Ulm Arduino library

Generated by Doxygen 1.9.8

Chapter 1

Ulm_Weatherballoon

This repository is a library implemented for the Arduino Framework. It is used for the implementation of dataloggers for use in high-altitude balloons.

Falko Schmidt, 2023

1.1 Supported electrical components

1.1.1 Sensors

The list below shows all sensors, that are supported by this library. Please note, that this library is only a lightweight addition to the Arduino framework to hide some nast details, that might be confusing for students, meaning that many implementations are defined in other libraries which this depends on.

Name of sensor	Type	Link (Mouser)	Link (DigiKey)	Link (JLCPCB)
DS18B20	Temperature sensor	Part at Mouser	Part at DigiKey	Part at JLCPCB
LSM6DS3	Inertial measurement unit (IMU)	Part at Mouser	Part at DigiKey	Part at JLCPCB
MS5607	Altimeter	Part at Mouser	Part at DigiKey	Part at JLCPCB

1.1.2 Actors

A datalogger not only consists of sensors. There also has to be some output to the user in case of an error. For this, several actors can be used:

Name	Description	Implementation (Header)	Further details
Generic LED	A simple LED that is connected to a GPIO of a microcontroller.	Implementation	Details
Built-In LED	The on-board LED of an Arduino.	Implementation	Details
RGB LED (a.k.a. NeoPixel)	A single NeoPixel RGB LED connected to any GPIO of the microcontroller.	Implementation	Details

Name	Description	Implementation (Header)	Further details
OLED-Display	A small OLED-display that has the dimensions of 128 x 64 pixels.	Implementation	Details

1.1.2.1 Generic LED

An LED can either be ACTIVE_HIGH, meaning that it is ON, if pin is HIGH. Otherwise, it is ACTIVE_LOW, meaning it is ON, if pin is LOW. Please make sure to use the correct setting, otherwise your LED will react inverted! Check the table below to see the valid implementation and according schematic.

Schematic	Minimum running software example

`void setup()  // Init the LED. led.begin();`
`void loop()  // Toggle LED on and of forever. led.on();`
`delay(1000); led.off(); delay(1000);`	

1.1.2.2 Built-in LED

Use this to control the built-in LED, which the Arduino UNO, NANO, MEGA and several others support. You do not need to pass any pin, this will be done automatically.

Schematic	Minimum running software example

Built-in LED.
No additional hardware required.
	`void setup()  // Init the LED. led.begin();`
	`void loop()  // Toggle LED on and of forever. led.on();`
	`delay(1000); led.off(); delay(1000);`

1.1.2.3 RGB LED

This implementation only works for a SINGLE NeoPixel. If you want to control a chain of them, then please refer to the Adafruit NeoPixel library instead! Some predefined methods allow a single RGB LED to be easily used as status indicator.

- **C1** is a decoupling capacitor, which can provide power if needed (for example when changing color, dimming or similar operations) without directly pulling current from the main power supply.
- **R1** is used to prevent voltage spikes on the data line to reach the chips built into the LEDs. To prevent this effect, please use a resistor with a value of 300 - 500. Click here for additional information.
- **R2** is used for the same reasons. According to several experiences online, omitting the resistor results in the LED no longer working.
- Instead of the **WS2813**, also **WS2812** LEDs can be used.

Schematic	Minimum running software example
	`void setup()  Serial.begin(9600); if(!rgb.begin())  Serial.println("Failed to init RGB LED!"); while(1);` `void loop()  rgb.showSuccess(); // Green delay(1000); rgb.showWarning(); // Yellow delay(1000); rgb.showError(); // Red delay(1000);`

1.1.2.4 OLED-Display

This only works for the dimension of 128 x 64 pixels and the i2c address 0x3C. If your display does not apply to these characteristics, then the implementation can not be used!

Schematic	Minimum running software example

	`void setup()  if(display.begin())  display.setCursor(0, 0);`
	`display.print("Hello!");`
	`void loop()  //...`

1.1.3 Storage modules

Of course, sampling sensors it not enough. You need to store your read values somewhere. These implementations can be used to store data:

Name	Description	Implementation (Header)	Further Details
Ulm SD-Storage	SD-Card storage with configurable auto-detect pin and optional data redundancy implementation	Implementation	Details

1.1.3.1 Ulm SD Storage

The Ulm SD Storage describes the hardware structure listed below. It basically consists of an SD card which will be accessed via SPI. An optional detection pin can be used to check, if a card is inserted.

- **C1:** This capacitor is used to buffer the input power for the +3.3V voltage regulator.
- **C2:** A small transistor connected to the BYPASS pin (BP) if the voltage regulator.
- **C3:** This capacitor is used to buffer the generated +3V3 at the output of the voltage regulator.
- **C4 and C5:** Remember the decoupling capacitor described for the RGB LEDs? Same technic applies here. The component 'U2' can suddenly draw more current and therefore require more power. This will be delivered from the capacitors C4 and C5 and not directly influence the main power supply.
- **C6:** SD cards require some power during reading and more during writing processes. Again a capacitor is used to decouple the SD card from the main power source and the voltage regulator. This capacitor must have a size of 10 uF.
- **J1:** The SD-card slot. There are several different components out there. We will use the part 'Hirose DM3D-SF', which is very simple to use and reliable!
- **R1:** Pull-up resistor for the Chip-select / Slave-select (CS / SS) pin of the SD-card. If this pin is low, then the SD-card will react to and process commands sent via SPI. To prevent this from accidentally happen, a pull-up resistor is used. 10 k are appropriate.
- **R2:** Pull-up resistor for the detect pin. If an SD-card is inserted, then the detect pin will be LOW. But what happens, if no SD-card is inserted? This has to be well-defined, and therefore we will use a pull-up resistor of 10 k. This will set the pin to HIGH, if no SD-card is inserted. Please note: Other than the SS pin, which pulls up to +3.3 V, it is necessary, that the detection pin pulls up to the voltage level of the microcontroller. In case of most Arduinos, this is 5 V.
- **U1:** This component is a small voltage regulator, which provides an output voltage of +3.3 V for a wide input voltage (for example +5 V). Now, if you take a close look at the Arduino, you might notice a pin labeled with '3.3 V'. Why not just use that instead? We can not directly use the +3.3 V provided by our Arduino, because it can most likely not provide enough current, so instead we will create our own needed voltage from the +5 V power rail, where we can get more than enough power from!
- **U2:** Okay, we are almost done with the description of this circuit! Only this weird component 'TXB0104' left - what is that? Remember, that our SD-card needs +3.3 V? Our Arduino uses +5 V. And these two components need to exchange data at some point. If we would directly connect the +5 V signals from our Arduino to the SD-card, then we would break it quite fast. Also, the other way round might be a problem: Directly connecting the +3.3 V to an Arduino might be detected as LOW instead of HIGH. The component 'TXB0104' provides a solution to that problem: It translates the Arduino signals (+5 V) into values, that the SD-card will understand (+3.3 V) and vice versa. These components are called logic level converters.

Schematic	Minimum running software example
	`void setup()  Serial.begin(9600); if(!storage.begin())  Serial.println("Storage fail!");` `void loop()  storage.store("This will be stored!"); delay(1000);`

1.2 Supported data-loggers

Below you can find a list of supported data-loggers by this library. Several designs will be added in near future!

1.2.1 SGUduino 2024 (Arduino UNO R4) [Falko Schmidt]

The SGUduino 2024 Datalogger was created for the Seminar NWT course, that took place 2023-2024 at Ulm University. This PCB was created to show students a way how to structure sensors in a neat way on a PCB that can be used with an Arduino UNO. These electrical components are used:

- RGB LED of type 'WS2812'
- RESET button
- DS18B20 temperature sensor
- LSM6DS3 IMU
- microSD card with logic-level-converter
- MS5607 altimeter
- stemmaQT connector for further external sensors.

