailed Description

MeGaWiFi command message definitions. Contains the definition of the command codes and the data structures conforming the command message queries and responses.

**Author**

Jesus Alonso (doragasu)

**Date**

2015

## 4.4.2 Enumeration Type Documentation

### 4.4.2.1 MwSockStat

`enum MwSockStat`

Socket status.

**Enumerator**

| | |
|---|---|
| MW_SOCK_NONE | Unused socket. |
| MW_SOCK_TCP_LISTEN | Socket bound and listening. |
| MW_SOCK_TCP_EST | TCP socket, connection established. |
| MW_SOCK_UDP_READY | UDP socket ready for sending/receiving. |

### 4.4.2.2 MwState

`enum MwState`

MwState Possible states of the system state machine.

**Enumerator**

| | |
|---|---|
| MW_ST_INIT | Initialization state. |
| MW_ST_IDLE | Idle state, until connected to an AP. |
| MW_ST_AP_JOIN | Trying to join an access point. |
| MW_ST_SCAN | Scanning access points. |
| MW_ST_READY | Connected to The Internet. |
| MW_ST_TRANSPARENT | Transparent communication state. |
| MW_ST_MAX | Limit number for state machine. |

## 4.5 UartRegs

16C550 UART registers

### Macros

- #define UART_RHR (∗((volatile uint8_t∗)(UART_BASE + 0)))

  *Receiver holding register. Read only.*
- #define UART_THR (∗((volatile uint8_t∗)(UART_BASE + 0)))

  *Transmit holding register. Write only.*
- #define UART_IER (∗((volatile uint8_t∗)(UART_BASE + 2)))

  *Interrupt enable register. Write only.*
- #define UART_FCR (∗((volatile uint8_t∗)(UART_BASE + 4)))

  *FIFO control register. Write only.*
- #define UART_ISR (∗((volatile uint8_t∗)(UART_BASE + 4)))

  *Interrupt status register. Read only.*
- #define UART_LCR (∗((volatile uint8_t∗)(UART_BASE + 6)))

  *Line control register. Write only.*
- #define UART_MCR (∗((volatile uint8_t∗)(UART_BASE + 8)))

  *Modem control register. Write only.*
- #define UART_LSR (∗((volatile uint8_t∗)(UART_BASE + 10)))

  *Line status register. Read only.*
- #define UART_MSR (∗((volatile uint8_t∗)(UART_BASE + 12)))

  *Modem status register. Read only.*
- #define UART_SPR (∗((volatile uint8_t∗)(UART_BASE + 14)))

  *Scratchpad register.*
- #define UART_DLL (∗((volatile uint8_t∗)(UART_BASE + 0)))

  *Divisor latch LSB. Acessed only when LCR[7] = 1.*
- #define UART_DLM (∗((volatile uint8_t∗)(UART_BASE + 2)))

  *Divisor latch MSB. Acessed only when LCR[7] = 1.*

### 4.5.1 Detailed Description

16C550 UART registers

**Note**

Do NOT access IER, FCR, LCR and MCR directly, use Set/Get functions. Remaining registers can be directly accessed, but meeting the read only/write only restrictions.

## 4.6 UartOuts

Output pins controlled by the MCR UART register.

### Macros

- #define UART_MCR__DTR 0x01

  *Data Terminal Ready.*
- #define UART_MCR__RTS 0x02

  *Request To Send.*
- #define UART_MCR__OUT1 0x04

  *GPIO pin 1.*
- #define UART_MCR__OUT2 0x08

  *GPIO pin 2.*

### 4.6.1 Detailed Description

Output pins controlled by the MCR UART register.

## 4.7 UartIns

Input pins readed in the MSR UART register.

#### Macros

- #define UART_MSR__DSR 0x20

  *Data Set Ready.*

### 4.7.1 Detailed Description

Input pins readed in the MSR UART register.

## 4.8 ReturnCodes

OK/Error codes returned by several functions.

### Macros

- #define LSD_OK 0

  *Function completed successfully.*
- #define LSD_ERROR -1

  *Generic error code.*
- #define LSD_FRAMING_ERROR -2

  *A framing error occurred. Possible data loss.*

### 4.8.1 Detailed Description

OK/Error codes returned by several functions.

## 4.9 MwCtrlPins

Pins used to control WiFi module.

### Macros

- #define MW__RESET UART_MCR__OUT1

  *Reset out.*
- #define MW__PRG UART_MCR__OUT2

  *Program out.*
- #define MW__PD UART_MCR__DTR

  *Power Down out.*
- #define MW__DAT UART_MSR__DSR

  *Data request in.*

### 4.9.1 Detailed Description

Pins used to control WiFi module.

## 4.10 MwRetVals

Function return values.

### Macros

- #define MW_OK 0

  *The function completed successfully.*
- #define MW_ERROR -1

  *The function completed with error.*

### 4.10.1 Detailed Description

Function return values.

## 4.11 MwCmds

Supported commands.

