mLink Library Reference Guide for

mLink DHT22 Temperature & Humidity Module (HCMODU0181)

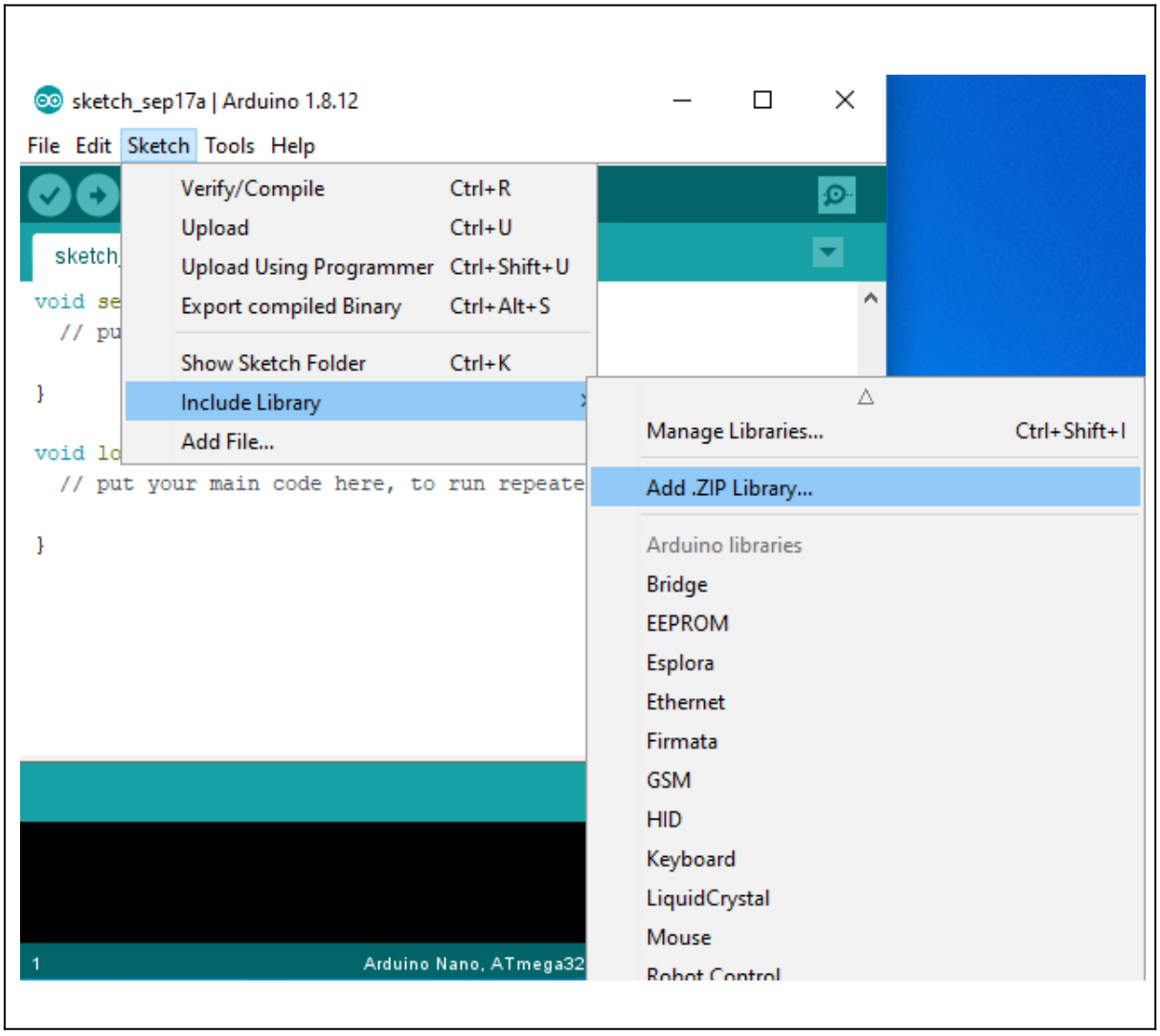
Installing the mLink library

Adding the mLink library to your Arduino IDE can be done in the same way as any other Arduino library:

