ELEC351 Environmental Monitoring System - Group D

Generated by Doxygen 1.9.3

Chapter 1

Environmental Monitor

1.1 Link to GitHub Repository

LINK TO GIT REPO

This project is an Environmental Monitor with the capability of monitoring the temperature, pressure, and light level of a space. The software make use of multithreading in order to sample at a deterministic rate. This data is then submitted onto a webpage to be monitored, as well as saved onto an SD card in a human-readable format for latter viewing and processing.

Should any of the defined sensor thresholds be crossed, an alarm will be raised and errors logged to the serial output.

1.2 Requirements

The specification outlines multiple requirments needed for the environmental sensor to be consider fit for purpose. The table belows contains this list of requirements, as well as where they have been fulfilled in the design.

Requirement	Fullfilled	Where
1	Yes	sampler.h
2	Yes	buffer.h
3	Yes	buffer.h
4	Yes - buzzer can be enabled via macro	sampler.h
5	Yes	throughout
6	Yes - no spin locks or busy/waits	throughout
7	Partial - serial capture only	SerialIn.h
8	Partial - internal webpage	HTTP_Server.h
9	Yes - error handler will respond to time outs	throughout
10	Yes	ErrorHandler.h
11	Yes	LEDMatrix.h
12	Yes	throughout

1.3 Dependancies

The main.cpp file contains all the instantiations needed in the correct order. However, several of the modules of this project can be used as standalone classes. To ensure that these classes are instantiated correctly, the dependencies are listed below. Generally, a 'CustomQueue' will need to be instantiated before anything else, to provide an output logging method.

Error Handler

- Custom Queue
  - For error logging

Sampler

- Error Handler
  - To control error outputs

NTPConnection

- Network Interface
  - For creating a network interface
- Custom Queue
  - For connection logging
- Error Handler
  - To control error outpus

HTTP_Server

- Custom Queue
  - For connection logging
- Error Handler
  - To control error outpus
- Sampler
  - To get the samples to output on the webpage
- TCP Socket
  - To allow network connection

SerialIn

- Custom Queue
  - For connection logging
- Sampler
  - To allow the threshold limits to be changed
- Buffer
  - To allow the communication between buffer and serial in to evaluate the data points.

1.4 Usuage of the Environmental Sensor

1.4.1 Error Codes

In the event of an error, the Error Handler will automatically perform actions based on the severity:

Severity	Actions
Warning	Display error code, light yellow LED
Critical	Display error code, light red LED, alarm for 30s then reset
Fatal	Immediate reset

A list of error codes can be seen below:

Module	Severity	Code	Description
Buffer	Critical	10	Buffer is full. This is indicative of a further problem, as the buffer cannot flush to the SD Card output.
	Critical	11	Buffer Lock Timeout.
	Critical	12	Timer Lock Timeout.
	Warning	13	Empty Flush. The buffer has attempted to flush when empty.
	Critical	14	Buffer Flush Timeout. The buffer has failed to acquire the lock in time.
SD Card	Critical	20	No SD Card is mounted.
	Critical	21	No SD Card File. The SD Card may be full.
	Critical	22	SD Card slot is empty, and the buffer cannot flush.
Networking			
	Fatal	40	No network interface found
	Fatal	41	Could not connect to server
	Fatal	42	Failed to get time from NTP server
	Fatal	43	Could not get IP address
	Fatal	44	Listener Error
ErrorHandler	Fatal	99	Flag clear error. The error handler is unresponsive.

The same output is used to display environmental warnings.