With the sensor set mentioned above, several different characteristic values of the atmosphere can be measured during the flight in a weather balloon:

- Uptime [ms]
- Surrounding temperature [℃]
- Gyroscope X, Y and Z [deg/s]
- Acceleration X, Y and Z [g]
- Air pressure [mBar]
- Altitude above sea level [m]

If you have this PCB and want to start programming and testing, then check out the ready-to-run program.

1.2.2 Stratduino 2024 (Arduino UNO R3/R4) [Falko Schmidt]

— INFORMATION WILL FOLLOW SOON —

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

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

Chapter 5

Data Structure Documentation

5.1 SensorData Struct Reference

Data Fields

- unsigned long **uptimeMillis**
- float **temperature**
- float **accelerationX**
- float **accelerationY**
- float **accelerationZ**
- float **gyroscopeX**
- float **gyroscopeY**
- float **gyroscopeZ**
- float **temperatureAlt**
- float **pressure**
- float **altitude**

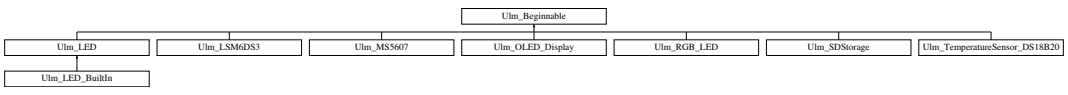
5.1.1 Detailed Description

Structure representing our raw sensor data. All of these values will be sampled and stored to SD card.
The documentation for this struct was generated from the following file:

- src/main.cpp

5.2 Ulm_Beginnable Class Reference

`#include <Ulm_Beginnable.h>`
Inheritance diagram for Ulm_Beginnable:



Public Member Functions

- virtual bool begin ()=0

5.2.1 Detailed Description

All electrical components need some sort of initialisation process after the microcontroller receives power. Sadly, the arduino framework does not provide a common name for an init-method and so there are multiple wild names in libraries, like `begin()`, `init()`, `start()` or others.
To at least provide a uniform name within this library, all components will inherit this class and therefore must implement the virtual method `begin(...)`.

5.2.2 Member Function Documentation

5.2.2.1 begin()

```
virtual bool Ulm_Beginnable::begin ( )  [pure virtual]
```

Initializes the component.

Returns

`true` on, and only on, success without ANY error, `false` otherwise.

Implemented in Ulm_LED, Ulm_OLED_Display, Ulm_RGB_LED, Ulm_LSM6DS3, Ulm_MS5607, Ulm_TemperatureSensor_DS18B20 and Ulm_SDStorage.

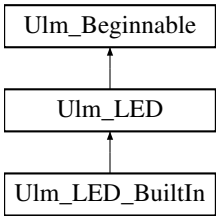
The documentation for this class was generated from the following file:

- src/Ulm_Beginnable.h

5.3 Ulm_LED Class Reference

`#include <Ulm_LED.h>`

Inheritance diagram for Ulm_LED:



Public Types

- enum **Mode** { **ACTIVE_LOW** , **ACTIVE_HIGH** }

Public Member Functions

- Ulm_LED (uint8_t pin, Mode mode=ACTIVE_HIGH)
- bool begin () override
- void on ()
- void off ()
- void toggle () const

5.3.1 Detailed Description

This class represents a simple LED.

Author

Falko Schmidt

Since

1.0

5.3.2 Constructor & Destructor Documentation

5.3.2.1 Ulm_LED()

```
Ulm_LED::Ulm_LED (
            uint8_t pin,
            Ulm_LED::Mode mode = ACTIVE_HIGH )  [explicit]
```

Default constructor to create an object of type Ulm_LED.

Parameters

pin	the physical pin, which the LED is connected to.
mode	optional argument to configure the activation mode of the LED. This can either be `ACTIVE_HIGH`, meaning that the LED will turn on, if pin is set to `HIGH`, or `ACTIVE_LOW`, meaning that the LED will turn on, if pin is set to `LOW`. This value defaults to `ACTIVE_HIGH`.

5.3.3 Member Function Documentation

5.3.3.1 begin()

`bool Ulm_LED::begin ( ) [override], [virtual]`
Initializes LED and turns it off.

Returns

always `true`.

Implements Ulm_Beginnable.

5.3.3.2 off()

`void Ulm_LED::off ( )`
Turns the LED off.

5.3.3.3 on()

`void Ulm_LED::on ( )`
Turns the LED on.

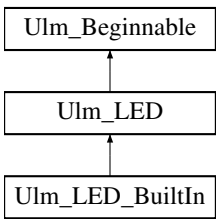
5.3.3.4 toggle()

`void Ulm_LED::toggle ( ) const`
Toggles the LED. If the LED was on before, then calling this method will turn off the LED, otherwise the LED will be turned on.
The documentation for this class was generated from the following files:

- src/actors/Ulm_LED.h
- src/actors/Ulm_LED.cpp

5.4 Ulm_LED_BuiltIn Class Reference

`#include <Ulm_LED_BuiltIn.h>`
Inheritance diagram for Ulm_LED_BuiltIn:



Public Member Functions

- Ulm_LED_BuiltIn ()

Public Member Functions inherited from Ulm_LED

- Ulm_LED (uint8_t pin, Mode mode=ACTIVE_HIGH)
- bool begin () override
- void on ()
- void off ()
- void toggle () const

Additional Inherited Members

Public Types inherited from Ulm_LED

- enum **Mode** { **ACTIVE_LOW** , **ACTIVE_HIGH** }

5.4.1 Detailed Description

This class represents the built-in LED of an Arduino.

Since

1.0

Author

Falko Schmidt

5.4.2 Constructor & Destructor Documentation

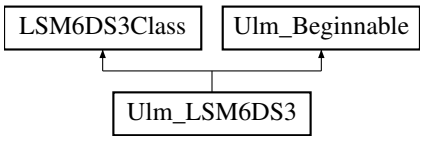
5.4.2.1 Ulm_LED_BuiltIn()

`Ulm_LED_BuiltIn::Ulm_LED_BuiltIn ( )`
Default constructor. This will construct an object of type `Ulm_LED` at pin `LED_BUILTIN`.
The documentation for this class was generated from the following files:

- src/actors/Ulm_LED_BuiltIn.h
- src/actors/Ulm_LED_BuiltIn.cpp

5.5 Ulm_LSM6DS3 Class Reference

`#include <Ulm_LSM6DS3.h>`
Inheritance diagram for Ulm_LSM6DS3:



Public Member Functions

- Ulm_LSM6DS3 (uint8_t address, TwoWire &wire=Wire)
- bool begin () override

5.5.1 Detailed Description

This class represents IMUs of type LSM6DS3.

Author

Falko Schmidt

Since

1.0

5.5.2 Constructor & Destructor Documentation

5.5.2.1 Ulm_LSM6DS3()

```
Ulm_LSM6DS3::Ulm_LSM6DS3 (
            uint8_t address,
            TwoWire & wire = Wire )  [explicit]
```

Constructor.

Parameters

address	the i2c address of the LSM6DS3 sensor.
wire	the i2c interface.

5.5.3 Member Function Documentation

5.5.3.1 begin()

```
bool Ulm_LSM6DS3::begin ( )  [override], [virtual]
```

Inits the sensor.