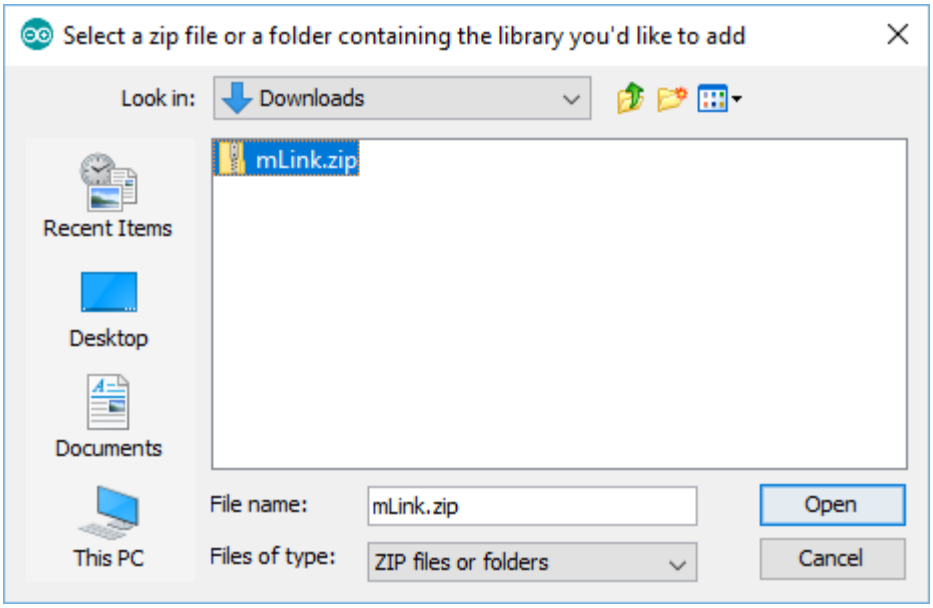
First download the mLink.zip file from the software section of our support forum here:

https://hobbycomponents.com/mLink

Once downloaded, open up your Arduino IDE and go to Sketch->Include Library->Add .ZIP Library.



In the file selection dialogue window that opens, navigate to wherever you downloaded the mLink .zip file and select it, then click the ‘Open’ button.



Including the mLink library in your sketch

Adding the mLink library to your sketch consists of 3 steps; Firstly include the mLink header file (mLink.h) at the top of your sketch, create an instance of the library, then finally initialise the library inside the startup() function:

```
// Step 1: Include the mLink library
#include "mLink.h"


//Step 2: Create an instance of the library
mLink mLink;


void setup()
{
  // Step 3: Initialise the library
  mLink.init();
}


void loop()
{
}
```

Quick library reference table

COMMAND		PARAMETERS	RETURNS
init()	Initialises the mLink library	None	n/a
readBit(*add*, *reg*, *bit*)	Reads the state of a bit from one of the mLink registers	*add* = *byte* value containing I2C address of mLink module *reg* = *byte* value containing register index *bit* = byte value containing the bit number to read (0 to 7)	*boolean* value containing the state of the bit
read(*add*, *reg*)	Reads the contents of one of the mLink registers	*add* = *byte* value containing I2C address of mLink module *reg* = *byte* value containing register index	*byte* value containing the state of the register
readInt(*add*, *reg*)	Reads the contents of 2 consecutive registers and returns the result as an unsigned integer	*add* = byte value containing I2C address of mLink module *reg* = byte value containing register index of the first register	*unsigned integer* containing the values of the two registers
writeBit(*add*, *reg*, *bit*, *state*)	Writes to a bit in one of the mLink registers	*add* = *byte* value containing I2C address of mLink module *reg* = *byte* value containing register index *bit* = *byte* value containing the bit number to write to (0 to 7) *state* = boolean value to set the bit to	n/a
write(*add*, *reg*, *data*)	Writes data to one of the mLink registers	*add* = *byte* value containing I2C address of mLink module *reg* = *byte* value containing register index *data* = *byte* value containing the data to write to the register	n/a
sleep(*add*);	Puts the module into a low power sleep mode.	*add* = *byte* value containing I2C address of mLink module	n/a
busy(*add*);	Checks the state of the busy bit in the status register	*add* = *byte* value containing I2C address of mLink module	Boolean value: 0 = ready 1 = busy
DHT22_Temp(*add*);	Library macro that reads the temperature registers and returns the result in °C	*add* = *byte* value containing I2C address of mLink module	float value containing the temperature in °C to 1dp
DHT22_Hum(*add*);	Library macro that reads the humidity registers and returns the result in %RH	*add* = *byte* value containing I2C address of mLink module	Float value containing the relative humidity to 1dp