Severity	Actions
Environmental	Display error code, Alarm for 3s

A list of environmental errors can be seen below:

Sensor	Code	Description
Temperature	30	Lower temperature limit exceeded.
	31	Upper temperature limit exceeded.
Pressure	33	Lower pressure limit exceeded
	34	Upper pressure limit exceeded
Light	35	Lower light limit exceeded
	36	Upper light limit exceeded

1.4.2 Sending Commands

The Environmental Sensor includes the ability to send commands via a serial interface. Upon initialising, a help screen will be displayed on the connected serial monitor. This list of commands is also available below:

Command	Syntax	Description
latest	latest	Fetch the latest date/time and data sample and display it over serial.
buffered	buffered	Read the number of samples in the buffer and return it.
flush	flush	Write all the samples in the buffer to the SD. Sends acknowledgement after.
set low	set_low (at prompt) - type each limit	Changes lower alarm threshold
set high	set_high (at prompt) - type each limit	Changes upper alarm threshold
plot	plot - (at prompt) char (T/P/L)	Change the matrix display to plot a different sensors data.

1.4.3 Changing the Matrix Display

To change the sensor data that the matrix is displaying, two methods are provided. Either send the serial command 'plot' alongside either T/P/L for the desired sensor. Alternatively, button A can be pressed to toggle to the next sensor. This will take effect on the next sample interval.

1.4.4 Buffer Details

The buffer class contains the SD card functionality, writing to the SD card in a format that is human readblae and easy to edit. The environmental data samples obtained every 10s are buffered in internal memory, and after a minute, when the buffer has reached 90% capacity the samples (tracked by semaphores), along with the time and date they were obtained, are written to the SD card. The thread safe buffer has been encapsulated in one class, using semaphores to track the sample count and space available. Writing to the buffer when full returns an error and lights a red LED. You are warned when attempting to flush an empty buffer. Two threads are use to raise flags for writing to the buffer and flushing. Appropriate mitigation of deadlocks and thread starvation by using timeouts. Timeouts are logged as critical erros in the serial. The ErrorHandler object is referenced by pointer to output the error codes. The CustomQueue is an event queue used to output additional information to the serial.

initSD : Using the sd card detectors interrupt capable pin to check the sd card is mounted. It is also initiated in the bufferClass constructor. green LED lights when mounted, upon removal it turns off. writeBufferAuto : Obtain every data set along with the data and time stamp for when it was taken (aquired from network time server). Writing to the buffer while full, mutex locks that are not acquried or time out will return errors and flash a red led and serial buzzer message is output for 30s. Buffer and time data is protected by mutex locks. flushBuffer : Samples, time and data flushed to text file. An empty flush produces a warning. bufferCount : return number of data sets being held in buffer to be output when requested by SerialIn. fetchLatestRecord : return the latest data set being held in but to be output when requested by SerialIn.

1.4.5 LEDMatrix

Contains a thread-safe LED Matrix class, allowing for the use of the MSB's LED matrix display. The class contains a method for plotting a bar display of the 8 most recent sets of environmental values from the sampler. There are methods for clearing the display, writingMatrix writes the specific bytes to the LED matrix. The plot method plots the internal values as a bar graph. There is an additional test method to check ths is done correctly. The internal method update function uses the updatedSamples to control all of the lights. The matrixThread is the callback method for the MatrixThread thread object, it will run the plot to mimic a hold functionality.

1.5 Contributions

Jack Pendlebury

Authored

- sampler
- ErrorHandler
- CustomQueue

Contributed

- Documentation
- SevenSegmentDisplay
- LEDMatrix
- SerialIn

Noah Harvey

Authored

- SD Card
- Buffer

Contributed

- SerialIn
- ErrorHandler

Luke Waller

Authored

- SevenSegmentDisplay
- LEDMatrix
- NTPConnection
- HTTP_Server
- SerialIn

Contributed

- ErrorHandler
- CustomQueue
- Buffer

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

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

Chapter 4

Class Documentation

4.1 bufferClass Class Reference

buffer class with sd card functionality and dedicated threads for writig to the buffer flushing the sd card

```
#include <buffer.h>
```

Classes

- struct liveData

  Structure for holding sample data, sampled every 10s, and time and data that the data was obtained.