Returns

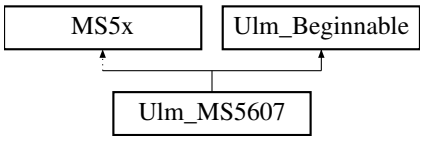
`true` on success, `false` otherwise.

Implements Ulm_Beginnable.
The documentation for this class was generated from the following files:

- src/sensors/Ulm_LSM6DS3.h
- src/sensors/Ulm_LSM6DS3.cpp

5.6 Ulm_MS5607 Class Reference

`#include <Ulm_MS5607.h>`
Inheritance diagram for Ulm_MS5607:



Public Member Functions

- Ulm_MS5607 (int8_t address)
- bool begin () override
- float getTemperature ()
- float getPressure ()
- float getAltitude ()

5.6.1 Detailed Description

This class represents altitude sensors of type MS5607. The pressure will be measured in mBar, temperature is in Celsius.

Author

Falko Schmidt

Since

1.0

5.6.2 Constructor & Destructor Documentation

5.6.2.1 Ulm_MS5607()

```
Ulm_MS5607::Ulm_MS5607 (
            int8_t address )  [explicit]
```

Constructor.

Parameters

address	the i2c address of MS5607 sensor.
aWire	the i2c connection, defaults to `&Wire`.

5.6.3 Member Function Documentation

5.6.3.1 begin()

```
bool Ulm_MS5607::begin ( )  [override], [virtual]
```

Initializes the sensor.

Returns

`true` on success, `false` otherwise.

Implements Ulm_Beginnable.

5.6.3.2 getAltitude()

```
float Ulm_MS5607::getAltitude ( )
```

Calculates the altitude using temperature-compensation.

Returns

the current altitude in meters.

5.6.3.3 getPressure()

```
float Ulm_MS5607::getPressure ( )
```

Reads the current pressure from the sensor.

Returns

the pressure in mBar.

5.6.3.4 getTemperature()

```
float Ulm_MS5607::getTemperature ( )
```

Reads the temperature from the sensor.

Returns

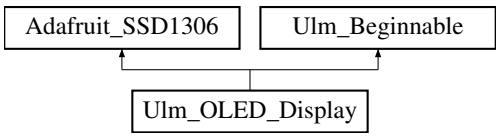
the temperature in Celcius.

The documentation for this class was generated from the following files:

- src/sensors/Ulm_MS5607.h
- src/sensors/Ulm_MS5607.cpp

5.7 Ulm_OLED_Display Class Reference

`#include <Ulm_OLED_Display.h>`
Inheritance diagram for Ulm_OLED_Display:



Public Member Functions

- Ulm_OLED_Display ()
- bool begin () override

5.7.1 Detailed Description

This class represents an OLED with dimensions of 128 x 64 pixels which is connected via i2c at address 0x3C.

Author

Falko Schmidt

Since

1.0

5.7.2 Constructor & Destructor Documentation

5.7.2.1 Ulm_OLED_Display()

`Ulm_OLED_Display::Ulm_OLED_Display ( )`
Default constructor.

5.7.3 Member Function Documentation

5.7.3.1 begin()

`bool Ulm_OLED_Display::begin ( ) [override], [virtual]`
Initializes the display. After calling this method, the display can be written to.

Returns

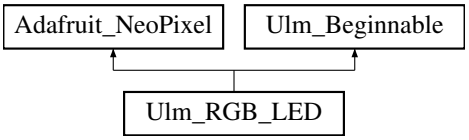
`true` on success, `false` otherwise.

Implements Ulm_Beginnable.
The documentation for this class was generated from the following files:

- src/actors/Ulm_OLED_Display.h
- src/actors/Ulm_OLED_Display.cpp

5.8 Ulm_RGB_LED Class Reference

`#include <Ulm_RGB_LED.h>`
Inheritance diagram for Ulm_RGB_LED:



Public Member Functions

- Ulm_RGB_LED (int16_t pin)
- bool begin () override
- void off ()
- void showSuccess ()
- void showError ()
- void showWarning ()

5.8.1 Detailed Description

This class represents an RGB LED of type WS2812, WS2813 or similar.

Author

Falko Schmidt

Since

1.0

5.8.2 Constructor & Destructor Documentation

5.8.2.1 Ulm_RGB_LED()

```
Ulm_RGB_LED::Ulm_RGB_LED (
            int16_t pin )  [explicit]
```

Default constructor.

Parameters

pin	the pin, which the RGB LED is connected to.

5.8.3 Member Function Documentation

5.8.3.1 begin()

```
bool Ulm_RGB_LED::begin ( )  [override], [virtual]
```

Inits the RGB LED and turns it off.

Returns

always `true`.

Implements Ulm_Beginnable.

5.8.3.2 off()

```
void Ulm_RGB_LED::off ( )
```

Turns the RGB LED off.

5.8.3.3 showError()

```
void Ulm_RGB_LED::showError ( )
```

Shows a red color on the RGB LED.

5.8.3.4 showSuccess()

```
void Ulm_RGB_LED::showSuccess ( )
```

Shows a green color on the RGB LED.

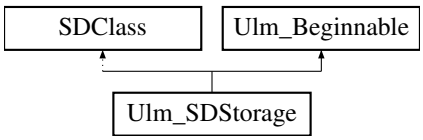
5.8.3.5 showWarning()

`void Ulm_RGB_LED::showWarning ( )`
Shows a yellow color on the RGB LED.
The documentation for this class was generated from the following files:

- src/actors/Ulm_RGB_LED.h
- src/actors/Ulm_RGB_LED.cpp

5.9 Ulm_SDStorage Class Reference

`#include <Ulm_SDStorage.h>`
Inheritance diagram for Ulm_SDStorage:



Data Structures

- class Ulm_SDStorage_Builder
- class Ulm_SDStorage_CSPinBuilder
- class Ulm_SDStorage_DataFileBuilder
- class Ulm_SDStorage_DetectPinBuilder
- class Ulm_SDStorage_DirectoryBuilder
- class Ulm_Storage_OptionalArgsBuilder

Public Types

- enum FileType { **CSV** , **LOG** , **TXT** }

Public Member Functions

- bool begin () override
- bool store (const String &s)
- bool store (const char ∗str)
- void setCsPin (uint8_t csPin)
- void setDetectPin (int8_t detectPin)
- void setDirectory (const String &directory)
- void setDataFileName (const String &dataFileName)
- void setRedundancyFileName (const String &redundancyFileName)
- void setExtremeEnvironmentLogic (bool extremeEnvironmentLogic)

Static Public Member Functions

- static Ulm_SDStorage_CSPinBuilder && builder ()

5.9.1 Detailed Description

This class represents and SD storage with optional detection pin. Some special additions, like data redundancy and other additional actions for extreme environments are implemented to be used in high altitude balloons.
Please note, that rather than directly instantiating this class, you want to use the according builder.

See also

Ulm_SDStorage::builder

Author

Falko Schmidt

Since

1.0

5.9.2 Member Enumeration Documentation

5.9.2.1 FileType

`enum Ulm_SDStorage::FileType`

This enum represents valid and supported file types.

5.9.3 Member Function Documentation

5.9.3.1 begin()

`bool Ulm_SDStorage::begin ( ) [override], [virtual]`

Initialises the SD card, creates the directory and all needed files. After calling this method, you can directly start storing data into files.

Returns

`true` if, and only if, everything went as planned and without ANY error, `false` otherwise.

Implements Ulm_Beginnable.

5.9.3.2 builder()

`Ulm_SDStorage::Ulm_SDStorage_CSPinBuilder && Ulm_SDStorage::builder ( ) [static]`

This static method can be used to receive a builder object for storage creation.

Returns

a new builder.