mLink 12bit Port Expander Library Commands

mLink.init()

Description

Initialises the mLink library
Add to the setup() section of your sketch to initialise the mLink library

Syntax

mLink.init()

Parameters

None

Returns

Nothing

Example Code

```
void setup()
{
  mLink.init();
}

void loop()
{
}
```

mLink.readBit(add, reg, bit)

Description

Reads the state of a bit from one of the mLink modules 8 bit registers and returns the result as a boolean value.

Parameters

add: byte value containing I2C address of mLink module. Alternatively, if the mLink module is set to its default I2C address (0x51) you can use the predefined value:

DHT22_I2C_ADD

reg: byte value containing the register number to read. You can either specify the register number (see register table) or you can use one of the following predefined values:

MLINK_STATUS_REG

bit: byte value containing the bit number within the specified register to read. Valid values are 0 to 7.

Returns

A boolean value representing the state of the bit.

Example Code

Reads the state of bit 0 (COM error bit) from the status register

```
boolean result = mLink.readBit(DHT22_I2C_ADD, MLINK_STATUS_REG, 0);
```

mLink.read(*add*, *reg*)

Description

Reads the state of one of the mLink modules 8 bit registers and returns the result as a byte.

Parameters

add: byte value containing I2C address of mLink module. Alternatively, if the mLink module is set to its default I2C address (0x51) you can use the predefined value:

DHT22_I2C_ADD

reg: byte value containing the register number to read. You can either specify the register number (see register table) or you can use one of the following predefined values:

MLINK_STATUS_REG
MLINK_ADD_REG
MLINK_MOD_TYPE_REG
MLINK_MOD_SUBTYPE_REG
MLINK_SW_VER_REG
MLINK_DHT22_TEMPH_REG
MLINK_DHT22_TEMPL_REG
MLINK_DHT22_HUMH_REG
MLINK_DHT22_HUML_REG

Returns

A byte value representing the state of the register.

Example Code

Reads the contents of the software version register (register 4)

```
byte result = mLink.read(DHT22_I2C_ADD, MLINK_SW_VER_REG);
```

mLink.readInt(*add*, *reg*)

Description

Reads the state of two consecutive 8 bit registers and returns the result as an unsigned int.

Parameters

add: byte value containing I2C address of mLink module. Alternatively if the mLink module is set to its default I2C address (0x51) you can use the predefined value:

DHT22_I2C_ADD

reg: byte value containing the first register number to read. You can either specify the register number (see register table) or you can use one of the following predefined values:

DHT22_READ_TEMP
DHT22_READ_HUM

Returns

An unsigned int containing both registers where the low byte is the first register and the high byte is the second register.

Example Code

Reads the contents of the two temperature registers (register 11 & 12).

```
unsigned int result = mLink.readInt(DHT22_I2C_ADD, DHT22_READ_TEMP);
```

mLink.write(*add*, *reg*, *data*)

Description

Writes to one of the mLink modules 8 bit registers.