### Macros

- #define MW_CMD_OK 0

  *OK command reply.*
- #define MW_CMD_VERSION 1

  *Get firmware version.*
- #define MW_CMD_ECHO 2

  *Echo data.*
- #define MW_CMD_AP_SCAN 3

  *Scan for access points.*
- #define MW_CMD_AP_CFG 4

  *Configure access point.*
- #define MW_CMD_AP_CFG_GET 5

  *Get access point configuration.*
- #define MW_CMD_IP_CFG 6

  *Configure IPv4.*
- #define MW_CMD_IP_CFG_GET 7

  *Get IPv4 configuration.*
- #define MW_CMD_AP_JOIN 8

  *Join access point.*
- #define MW_CMD_AP_LEAVE 9

  *Leave previously joined access point.*
- #define MW_CMD_TCP_CON 10

  *Connect TCP socket.*
- #define MW_CMD_TCP_BIND 11

  *Bind TCP socket to port.*
- #define MW_CMD_TCP_ACCEPT 12

  *Accept incomint TCP connection.*
- #define MW_CMD_TCP_DISC 13

  *Disconnect and free TCP socket.*
- #define MW_CMD_UDP_SET 14

  *Configure UDP socket.*
- #define MW_CMD_UDP_CLR 15

  *Clear and free UDP socket.*
- #define MW_CMD_SOCK_STAT 16

  *Get socket status.*
- #define MW_CMD_PING 17

  *Ping host.*
- #define MW_CMD_SNTP_CFG 18

  *Configure SNTP service.*
- #define MW_CMD_DATETIME 19

  *Get date and time.*
- #define MW_CMD_DT_SET 20

  *Set date and time.*
- #define MW_CMD_FLASH_WRITE 21

*Write to WiFi module flash.*
- #define MW_CMD_FLASH_READ 22

  *Read from WiFi module flash.*
- #define MW_CMD_FLASH_ERASE 23

  *Erase sector from WiFi flash.*
- #define MW_CMD_FLASH_ID 24

  *Get WiFi flash chip identifiers.*
- #define MW_CMD_SYS_STAT 25

  *Get system status.*
- #define MW_CMD_DEF_CFG_SET 26

  *Set default configuration.*
- #define MW_CMD_HRNG_GET 27

  *Gets random numbers.*
- #define MW_CMD_ERROR 255

  *Error command reply.*

### 4.11.1 Detailed Description

Supported commands.

# Chapter 5

# Data Structure Documentation

## 5.1 MwApData Struct Reference

Access Point data.

```
#include <megawifi.h>
```

### Data Fields

- char auth

  *Authentication type.*
- char channel

  *WiFi channel.*
- char str

  *Signal strength.*
- char ssidLen

  *Length of ssid field.*
- char * ssid

  *SSID string (not NULL terminated).*

### 5.1.1 Detailed Description

Access Point data.

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

- megawifi.h

## 5.2 MwCmd Struct Reference

Command sent to system FSM.

```
#include <mw-msg.h>
```

### Data Fields

- uint16_t cmd

  *Command code.*
- uint16_t datalen

  *Data length.*
-

  union {
    uint8_t ch
    uint8_t data [MW_CMD_MAX_BUFLEN]
      *RAW data in uint8_t format.*
    uint32_t dwData [MW_CMD_MAX_BUFLEN/sizeof(uint32_t)]
      *RAW data in uint32_t format.*
    MwMsgInAddr inAddr
      *Internet address.*
    MwMsgApCfg apCfg
      *Access Point configuration.*
    MwMsgIpCfg ipCfg
      *IP configuration.*
    MwMsgSntpCfg sntpCfg
      *SNTP client configuration.*
    MwMsgDateTime datetime
      *Date and time message.*
    MwMsgFlashData flData
      *Flash memory data.*
    MwMsgFlashRange flRange
      *Flash memory range.*
    MwMsgBind bind
      *Bind message.*
    MwMsgSysStat sysStat
      *System status.*
    uint16_t flSect
      *Flash sector.*
    uint32_t flId
      *Flash IDs.*
    uint16_t rndLen
      *Length of the random buffer to fill.*
  };

## 5.2.1 Detailed Description

Command sent to system FSM.

## 5.2.2 Field Documentation

### 5.2.2.1 ch

`uint8_t MwCmd::ch`

Channel number for channel related requests

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

- mw-msg.h

## 5.3 MwIpCfg Struct Reference

IP configuration parameters.

```
#include <mw-msg.h>
```

### Data Fields

- uint32_t addr

  *Host IP address in binary format.*
- uint32_t mask

  *Subnet mask in binary IP format.*
- uint32_t gateway

  *Gateway IP address in binary format.*
- uint32_t dns1

  *DNS server 1 IP address in binary format.*
- uint32_t dns2

  *DNS server 2 IP address in binary format.*

### 5.3.1 Detailed Description

IP configuration parameters.

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

- mw-msg.h

## 5.4 MwMsgApCfg Struct Reference

AP configuration message.

```
#include <mw-msg.h>
```

### Data Fields

- uint8_t cfgNum

  *Configuration number.*
- char ssid [MW_SSID_MAXLEN]

  *SSID string.*
- char pass [MW_PASS_MAXLEN]

  *Password string.*

### 5.4.1 Detailed Description

AP configuration message.