See also

Ulm_SDStorage_Builder

5.9.3.3 setCsPin()

```
void Ulm_SDStorage::setCsPin (
            uint8_t csPin )
```

Sets the chip-select pin of the SD card.

Parameters

csPin	the physical pin number, which the SD card is connected to.

5.9.3.4 setDataFileName()

```
void Ulm_SDStorage::setDataFileName (
            const String & dataFileName )
```

Sets the file name of primary data file.

Parameters

dataFileName	the data file name.

5.9.3.5 setDetectPin()

```
void Ulm_SDStorage::setDetectPin (
            int8_t detectPin )
```

Sets the detect pin of the SD card slot. If no detect pin is available or unused, then set the parameter to -1.

Parameters

detectPin	the physical pin number, which the SD card slot is connected to.

5.9.3.6 setDirectory()

```
void Ulm_SDStorage::setDirectory (
            const String & directory )
```

Sets the directory, in which all log files are stored. Please note, that subdirectories are not supported. You can either use the root directory ("/") or DIRECT subdirs within the root.

Parameters

directory	

5.9.3.7 setExtremeEnvironmentLogic()

```
void Ulm_SDStorage::setExtremeEnvironmentLogic (
            bool extremeEnvironmentLogic )
```

Sets whether additional actions for extreme environments should be taken or not. Please note, that using this feature will result in overhead in flash and execution time.

Parameters

extremeEnvironmentLogic	whether additional actions should be taken or not.

5.9.3.8 setRedundancyFileName()

```
void Ulm_SDStorage::setRedundancyFileName (
            const String & redundancyFileName )
```

Sets the file name of the optional redundancy file.

Parameters

redundancyFileName	the redundancy file name.

5.9.3.9 store() [1/2]

```
bool Ulm_SDStorage::store (
            const char * str )
```

Stores a const char∗ in data and optional redundancy file.

Parameters

str	the string of type const char∗ to be stored.

Returns

`true` on success, `false` otherwise.

5.9.3.10 store() [2/2]

```
bool Ulm_SDStorage::store (
            const String & s )
```

Stores a string in data and optional redundancy file.

Parameters

s	the string to be stored.

Returns

`true` on success, `false` otherwise.

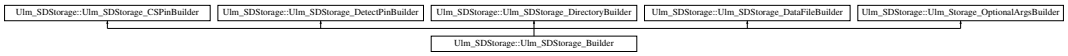
The documentation for this class was generated from the following files:

- src/storage/Ulm_SDStorage.h
- src/storage/Ulm_SDStorage.cpp

5.10 Ulm_SDStorage::Ulm_SDStorage_Builder Class Reference

`#include <Ulm_SDStorage.h>`

Inheritance diagram for Ulm_SDStorage::Ulm_SDStorage_Builder:



Public Member Functions

- Ulm_SDStorage_Builder ()
- Ulm_SDStorage_DetectPinBuilder & atPin (uint8_t pin) override
- Ulm_SDStorage_DirectoryBuilder & withDetectPin (int8_t detPin) override
- Ulm_SDStorage_DirectoryBuilder & withoutDetectPin () override
- Ulm_SDStorage_DataFileBuilder & atDirectory (const String &dir) override
- Ulm_SDStorage_DataFileBuilder & atRootDirectory () override
- Ulm_Storage_OptionalArgsBuilder & withDataFile (const String &dataFilePrefix, enum FileType fileType) override
- Ulm_Storage_OptionalArgsBuilder & withRedundancyFile (const String &redFilePrefix, enum FileType file↩ Type) override
- Ulm_Storage_OptionalArgsBuilder & withLogicForExtremeEnvironments () override
- Ulm_SDStorage build () override

5.10.1 Detailed Description

This class is the actual builder class for Ulm_SDStorage. It is implemented in a step-builder pattern, meaning, that a certain order of method calls is enforced. This enables better bug finding and ensures, that the SD Storage runs with valid and all required values.

Author

Falko Schmidt

Since

1.0

5.10.2 Constructor & Destructor Documentation

5.10.2.1 Ulm_SDStorage_Builder()

```
Ulm_SDStorage::Ulm_SDStorage_Builder::Ulm_SDStorage_Builder ( )
```

Default constructor.

5.10.3 Member Function Documentation

5.10.3.1 atDirectory()

```
Ulm_SDStorage::Ulm_SDStorage_DataFileBuilder & Ulm_SDStorage::Ulm_SDStorage_Builder::at↩
Directory (
            const String & dir )  [override], [virtual]
```

The directory, in which all logs will be created. Please note: The maximum length of the directory name is restricted to 8 chars! Also, the directory will only consist of capital letters, no matter what your input to this method is. Also do not end this string with a '/'. This will be done automatically internally! If the given directory name is invalid, then the root directory will be used instead.

Parameters

dir	the directory name.

Returns

a pointer to this builder to enable method chaining.

Implements Ulm_SDStorage::Ulm_SDStorage_DirectoryBuilder.

5.10.3.2 atPin()

```
Ulm_SDStorage::Ulm_SDStorage_DetectPinBuilder & Ulm_SDStorage::Ulm_SDStorage_Builder::atPin (
            uint8_t pin )  [override], [virtual]
```

Configures the physical pin number, which the chip-select pin of the SD card is connected to.

Parameters

pin	the physical pin of CS of SD card.

Returns

a pointer to this builder to enable method chaining.

Implements Ulm_SDStorage::Ulm_SDStorage_CSPinBuilder.

5.10.3.3 atRootDirectory()

```
Ulm_SDStorage::Ulm_SDStorage_DataFileBuilder & Ulm_SDStorage::Ulm_SDStorage_Builder::atRoot↩
Directory ( )  [override], [virtual]
```

If you do not want to create any additional directory on your SD card, but instead just directly save all files in your top-level directory, then call this method.

Returns

a pointer to this builder to enable method chaining.

Implements Ulm_SDStorage::Ulm_SDStorage_DirectoryBuilder.

5.10.3.4 build()

```
Ulm_SDStorage Ulm_SDStorage::Ulm_SDStorage_Builder::build ( )  [override], [virtual]
```

Finally, we want to create our SD Storage! Once you are happy with your configuration, you can call this method, which will return the SD Storage configured the way to wanted to.

Returns

the SD Storage according to your arguments.

Implements Ulm_SDStorage::Ulm_Storage_OptionalArgsBuilder.

5.10.3.5 withDataFile()

```
Ulm_SDStorage::Ulm_Storage_OptionalArgsBuilder & Ulm_SDStorage::Ulm_SDStorage_Builder::with↩
DataFile (
            const String & dataFilePrefix,
            enum FileType fileType )  [override], [virtual]
```

Configures the data file name using given prefix and file type. If you for example pass these values: prefix: DATA_ and fileType: CSV, then the resulting name will be DATA_000.csv where '000' will automatically be counted up on every start (existing files are not appended to or overwritten).

Parameters

dataFilePrefix	the prefix of the data file.
fileType	the type of the file, that will be created.

Returns

a pointer to this builder to enable method chaining.

Implements Ulm_SDStorage::Ulm_SDStorage_DataFileBuilder.

5.10.3.6 withDetectPin()

```
Ulm_SDStorage::Ulm_SDStorage_DirectoryBuilder & Ulm_SDStorage::Ulm_SDStorage_Builder::with↩
DetectPin (
            int8_t detPin )  [override], [virtual]
```

Configures the physical pin number, which the detection pin of the SD card is connected to. The detection pin will tell you, if an SD card is inserted or not.

Parameters

detectPin	the physical pin of the detection pin of your SD card slot.

Returns

a pointer to this builder to enable method chaining.