Public Member Functions

- bufferClass (sampler ∗buffersampler, ErrorHandler ∗bufferEH, CustomQueue ∗bufferPQ)

  Construct ErrorHandler, sampler and CustomQueue objects.
- void fetchLatestRecord ()

  return the newest sample set writtin into the buffer along with the time and data so it can be output to the serial
- void flushBuffer ()

  flush all contents of the buffer to the sd card
- void bufferCount ()

  return the number of samples in the buffer to the serial
- void printBufferContents ()

  Outputs the contents of the buffer to serial.
- void initSD ()

  Initiate the sd card by checking that it is mounted.
- void **SDRemoved** ()

  when the interrupt pin for the sd card falls, check the sd card is mounted
- void **flashGreen** ()

  upon flushing of the buffer to the sd card, the green led shall flash

Public Attributes

- samples **sampleData**
- liveData **buffer** [buffer_size]

    buffer of size defined
- liveData **dataRecord**

    for holding data in the buffer
- liveData **printRecord** [buffer_size]

    for printing data to serial
- int **runFlush** = 1
- int **runPrint** = 1
- bool **SDMount**

Private Types

- typedef void(∗ **funcPointer_t**) (void)

Private Member Functions

- void **writeFlag** ()

    setting flag to initiate writing data into the buffer
- void **flushFlag** ()

    setting flag to initiate flush to sd card
- void **writeBufferAuto** ()

    waits for the write flag to indicate that there are new samples to be written into the buffer, then adds the environmental data into the next space
- void whenToFlush ()

    waits for the flush flag and checks if a minute has passed since the last flush and that buffer is at 90% capacity.

Private Attributes

- Mutex **bufferLock**

    mutex buffer lock to protect data in buffer
- Mutex **timeLock**

    mutex time lock to protect time
- Ticker **bufferWriteTick**

    ticker for writing into the buffer every 15s
- Ticker **bufferFlushTick**

    ticker for checking buffer capacity every 60s
- Timer **t**

    timer for tracking time in between flushes
- time_t **timestamp**

    to obtain real time
- InterruptIn **SDDetector**

    to check if sd card is mounted
- sampler ∗ **bSamp**

    pointer to sampler
- ErrorHandler ∗ **BEH**

    pointer to the error handler

- CustomQueue ∗ **PQ**

    pointer to the output error queue
- Thread **writeThread**
- Thread **flushThread**
- unsigned int newIDX

    number of slots in buffer, both used to track position and next space available.
- unsigned int **oldIDX** = buffer_size - 1
- float **flushTiming**

    time since last flush
- float **hourPassed** = 59 ∗ 60

    hour timer

4.1.1 Detailed Description

buffer class with sd card functionality and dedicated threads for writig to the buffer flushing the sd card

4.1.2 Constructor & Destructor Documentation

4.1.2.1 bufferClass()

```
bufferClass::bufferClass (
            sampler ∗ buffersampler,
            ErrorHandler ∗ bufferEH,
            CustomQueue ∗ bufferPQ )
```

Construct ErrorHandler, sampler and CustomQueue objects.

This constructor must be given a pointer to each in order. ErrorHandler - outputs errors over serial. sampler - obtain new enviromental samples. CustomQueue - outputs additional information over serial.

4.1.3 Member Function Documentation

4.1.3.1 bufferCount()

```
void bufferClass::bufferCount ( )
```

return the number of samples in the buffer to the serial

Parameters

newIDX	holds the number of datasets in the buffer

4.1.3.2 fetchLatestRecord()

```
void bufferClass::fetchLatestRecord ( )
```

return the newest sample set writtin into the buffer along with the time and data so it can be output to the serial

Parameters

newIDX	the position of latest set of data to be written into the buffer

4.1.3.3 flushBuffer()

```
void bufferClass::flushBuffer ( )
```

flush all contents of the buffer to the sd card

Parameters

runFlush	1 to continue iterating through all samples sets, 0 to terminate as there is no data left

4.1.3.4 initSD()

```
void bufferClass::initSD ( )
```

Initiate the sd card by checking that it is mounted.