**Warning**

> If ssid length is MW_SSID_MAXLEN, the string will not be NULL terminated. Also if pass length equals MW_PASS_MAXLEN, pass

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

- mw-msg.h

## 5.5 MwMsgBind Struct Reference

Bind message data.

```
#include <mw-msg.h>
```

### Data Fields

- uint32_t reserved

  *Reserved, set to 0.*
- uint16_t port

  *Port to bind to.*
- uint8_t channel

  *Channel used for the socket bound to port.*

### 5.5.1 Detailed Description

Bind message data.

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

- mw-msg.h

## 5.6 MwMsgDateTime Struct Reference

Date and time message.

```
#include <mw-msg.h>
```

### Data Fields

- uint32_t dtBin [2]
- char dtStr [MW_CMD_MAX_BUFLEN - sizeof(uint64_t)]

  *Date and time in textual format.*

### 5.6.1 Detailed Description

Date and time message.

### 5.6.2 Field Documentation

#### 5.6.2.1 dtBin

```
uint32_t MwMsgDateTime::dtBin[2]
```

Number of seconds since Epoch (64-bit)

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

- mw-msg.h

## 5.7 MwMsgFlashData Struct Reference

Flash memory address and data.

```
#include <mw-msg.h>
```

### Data Fields

- uint32_t addr
- uint8_t data [MW_CMD_MAX_BUFLEN - sizeof(uint32_t)]

  *Data associated to the address.*

### 5.7.1 Detailed Description

Flash memory address and data.

### 5.7.2 Field Documentation

#### 5.7.2.1 addr

```
uint32_t MwMsgFlashData::addr
```

Flash memory address

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

- mw-msg.h

## 5.8 MwMsgFlashRange Struct Reference

Flash memory block.

```
#include <mw-msg.h>
```

### Data Fields

- uint32_t addr

  *Start address.*
- uint16_t len

  *Length of the block.*

### 5.8.1 Detailed Description

Flash memory block.

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

- mw-msg.h

## 5.9 MwMsgInAddr Struct Reference

TCP/UDP address message.

```
#include <mw-msg.h>
```

### Data Fields

- char dst_port [6]

  *TCP destination port string.*
- char src_port [6]

  *TCP source port string.*
- uint8_t channel
- char dstAddr [MW_CMD_MAX_BUFLEN - 6 - 6 - 1]

  *Data payload.*

### 5.9.1 Detailed Description

TCP/UDP address message.

### 5.9.2 Field Documentation

#### 5.9.2.1 channel

`uint8_t MwMsgInAddr::channel`

LSD channel used for communications

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

- mw-msg.h

## 5.10 MwMsgIpCfg Struct Reference

IP configuration message.

```
#include <mw-msg.h>
```

### Data Fields

- uint8_t cfgNum
  *Configuration number.*
- uint8_t reserved [3]
  *Reserved (set to 0)*
- MwIpCfg ip
  *IPv4 configuration data.*

### 5.10.1 Detailed Description

IP configuration message.

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

- mw-msg.h

## 5.11 MwMsgSntpCfg Struct Reference

SNTP and timezone configuration.

```
#include <mw-msg.h>
```

### Data Fields

- uint16_t upDelay

    *Update delay in seconds (min: 15)*
- int8_t tz

    *Timezone (from -11 to 13)*
- uint8_t dst
- char servers [MW_CMD_MAX_BUFLEN - 4]

### 5.11.1 Detailed Description

SNTP and timezone configuration.

### 5.11.2 Field Documentation

#### 5.11.2.1 dst

```
uint8_t MwMsgSntpCfg::dst
```

Daylight savines (set to 1 to add 1 hour)

#### 5.11.2.2 servers

```
char MwMsgSntpCfg::servers[MW_CMD_MAX_BUFLEN - 4]
```

Up to 3 NTP server URLs, separated by a NULL character. A double NULL marks the end of the server list.

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

- mw-msg.h

## 5.12 MwMsgSysStat Union Reference

System status.

```
#include <mw-msg.h>
```

### Data Fields

- uint32_t st_flags

  *Accesses all the flags at once.*
-

  struct {
    MwState sys_stat:8
      *System status.*
    uint8_t online:1
      *Module is connected to the Internet.*
    uint8_t cfg_ok:1
      *Configuration OK.*
    uint8_t dt_ok:1
      *Date and time synchronized at least once.*
    uint8_t cfg:2
      *Network configuration set.*
    uint16_t reserved:3
      *Reserved flags.*
    uint16_t ch_ev:16
      *Channel flags with the pending event.*
  };

### 5.12.1 Detailed Description

System status.

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

- mw-msg.h

## 5.13 UartShadow Struct Reference

Structure with the shadow registers.

```
#include <16c550.h>
```

### Data Fields

- uint8_t IER

  *Interrupt Enable Register.*
- uint8_t FCR

  *FIFO Control Register.*
- uint8_t LCR

  *Line Control Register.*
- uint8_t MCR

  *Modem Control Register.*

### 5.13.1 Detailed Description

Structure with the shadow registers.

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

- 16c550.h

# Index