Implements Ulm_SDStorage::Ulm_SDStorage_DetectPinBuilder.

5.10.3.7 withLogicForExtremeEnvironments()

```
Ulm_SDStorage::Ulm_Storage_OptionalArgsBuilder & Ulm_SDStorage::Ulm_SDStorage_Builder::with↩
LogicForExtremeEnvironments ( )  [override], [virtual]
```

If you use this class for some extreme environments (cold, hot, high radiation,...), then you can simply call this method to add some additional actions to hopefully safely store your data.

Returns

a pointer to this builder to enable method chaining.

Implements Ulm_SDStorage::Ulm_Storage_OptionalArgsBuilder.

5.10.3.8 withoutDetectPin()

```
Ulm_SDStorage::Ulm_SDStorage_DirectoryBuilder & Ulm_SDStorage::Ulm_SDStorage_Builder::without↩
DetectPin ( )  [override], [virtual]
```

Call this method, if your hardware does not support any detection pin.

Returns

a pointer to this builder to enable method chaining.

Implements Ulm_SDStorage::Ulm_SDStorage_DetectPinBuilder.

5.10.3.9 withRedundancyFile()

```
Ulm_SDStorage::Ulm_Storage_OptionalArgsBuilder & Ulm_SDStorage::Ulm_SDStorage_Builder::with↩
RedundancyFile (
            const String & redFilePrefix,
            enum FileType fileType )  [override], [virtual]
```

Configures optional data redundancy by creating a redundancy file, which will contain the same data as the prior defined data file. This feature is very useful to prevent eventual data corruption, but it comes at the cost of higher flash usage and runtime. The time needed to store the data doubles.

Parameters

redFilePrefix	the prefix of the redundancy file.
fileType	the type of the file, that will be created.

Returns

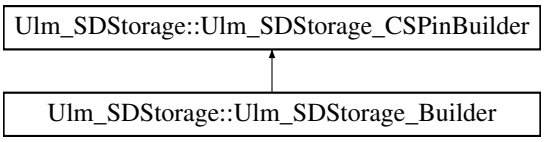
a pointer to this builder to enable method chaining.

Implements Ulm_SDStorage::Ulm_Storage_OptionalArgsBuilder.
The documentation for this class was generated from the following files:

- src/storage/Ulm_SDStorage.h
- src/storage/Ulm_SDStorage.cpp

5.11 Ulm_SDStorage::Ulm_SDStorage_CSPinBuilder Class Reference

`#include <Ulm_SDStorage.h>`
Inheritance diagram for Ulm_SDStorage::Ulm_SDStorage_CSPinBuilder:



Public Member Functions

- virtual Ulm_SDStorage::Ulm_SDStorage_DetectPinBuilder & atPin (uint8_t pin)=0

5.11.1 Detailed Description

This class represents the first mandatory step of building an SD Storage, which is setting the chip-select pin of the SD card.

Author

Falko Schmidt

Since

1.0

5.11.2 Member Function Documentation

5.11.2.1 atPin()

```
virtual Ulm_SDStorage::Ulm_SDStorage_DetectPinBuilder & Ulm_SDStorage::Ulm_SDStorage_CSPin↩
Builder::atPin (
            uint8_t pin )  [pure virtual]
```

Configures the physical pin number, which the chip-select pin of the SD card is connected to.

Parameters

pin	the physical pin of CS of SD card.

Returns

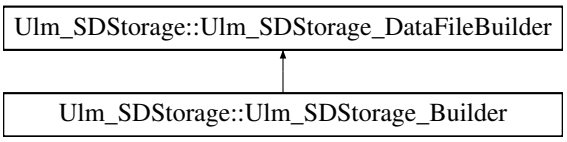
a pointer to this builder to enable method chaining.

Implemented in Ulm_SDStorage::Ulm_SDStorage_Builder.
The documentation for this class was generated from the following file:

- src/storage/Ulm_SDStorage.h

5.12 Ulm_SDStorage::Ulm_SDStorage_DataFileBuilder Class Reference

`#include <Ulm_SDStorage.h>`
Inheritance diagram for Ulm_SDStorage::Ulm_SDStorage_DataFileBuilder:



Public Member Functions

- virtual Ulm_Storage_OptionalArgsBuilder & withDataFile (const String &dataFilePrefix, enum FileType file↩ Type)=0

5.12.1 Detailed Description

This class represents the fourth mandatory step of building an SD Storage, which is configuring the data file name.

Author

Falko Schmidt

Since

1.0

5.12.2 Member Function Documentation

5.12.2.1 withDataFile()

```
virtual Ulm_Storage_OptionalArgsBuilder & Ulm_SDStorage::Ulm_SDStorage_DataFileBuilder::with↩
DataFile (
            const String & dataFilePrefix,
            enum FileType fileType )  [pure virtual]
```

Configures the data file name using given prefix and file type.

Parameters

dataFilePrefix	the prefix of the data file.
fileType	the type of the file, that will be created.

Returns

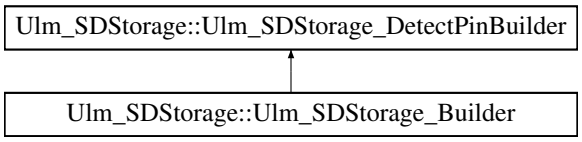
a pointer to this builder to enable method chaining.

Implemented in Ulm_SDStorage::Ulm_SDStorage_Builder.
The documentation for this class was generated from the following file:

- src/storage/Ulm_SDStorage.h

5.13 Ulm_SDStorage::Ulm_SDStorage_DetectPinBuilder Class Reference

`#include <Ulm_SDStorage.h>`
Inheritance diagram for Ulm_SDStorage::Ulm_SDStorage_DetectPinBuilder:



Public Member Functions

- virtual Ulm_SDStorage::Ulm_SDStorage_DirectoryBuilder & withDetectPin (int8_t detectPin)=0
- virtual Ulm_SDStorage::Ulm_SDStorage_DirectoryBuilder & withoutDetectPin ()=0

5.13.1 Detailed Description

This class represents the second mandatory step of building an SD Storage, which is setting a detection pin. To also support other hardware-structures, it is possible to not set any detection pin.

Author

Falko Schmidt

Since

1.0

5.13.2 Member Function Documentation

5.13.2.1 withDetectPin()

```
virtual Ulm_SDStorage::Ulm_SDStorage_DirectoryBuilder & Ulm_SDStorage::Ulm_SDStorage_Detect↩
PinBuilder::withDetectPin (
            int8_t detectPin ) [pure virtual]
```

Configures the physical pin number, which the detection pin of the SD card is connected to. The detection pin will tell you, if an SD card is inserted or not.

Parameters

detectPin	the physical pin of the detection pin of your SD card slot.

Returns

a pointer to this builder to enable method chaining.

Implemented in Ulm_SDStorage::Ulm_SDStorage_Builder.

5.13.2.2 withoutDetectPin()

```
virtual Ulm_SDStorage::Ulm_SDStorage_DirectoryBuilder & Ulm_SDStorage::Ulm_SDStorage_Detect↩
PinBuilder::withoutDetectPin ( )  [pure virtual]
```

Call this method, if your hardware does not support any detection pin.

Returns

a pointer to this builder to enable method chaining.