Parameters

add: byte value containing I2C address of mLink module. Alternatively if the mLink module is set to its default I2C address (0x51) you can use the predefined value:

DHT22_I2C_ADD

reg: byte value containing the register number to write to. You can either specify the register number (see register table) or you can use one of the following predefined values:

MLINK_STATUS_REG
MLINK_ADD_REG
MLINK_SLEEP_REG
MLINK_DHT22_DHTREAD_REG

data: byte value containing the data to write to the register

Returns

None

Example Code

Triggers a new temperature and humidity measurement from the DHT22 sensor by writing any value to the MLINK_DHT22_DHTREAD_REG register.

```
mLink.write(DHT22_I2C_ADD, MLINK_DHT22_DHTREAD_REG, 1);
```

Alternatively in the above example of triggering a new measurement you can use the DHT22_START_MEAS definition to simplify the above command:

```
mLink.write(DHT22_I2C_ADD, DHT22_START_MEAS);
```

mLink.sleep(*add*);

Description

Puts the module into a low power sleep mode.

Sleep mode is automatically exited on the next register read or write.

Parameters

add: byte value containing I2C address of mLink module. Alternatively if the mLink module is set to its default I2C address (0x51) you can use the predefined value:

DHT22_I2C_ADD

Returns

None

Example Code

Puts the module into low power sleep mode.

```
mLink.sleep(DHT22_I2C_ADD);
```

mLink.busy(*add*);

Description

Checks the state of the busy bit in the status register. The busy bit is set when a DHT22 measurement is in progress and reset when complete. Therefore this instruction can be used to check when a measurement has completed after being triggered by writing to the DHT read register.

Parameters

add: byte value containing I2C address of mLink module. Alternatively if the mLink module is set to its default I2C address (0x51) you can use the predefined value:

DHT22_I2C_ADD

Returns

None

Example Code

Triggers a DHT22 measurement then uses the mLink.busy() function to wait until the measurement is complete.

```
mLink.write(I2C_ADD, DHT22_START_MEAS);  // Trigger a new measurement

while(mLink.busy(I2C_ADD));              // Wait for the new measurement

float temp = mLink.DHT22_Temp(I2C_ADD);  // Get the temperature in oC
float hum =  mLink.DHT22_Hum(I2C_ADD);   // Get the humidity in %RH
```

mLink.DHT22_Temp(*add*);

Description

Library macro that reads the temperature registers and returns the result in °C.

Parameters

add: byte value containing I2C address of mLink module. Alternatively if the mLink module is set to its default I2C address (0x51) you can use the predefined value:

DHT22_I2C_ADD

Returns

A float containing the last temperature measurement in oC to 1 decimal place.

Example Code

Triggers a DHT22 measurement and reads the temperature when complete.

```
mLink.write(I2C_ADD, DHT22_START_MEAS);  // Trigger a new measurement
while(mLink.busy(I2C_ADD));              // Wait for the new measurement
float temp = mLink.DHT22_Temp(I2C_ADD);  // Get the temperature in oC
```

mLink.DHT22_Hum(*add*);

Description

Library macro that reads the humidity registers and returns the result in °C.

Parameters

add: byte value containing I2C address of mLink module. Alternatively if the mLink module is set to its default I2C address (0x51) you can use the predefined value:

DHT22_I2C_ADD

Returns

A float containing the last humidity measurement in %RH to 1 decimal place.

Example Code

Triggers a DHT22 measurement and reads the humidity when complete.

```
mLink.write(I2C_ADD, DHT22_START_MEAS);  // Trigger a new measurement
while(mLink.busy(I2C_ADD));              // Wait for the new measurement
float hum =  mLink.DHT22_Hum(I2C_ADD);   // Get the humidity in %RH
```

DISCLAIMER

The mLink range is a series of modules intended for the hobbyist and educational markets. Where every care has been taken to ensure the reliability and durability of this product it should not be used in safety or reliability critical applications.

This library and document is provided "as is". Hobby Components Ltd makes no warranties, whether express, implied or statutory, including, but not limited to, implied warranties of merchantability and fitness for a particular purpose, accuracy or lack of negligence. Hobby Components Ltd shall not, in any circumstances, be liable for any damages, including, but not limited to, special, incidental or consequential damages for any reason whatsoever.

COPYRIGHT NOTICE