Parameters

SDDetector	to fire an interrupt when the sd card is not mounted, calling initSD to check
SDMount	1 - sd card is mounted, 0 - the sd card is not mounted

4.1.3.5 printBufferContents()

```
void bufferClass::printBufferContents ( )
```

Outputs the contents of the buffer to serial.

This was used for testing purposes.

Parameters

runPrint	1 to continue iteration until empty, 0 to terminate when empty.

4.1.3.6 whenToFlush()

```
void bufferClass::whenToFlush ( )  [private]
```

waits for the flush flag and checks if a minute has passed since the last flush and that buffer is at 90% capacity.

otherwise it will flush after an hour.

4.1.4 Member Data Documentation

4.1.4.1 newIDX

```
unsigned int bufferClass::newIDX  [private]
```

Initial value:

```
=
     buffer_size - 1
```

number of slots in buffer, both used to track position and next space available.

track number of data sets in the buffer

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

- buffer.h
- buffer.cpp

4.2 CustomQueue Class Reference

Event Queue wrapper class.

```
#include <CustomQueue.h>
```

Public Member Functions

- CustomQueue ()
  Default constructor.

Public Attributes

- EventQueue **custom**

    Event Queue Object. This class is a wrapper for this object.
- Thread **QUEUE_THREAD**

    Thread Object. This thread is solely responsible for the queue.

4.2.1 Detailed Description

Event Queue wrapper class.

An example call is shown below:

```
CustomQueue* queue;
queue->custom.call(printf, "FATAL Error Code - %d\n", (errorNumber & 255));
```

4.2.2 Constructor & Destructor Documentation

4.2.2.1 CustomQueue()

```
CustomQueue::CustomQueue ( )
```

Default constructor.

This constructor initialises the Thread Object QUEUE_THREAD and sets the queue to dispatch forever via a call-back. The message

```
"Queue Initialised"
```

will be displayed once initialisation has been completed.

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

- CustomQueue.h
- CustomQueue.cpp

4.3 ErrorHandler Class Reference

Thread-safe error handler class.

```
#include <ErrorHandler.h>
```

Public Member Functions

- ErrorHandler (CustomQueue ∗errorQueue)

    Construct an ErrorHandler object.
- void error_thread ()

    Main error handler functionality.
- void setErrorFlag (int errorCode)

    Public method for passing errors to the error handler.

Public Attributes

- Thread **ERROR_THREAD_NAME**

Private Types

- enum errorSeverity {
  WARNING = 0x0 , CRITICAL = 0x1 , FATAL = 0x2 , BUFF_FULL = 0x3 ,
  ENV_ERR = 0x4 , NET_ERROR = 0x5 , CLEAR = 0x9 }

  Enumerated value for storing the error severity.
- typedef void(∗ **funcPointer_t**) (void)

  Function pointer for callbacks.

Private Member Functions

- void **clear_all** ()

  Function for clearing the Error Handler's thread flags safely.
- void alarm_override ()

  ISR to handle the user button alarm override.

Private Attributes

- DigitalOut **yellowLED** = TRAF_YEL1_PIN
- DigitalOut **redLED** = TRAF_RED1_PIN
- InterruptIn **override_button**

  Interrupt attached to the User Button. Used for alarm override.
- SevenSegmentDisplay **errorDisplay**

  Seven Segment Display used for displaying the active error code.
- Mutex **flagLock**

  Error flag clearing lock.
- CustomQueue ∗ **queue**

  Pointer to the output error queue.
- int **flag_value** = 0
- int **currentErrorSeverity** = 0
- int **alarm_status** =0
- int **prevAlarmFlag** =0

4.3.1 Detailed Description

Thread-safe error handler class.