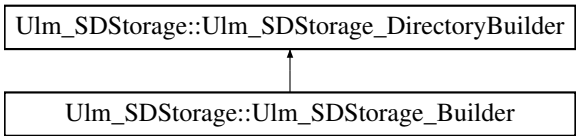
Implemented in Ulm_SDStorage::Ulm_SDStorage_Builder.
The documentation for this class was generated from the following file:

- src/storage/Ulm_SDStorage.h

5.14 Ulm_SDStorage::Ulm_SDStorage_DirectoryBuilder Class Reference

```
#include <Ulm_SDStorage.h>
```

Inheritance diagram for Ulm_SDStorage::Ulm_SDStorage_DirectoryBuilder:



Public Member Functions

- virtual Ulm_SDStorage_DataFileBuilder & atDirectory (const String &dir)=0
- virtual Ulm_SDStorage_DataFileBuilder & atRootDirectory ()=0

5.14.1 Detailed Description

This class represents the third mandatory step of building an SD Storage, which is configuring a directory, in which all files will be created.

Author

Falko Schmidt

Since

1.0

5.14.2 Member Function Documentation

5.14.2.1 atDirectory()

```
virtual Ulm_SDStorage_DataFileBuilder & Ulm_SDStorage::Ulm_SDStorage_DirectoryBuilder::at↩
Directory (
            const String & dir )  [pure virtual]
```

Configures the directory, in which the data and redundancy files will be stored. This allows you to have a somehow ordered structure on your SD card.

Parameters

dir	the directory name.

Returns

a pointer to this builder to enable method chaining.

Implemented in Ulm_SDStorage::Ulm_SDStorage_Builder.

5.14.2.2 atRootDirectory()

```
virtual Ulm_SDStorage_DataFileBuilder & Ulm_SDStorage::Ulm_SDStorage_DirectoryBuilder::at↩
RootDirectory ( )  [pure virtual]
```

If you do not want to create any additional directory on your SD card, but instead just directly save all files in your top-level directory, then call this method.

Returns

a pointer to this builder to enable method chaining.

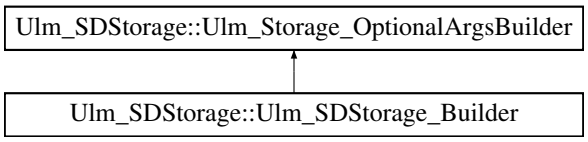
Implemented in Ulm_SDStorage::Ulm_SDStorage_Builder.
The documentation for this class was generated from the following file:

- src/storage/Ulm_SDStorage.h

5.15 Ulm_SDStorage::Ulm_Storage_OptionalArgsBuilder Class Reference

```
#include <Ulm_SDStorage.h>
```

Inheritance diagram for Ulm_SDStorage::Ulm_Storage_OptionalArgsBuilder:



Public Member Functions

- virtual Ulm_SDStorage::Ulm_Storage_OptionalArgsBuilder & withRedundancyFile (const String &redFile↩ Prefix, enum FileType fileType)=0
- virtual Ulm_SDStorage::Ulm_Storage_OptionalArgsBuilder & withLogicForExtremeEnvironments ()=0
- virtual Ulm_SDStorage build ()=0

5.15.1 Detailed Description

This class represents the last step(s) of building an SD storage. All of these features can be used optionally - there is no need to!

Author

Falko Schmidt

Since

1.0

5.15.2 Member Function Documentation

5.15.2.1 build()

```
virtual Ulm_SDStorage Ulm_SDStorage::Ulm_Storage_OptionalArgsBuilder::build ( )  [pure virtual]
```

Finally, we want to create our SD Storage! Once you are happy with your configuration, you can call this method, which will return the SD Storage configured the way to wanted to.

Returns

the SD Storage according to your arguments.

Implemented in Ulm_SDStorage::Ulm_SDStorage_Builder.

5.15.2.2 withLogicForExtremeEnvironments()

```
virtual Ulm_SDStorage::Ulm_Storage_OptionalArgsBuilder & Ulm_SDStorage::Ulm_Storage_Optional↩
ArgsBuilder::withLogicForExtremeEnvironments ( )  [pure virtual]
```

If you use this class for some extreme environments (cold, hot, high radiation,...), then you can simply call this method to add some additional actions to hopefully safely store your data.

Returns

a pointer to this builder to enable method chaining.

Implemented in Ulm_SDStorage::Ulm_SDStorage_Builder.

5.15.2.3 withRedundancyFile()

```
virtual Ulm_SDStorage::Ulm_Storage_OptionalArgsBuilder & Ulm_SDStorage::Ulm_Storage_Optional↩
ArgsBuilder::withRedundancyFile (
            const String & redFilePrefix,
            enum FileType fileType )  [pure virtual]
```

Configures optional data redundancy by creating a redundancy file, which will contain the same data as the prior defined data file. This feature is very useful to prevent eventual data corruption, but it comes at the cost of higher flash usage and runtime. The time needed to store the data doubles.

Parameters

redFilePrefix	the prefix of the redundancy file.
fileType	the type of the file, that will be created.

Returns

a pointer to this builder to enable method chaining.

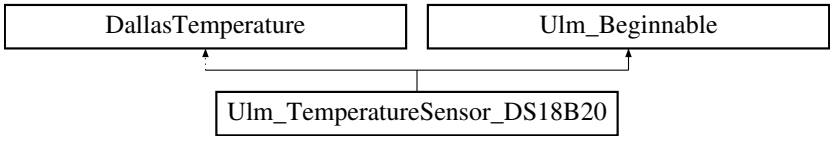
Implemented in Ulm_SDStorage::Ulm_SDStorage_Builder.
The documentation for this class was generated from the following file:

- src/storage/Ulm_SDStorage.h

5.16 Ulm_TemperatureSensor_DS18B20 Class Reference

```
#include <Ulm_TemperatureSensor_DS18B20.h>
```

Inheritance diagram for Ulm_TemperatureSensor_DS18B20:



Public Member Functions

- Ulm_TemperatureSensor_DS18B20 (uint8_t pin)
- bool begin () override
- float getTemperature ()

5.16.1 Detailed Description

This class represents temperature sensors of type DS18B20.

Author

Falko Schmidt

Since

1.0

5.16.2 Constructor & Destructor Documentation

5.16.2.1 Ulm_TemperatureSensor_DS18B20()

```
Ulm_TemperatureSensor_DS18B20::Ulm_TemperatureSensor_DS18B20 (
            uint8_t pin )  [explicit]
```

Constructor.

Parameters

pin	the pin, which the sensor is connected to.

5.16.3 Member Function Documentation

5.16.3.1 begin()

```
bool Ulm_TemperatureSensor_DS18B20::begin ( )  [override], [virtual]
```

Initializes the temperature sensor.

Returns

`true` if everything is successful and exactly one temperature sensor is found, `false` otherwise.

Implements Ulm_Beginnable.

5.16.3.2 getTemperature()

```
float Ulm_TemperatureSensor_DS18B20::getTemperature ( )
```

Reads out the sensor and returns the current temperature.

Returns

the current temperature in Celsius.

The documentation for this class was generated from the following files:

- src/sensors/Ulm_TemperatureSensor_DS18B20.h
- src/sensors/Ulm_TemperatureSensor_DS18B20.cpp

Chapter 6

File Documentation

6.1 Ulm_LED.h

```
00001 //
00002 // Created by Falko Alrik Schmidt on 11.10.23.
00003 //
00004
00005 #ifndef UULM_WEATHERBALOON_ARDUINO_FRAMEWORK_TEST_ULM_LED_H
00006 #define UULM_WEATHERBALOON_ARDUINO_FRAMEWORK_TEST_ULM_LED_H
00007
00008
00009 #include "Ulm_Beginnable.h"
00010
00017 class Ulm_LED : public Ulm_Beginnable {
00018
00019 public:
00020     enum Mode {
00021         ACTIVE_LOW,
00022         ACTIVE_HIGH
00023     };
00024
00025 private:
00029     const uint8_t pin;
00030
00034     const enum Mode mode;
00035
00036 public:
00045     explicit Ulm_LED(uint8_t pin, Mode mode = ACTIVE_HIGH);
00046
00051     bool begin() override;
00052
00056     void on();
00057
00061     void off();
00062
00067     void toggle() const;
00068 };
00069
00070
00071 #endif //UULM_WEATHERBALOON_ARDUINO_FRAMEWORK_TEST_ULM_LED_H
```

6.2 Ulm_LED_BuiltIn.h

```
00001 //
00002 // Created by Falko Alrik Schmidt on 11.10.23.
00003 //
00004
00005 #ifndef UULM_WEATHERBALOON_ARDUINO_FRAMEWORK_TEST_ULM_LED_BUILTIN_H
00006 #define UULM_WEATHERBALOON_ARDUINO_FRAMEWORK_TEST_ULM_LED_BUILTIN_H
00007
00008 #include "Ulm_LED.h"
00009
00016 class Ulm_LED_BuiltIn : public Ulm_LED {
00017
00018 public:
00023     Ulm_LED_BuiltIn();
00024
00025 };
00026
00027
00028 #endif //UULM_WEATHERBALOON_ARDUINO_FRAMEWORK_TEST_ULM_LED_BUILTIN_H
```

6.3 Ulm_OLED_Display.h

```
00001 //
00002 // Created by Falko Alrik Schmidt on 19.09.23.
00003 //
00004
00005 #ifndef UULM_WEATHERBALOON_ARDUINO_FRAMEWORK_TEST_ULM_OLED_DISPLAY_H
00006 #define UULM_WEATHERBALOON_ARDUINO_FRAMEWORK_TEST_ULM_OLED_DISPLAY_H
00007
00008 // This is used, to not show the splash screen
00009 // implemented in the used SSD1306 library.
00010 #define SSD1306_NO_SPLASH
00011
00012 #include "Adafruit_SSD1306.h"
00013 #include "Ulm_Beginnable.h"
00014
00022 class Ulm_OLED_Display : public Adafruit_SSD1306, public Ulm_Beginnable {
00023
00024 public:
00028     Ulm_OLED_Display();
00029
00035     bool begin() override;
00036 };
00037
00038
00039 #endif //UULM_WEATHERBALOON_ARDUINO_FRAMEWORK_TEST_ULM_OLED_DISPLAY_H
```

6.4 Ulm_RGB_LED.h

```
00001 //
00002 // Created by Falko Alrik Schmidt on 19.09.23.
00003 //
00004
00005 #ifndef UULM_WEATHERBALOON_ARDUINO_FRAMEWORK_TEST_ULM_RGB_LED_H
00006 #define UULM_WEATHERBALOON_ARDUINO_FRAMEWORK_TEST_ULM_RGB_LED_H
00007
00008
00009 #include "Adafruit_NeoPixel.h"
00010 #include "Ulm_Beginnable.h"
00011
00018 class Ulm_RGB_LED : public Adafruit_NeoPixel, public Ulm_Beginnable {
00019
00020 public:
00025     explicit Ulm_RGB_LED(int16_t pin);
00026
00031     bool begin() override;
00032
00036     void off();
00037
00041     void showSuccess();
00042
00046     void showError();
00047
00051     void showWarning();
00052
00053 private:
00059     void showColor(uint32_t color);
00060
00061 };
00062
00063
00064 #endif //UULM_WEATHERBALOON_ARDUINO_FRAMEWORK_TEST_ULM_RGB_LED_H
```

6.5 Ulm_LSM6DS3.h

```
00001 //
00002 // Created by Falko Alrik Schmidt on 26.09.23.
00003 //
00004
00005 #ifndef UULM_WEATHERBALOON_ARDUINO_FRAMEWORK_TEST_ULM_LSM6DS3_H
00006 #define UULM_WEATHERBALOON_ARDUINO_FRAMEWORK_TEST_ULM_LSM6DS3_H
00007
00008 #include <Wire.h>
00009 #include "Arduino_LSM6DS3.h"
00010 #include "Ulm_Beginnable.h"
00011
00018 class Ulm_LSM6DS3 : public LSM6DS3Class, public Ulm_Beginnable {
00019
00020 public:
00026     explicit Ulm_LSM6DS3(uint8_t address, TwoWire &wire = Wire);
00027
00032     bool begin() override;
```

```
00033 };
00034
00035
00036 #endif //UULM_WEATHERBALOON_ARDUINO_FRAMEWORK_TEST_ULM_LSM6DS3_H
```

6.6 Ulm_MS5607.h

```
00001 //
00002 // Created by Falko Alrik Schmidt on 26.09.23.
00003 //
00004
00005 #ifndef UULM_WEATHERBALOON_ARDUINO_FRAMEWORK_TEST_ULM_MS5607_H
00006 #define UULM_WEATHERBALOON_ARDUINO_FRAMEWORK_TEST_ULM_MS5607_H
00007
00008 #include <Wire.h>
00009 #include "Ulm_Beginnable.h"
00010 #include "MS5x.h"
00011
00019 class Ulm_MS5607 : MS5x, public Ulm_Beginnable {
00020
00021 public:
00027     explicit Ulm_MS5607(int8_t address);
00028
00033     bool begin() override;
00034
00039     float getTemperature();
00040
00045     float getPressure();
00046
00051     float getAltitude();
00052 };
00053
00054
00055 #endif //UULM_WEATHERBALOON_ARDUINO_FRAMEWORK_TEST_ULM_MS5607_H
```

6.7 Ulm_TemperatureSensor_DS18B20.h

```
00001 //
00002 // Created by Falko Alrik Schmidt on 19.09.23.
00003 //
00004
00005 #ifndef UULM_WEATHERBALOON_ARDUINO_FRAMEWORK_TEST_ULM_TEMPERATURESENSOR_DS18B20_H
00006 #define UULM_WEATHERBALOON_ARDUINO_FRAMEWORK_TEST_ULM_TEMPERATURESENSOR_DS18B20_H
00007
00008
00009 #include "DallasTemperature.h"
00010 #include "Ulm_Beginnable.h"
00011
00018 class Ulm_TemperatureSensor_DS18B20 : DallasTemperature, public Ulm_Beginnable {
00019
00020 private:
00025     OneWire oneWire;
00026
00027 public:
00032     explicit Ulm_TemperatureSensor_DS18B20(uint8_t pin);
00033
00039     bool begin() override;
00040
00045     float getTemperature();
00046 };
00047
00048
00049 #endif //UULM_WEATHERBALOON_ARDUINO_FRAMEWORK_TEST_ULM_TEMPERATURESENSOR_DS18B20_H
```