To send an error to the error handler, use the format:
```
EH.setErrorFlag(T_UPPER);
```

4.3.2 Member Enumeration Documentation

4.3.2.1 errorSeverity

`enum ErrorHandler::errorSeverity [private]`

Enumerated value for storing the error severity.

Corresponds to the first four bits of each error code. All severities will display the error code on the seven segment display.

Enumerator

WARNING	Lights a yellow LED.
CRITICAL	Lights a red LED, sounds a 30 second alarm, and performs a reset of the board.
FATAL	Immediate reset of the board.
BUFF_FULL	Legacy severity for testing buffer integration.
ENV_ERR	sounds a buzzer for three seconds.
NET_ERROR	networking error causes the boad to reset.
CLEAR	Clears all error outputs.

4.3.3 Constructor & Destructor Documentation

4.3.3.1 ErrorHandler()

```
ErrorHandler::ErrorHandler (
              CustomQueue * errorQueue )
```

Construct an ErrorHandler object.

This constructor must be given a pointer to an event queue in order to properly output error codes over serial. This class will not function without one, and no alternative constructor is provided.

```
EventQueue* queue = new EventQueue();
ErrorHandler EH(queue);
```

To send an error to the error handler, use the format:

```
EH.setErrorFlag(T_UPPER);
```

Parameters

errorQueue	- Pointer to an eventQueue object to be used for printing the error messages to a serial output device

4.3.4 Member Function Documentation

4.3.4.1 alarm_override()

```
void ErrorHandler::alarm_override ( )  [private]
```

ISR to handle the user button alarm override.

This ISR disables the buzzer prematurely, before the error handler normally would.

4.3.4.2 error_thread()

```
void ErrorHandler::error_thread (
             void  )
```

Main error handler functionality.

Waits for a flag to be set using setErrorFlag, before responding with the appropriate outputs based on the error severity. An example call can be seen below.

4.3.4.3 setErrorFlag()

```
void ErrorHandler::setErrorFlag (
             int errorCode )
```

Public method for passing errors to the error handler.

Parameters

errorCode	Error code value, should use one of the specified macros in ErrorHandler.h

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

- ErrorHandler.h
- ErrorHandler.cpp

4.4 HTTP_server Class Reference

Public Member Functions

- HTTP_server (CustomQueue ∗printQueue, ErrorHandler ∗errorHandler, sampler ∗webSampler)
  Construct the HTTP Server obejct.
- void HTTP_server_thread (void)
  HTTP Server thread.

Public Attributes

- Thread **HTTP_thread**

Private Attributes

- EthernetInterface **network**
  Create an ethernet interface to communicate with the network.
- TCPSocket **socket**
  Create a TCP socket to wait for an incoming connection.
- TCPSocket ∗ **client_socket**
  Pointer to the socket.
- SocketAddress **address**
  IP of the connection to the socket.
- sampler ∗ **webDataSampler**
- samples **sampledData**

4.4.1 Constructor & Destructor Documentation

4.4.1.1 HTTP_server()

```
HTTP_server::HTTP_server (
            CustomQueue * printQueue,
            ErrorHandler * errorHandler,
            sampler * webSampler )
```

Construct the HTTP Server obejct.

This constructor must be given a pointer to an event queue and the Error Handler. This allows it report errors and output the current state of the connection. This class will not function without this two arguments and no other constructor is provided.

```
CustomQueue* printQueue = new CustomQueue();
ErrorHandler EH(&printQueue);
HTTP_Server (&printQueue, &EH);
```

Parameters

printQueue	- pointer to a print queue object to allow the to use a print to output information
errorHanlder	- point to an error handler object to allow the NTP to raise errors Connects to NTP server to get current date and time and stores this locally

4.4.2 Member Function Documentation

4.4.2.1 HTTP_server_thread()

```
void HTTP_server::HTTP_server_thread (
            void )
```

HTTP Server thread.

Waits for a request to be sent to the HTTP interface. Accepts the Client Socket. Constructs the webpage HTML and returns this along with a 200 OK Message. Once completed it closes the Client Socket and waits for another request.