6.8 Ulm_SDStorage.h

```
00001 //
00002 // Created by Falko Alrik Schmidt on 09.09.23.
00003 //
00004
00005 #ifndef UULM_WEATHERBALOON_ARDUINO_FRAMEWORK_TEST_ULM_SDSTORAGE_H
00006 #define UULM_WEATHERBALOON_ARDUINO_FRAMEWORK_TEST_ULM_SDSTORAGE_H
00007
00008 #include <Arduino.h>
00009 #include "SD.h"
00010 #include "Ulm_Beginnable.h"
00011
00025 class Ulm_SDStorage : SDClass, public Ulm_Beginnable {
00026
```

```
00027 public:
00031     enum FileType {
00032         CSV,
00033         LOG,
00034         TXT
00035     };
00036
00037 public:
00038     class Ulm_SDStorage_Builder;
00039     class Ulm_SDStorage_CSPinBuilder;
00040     class Ulm_SDStorage_DetectPinBuilder;
00041     class Ulm_SDStorage_DirectoryBuilder;
00042     class Ulm_SDStorage_DataFileBuilder;
00043     class Ulm_Storage_OptionalArgsBuilder;
00044
00045 private:
00050     uint8_t csPin;
00051
00056     int8_t detectPin;
00057
00061     String directory;
00062
00066     String dataFileName;
00067
00071     String redundancyFileName;
00072
00077     bool extremeEnvironmentLogic;
00078
00079
00080 private:
00084     Ulm_SDStorage();
00085
00086
00087 public:
00093     //static Ulm_SDStorage_Builder builder();
00094     static Ulm_SDStorage_CSPinBuilder&& builder();
00095
00102     bool begin() override;
00103
00109     bool store(const String& s);
00110
00116     bool store(const char* str);
00117
00122     void setCsPin(uint8_t csPin);
00123
00129     void setDetectPin(int8_t detectPin);
00130
00137     void setDirectory(const String &directory);
00138
00143     void setDataFileName(const String &dataFileName);
00144
00149     void setRedundancyFileName(const String &redundancyFileName);
00150
00158     void setExtremeEnvironmentLogic(bool extremeEnvironmentLogic);
00159
00160
00161 private:
00171     static String determineGenericLogFileName(const String& prefix, Ulm_SDStorage::FileType fileType);
00172
00187     bool determineCurrentLogFileName(const String& dir, String& fileName);
00188
00196     size_t storeToFile(const char *str, const String& dir, const String& fileName);
00197
00202     bool hasRedundancyFile();
00203 };
00204
00205
00206 //
    ------------------------------------------------------------------------------------------------------------------
00207 // FIRST STEP - MANDATORY - SET CS PIN OF SD CARD
00208 //
    ------------------------------------------------------------------------------------------------------------------
00216 class Ulm_SDStorage::Ulm_SDStorage_CSPinBuilder {
00217 public:
00223     virtual Ulm_SDStorage::Ulm_SDStorage_DetectPinBuilder& atPin(uint8_t pin) = 0;
00224 };
00225
00226
00235 class Ulm_SDStorage::Ulm_SDStorage_DetectPinBuilder {
00236 public:
00243     virtual Ulm_SDStorage::Ulm_SDStorage_DirectoryBuilder& withDetectPin(int8_t detectPin) = 0;
00244
00249     virtual Ulm_SDStorage::Ulm_SDStorage_DirectoryBuilder& withoutDetectPin() = 0;
00250 };
00251
00252
00260 class Ulm_SDStorage::Ulm_SDStorage_DirectoryBuilder {
```

```
00261 public:
00268     virtual Ulm_SDStorage_DataFileBuilder& atDirectory(const String& dir) = 0;
00269
00275     virtual Ulm_SDStorage_DataFileBuilder& atRootDirectory() = 0;
00276 };
00277
00278
00286 class Ulm_SDStorage::Ulm_SDStorage_DataFileBuilder {
00287 public:
00294     virtual Ulm_Storage_OptionalArgsBuilder& withDataFile(const String& dataFilePrefix, enum FileType
      fileType) = 0;
00295 };
00296
00297
00305 class Ulm_SDStorage::Ulm_Storage_OptionalArgsBuilder {
00306 public:
00316     virtual Ulm_SDStorage::Ulm_Storage_OptionalArgsBuilder&
00317     withRedundancyFile(const String& redFilePrefix, enum FileType fileType) = 0;
00318
00325     virtual Ulm_SDStorage::Ulm_Storage_OptionalArgsBuilder&
00326     withLogicForExtremeEnvironments() = 0;
00327
00333     virtual Ulm_SDStorage build() = 0;
00334 };
00335
00336
00345 //template <bool foo>
00346 class Ulm_SDStorage::Ulm_SDStorage_Builder : public Ulm_SDStorage_CSPinBuilder,
00347                                              public Ulm_SDStorage_DetectPinBuilder,
00348                                              public Ulm_SDStorage_DirectoryBuilder,
00349                                              public Ulm_SDStorage_DataFileBuilder,
00350                                              public Ulm_Storage_OptionalArgsBuilder {
00351
00352 private:
00360     Ulm_SDStorage sdStorage;
00361
00362 public:
00366     Ulm_SDStorage_Builder();
00367
00368 public:
00374     Ulm_SDStorage_DetectPinBuilder &atPin(uint8_t pin) override;
00375
00376
00383     Ulm_SDStorage_DirectoryBuilder &withDetectPin(int8_t detPin) override;
00384
00389     Ulm_SDStorage_DirectoryBuilder &withoutDetectPin() override;
00390
00402     Ulm_SDStorage_DataFileBuilder &atDirectory(const String& dir) override;
00403
00409     Ulm_SDStorage_DataFileBuilder &atRootDirectory() override;
00410
00420     Ulm_Storage_OptionalArgsBuilder &withDataFile(const String& dataFilePrefix, enum FileType
      fileType) override;
00421
00431     Ulm_Storage_OptionalArgsBuilder &withRedundancyFile(const String& redFilePrefix, enum FileType
      fileType) override;
00432
00439     Ulm_Storage_OptionalArgsBuilder &withLogicForExtremeEnvironments() override;
00440
00446     Ulm_SDStorage build() override;
00447 };
00448
00449 #endif //UULM_WEATHERBALOON_ARDUINO_FRAMEWORK_TEST_ULM_SDSTORAGE_H
```

6.9 Ulm_Beginnable.h

```
00001 //
00002 // Created by Falko Alrik Schmidt on 11.10.23.
00003 //
00004
00005 #ifndef UULM_WEATHERBALOON_ARDUINO_FRAMEWORK_TEST_ULM_BEGINNABLE_H
00006 #define UULM_WEATHERBALOON_ARDUINO_FRAMEWORK_TEST_ULM_BEGINNABLE_H
00007
00018 class Ulm_Beginnable {
00019
00020 public:
00025     virtual bool begin() = 0;
00026
00027 };
00028
00029
00030 #endif //UULM_WEATHERBALOON_ARDUINO_FRAMEWORK_TEST_ULM_BEGINNABLE_H
```

6.10 Ulm_Weatherballoon.h

```
00001 // ============================================================================
00002 // This file is created due to the Arduino library guidelines.
00003 // Including this file will include all relevant header files from this
00004 // library.
00005 //
00006 // Falko Schmidt, 2023
00007 // ============================================================================
00008 #ifndef ULM_WEATHERBALLOON_ULM_WEATHERBALLOON_H
00009 #define ULM_WEATHERBALLOON_ULM_WEATHERBALLOON_H
00010
00011 // -----------------------------------------
00012 // LED
00013 // -----------------------------------------
00014 #include "actors/Ulm_LED.h"
00015 #include "actors/Ulm_LED_BuiltIn.h"
00016 #include "actors/Ulm_RGB_LED.h"
00017
00018 // -----------------------------------------
00019 // SENSORS
00020 // -----------------------------------------
00021 #include "sensors/Ulm_LSM6DS3.h"
00022 #include "sensors/Ulm_MS5607.h"
00023 #include "sensors/Ulm_TemperatureSensor_DS18B20.h"
00024
00025 // -----------------------------------------
00026 // DISPLAYS
00027 // -----------------------------------------
00028 #include "actors/Ulm_OLED_Display.h"
00029
00030 // -----------------------------------------
00031 // STORAGE
00032 // -----------------------------------------
00033 #include "storage/Ulm_SDStorage.h"
00034
00035 #endif //ULM_WEATHERBALLOON_ULM_WEATHERBALLOON_H
```

Index