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

- HTTP_Server.h
- HTTP_Server.cpp

4.5 LEDMatrix Class Reference

Thread-safe LED Matrix class.

```
#include <LEDMatrix.h>
```

Public Member Functions

- LEDMatrix ()

  Default constructor.
- void clear ()

  Method to clear the display.
- void writeMatrix (int RHC, int LHC, int ROW)

  Writes the speicifc bytes to the LED Matrix LED.
- void plot ()

  Method to plot the samples as a bar graph.
- void test ()

  Test method.
- void update (int updatedSamples[ ])

  Internal buffer update function.
- void matrixThread ()

  Callback method for the the MatrixThread thread object.

Public Attributes

- Thread **MatrixThread**

  Thread to run the matrix update function.

Private Attributes

- SPI **matrix_spi**

  MOSI, MISO, SCLK.
- DigitalOut **matrix_spi_cs**

  Chip Select ACTIVE LOW.
- DigitalOut **matrix_spi_oe**

  Output Enable ACTIVE LOW.
- int **samples** [8]

  Internal buffer for holding sample values to be plotted.

4.5.1 Detailed Description

Thread-safe LED Matrix class.

This class allows the use of the MSB's LED Matrix display in a thread-safe manner. It includes a method for plotting a bar display of eight values, as well as test functions for verifiying correct operation.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 LEDMatrix()

```
LEDMatrix::LEDMatrix ( )
```

Default constructor.

This initialises the matrix_spi with the correct pins, resets the internal buffer, and then starts the MatrixThread thread object with the matrixThread method as a callback.

4.5.3 Member Function Documentation

4.5.3.1 clear()

```
void LEDMatrix::clear ( )
```

Method to clear the display.

This method transmits three sets of 0x00 to the matrix display, clearing it.

4.5.3.2 matrixThread()

```
void LEDMatrix::matrixThread ( )
```

Callback method for the the MatrixThread thread object.

This method is passed to the thread as a callback, and simply runs the 'plot' method to mimic a hold functionality.

4.5.3.3 plot()

```
void LEDMatrix::plot ( )
```

Method to plot the samples as a bar graph.

This method will automatically plot the samples in the internal buffer as a graph. Use the update function to add new samples to be plotted.

4.5.3.4 test()

```
void LEDMatrix::test ( )
```

Test method.

Tests the functionality of the display by lighting up the rows in order.

4.5.3.5 update()

```
void LEDMatrix::update (
            int updatedSamples[] )
```

Internal buffer update function.

This method allows the internal buffer, used with the Plot method. By calling this in another class, values can be passed in.

Parameters

updatedSamples	eight value buffer of values 0-16, controlling how many lights on each row should be lit.

4.5.3.6 writeMatrix()

```
void LEDMatrix::writeMatrix (
            int RHC,
            int LHC,
            int ROW )
```

Writes the speicifc bytes to the LED Matrix LED.

Parameters

RHC	value between 0 - 255 to control the leds on RHC
LHC	value between 0 - 255 to control the leds on LHC
value	between 0 - 7 to control the leds on ROWS Returns: Null

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

- LEDMatrix.h
- LEDMatrix.cpp

4.6 limits Struct Reference

Structure for holding the alarm thresholds.

```
#include <sampling.h>
```

Public Member Functions

- limits ()

  Default limit constructor.
- limits (float limits[6])

  Constructor with default override.
- void bind (float limits[6])

  Method to set all limits at once using an array.
- void bind_upper (float limits[3])

  Method to set all upper limits at once using an array.
- void bind_lower (float limits[3])

  Method to set all lower limits at once using an array.

Public Attributes

- float **t_upper**
- float **t_lower**
- float **p_upper**
- float **p_lower**
- float **l_upper**
- float **l_lower**

4.6.1 Detailed Description

Structure for holding the alarm thresholds.

This structure stores six floats, an upper and lower threshold for each sensor type, and includes both a default and custom constructor, as well as a method for changing the thresholds in-situ.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 limits() [1/2]

```
limits::limits ( )  [inline]
```

Default limit constructor.

This creates an instance of limits with the default values.

4.6.2.2 limits() [2/2]

```
limits::limits (
            float limits[6] )  [inline]
```

Constructor with default override.

This constructor creates an instance of the struct limits, with the threshold values set equal to the input array.

Parameters

limits	Array of limits, upper and lower, temperature, pressure, light. Temperature in Celsius. Pressure in millibars. Light value as a 0->1 float value.

4.6.3 Member Function Documentation

4.6.3.1 bind()

```
void limits::bind (
            float limits[6] )  [inline]
```

Method to set all limits at once using an array.

This method allows an array to be used to override all limits at once, with a single call. Limits cannot be selectively over-written, the array must have a length of six.

Parameters

limits	Array of limits, upper and lower, temperature, pressure, light. Temperature in Celsius. Pressure in millibars. Light value as a 0->1 float value.

4.6.3.2 bind_lower()

```
void limits::bind_lower (
            float limits[3] )  [inline]
```

Method to set all lower limits at once using an array.

This method allows an array to be used to override lower limits at once, with a single call. Limits cannot be selectively over-written, the array must have a length of three. This was desinged for use with the remote command inputs.

Parameters

limits	Array of lowewr limits, temperature, pressure, light. Temperature in Celsius. Pressure in millibars. Light value as a 0->1 float value.

4.6.3.3 bind_upper()

```
void limits::bind_upper (
            float limits[3] )  [inline]
```

Method to set all upper limits at once using an array.

This method allows an array to be used to override upper limits at once, with a single call. Limits cannot be selectively over-written, the array must have a length of three. This was desinged for use with the remote command inputs.

Parameters

limits	Array of upper limits, temperature, pressure, light. Temperature in Celsius. Pressure in millibars. Light value as a 0->1 float value.

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

- sampling.h

4.7 bufferClass::liveData Struct Reference

Structure for holding sample data, sampled every 10s, and time and data that the data was obtained.

```
#include <buffer.h>
```

Public Attributes

- float **LDR**
- float **temp**
- float **pressure**
- char ∗ **realTime**

4.7.1 Detailed Description

Structure for holding sample data, sampled every 10s, and time and data that the data was obtained.

Parameters

LDR	float value of light level
temp	float value of temperate
pressue	float value of pressure
realTime	char - time and data

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

- buffer.h

4.8 NTPConnection Class Reference

NTP Connection class.

```
#include <NTPConnection.h>
```

Public Member Functions

- NTPConnection (CustomQueue ∗printQueue, ErrorHandler ∗errorHandler)

  Construct the Network Connecion obejct.
- time_t **getTime** ()

  Get time the current local time.

Public Attributes

- time_t **timestamp**

  variable of time_t to save the value obtained by the NTP Server

Private Attributes

- NetworkInterface ∗ **NTPInterface**

  pointer to the Network Interface
- CustomQueue ∗ **printQueue**

  pointer to the output error queue
- ErrorHandler ∗ **errorHandler**

  pointer to the error handler

4.8.1 Detailed Description

NTP Connection class.

While this class is not thread safe. The way it is called ensures that is will not interrupt any other processes. This class is primarily used during the Monitor's initialisation stage, and so will not interfere with any of the real-time operations.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 NTPConnection()

```
NTPConnection::NTPConnection (
            CustomQueue * printQueue,
            ErrorHandler * errorHandler )
```

Construct the Network Connecion obejct.

This constructor must be given a pointer to an event queue and the Error Handler. This allows it report errors and output the current state of the connection. This class will not function without these two arguments and no other constructor is provided.

```
CustomQueue* printQueue = new CustomQueue();
ErrorHandler EH(&printQueue);
NTPConnection NTP(&printQueue, &EH);
```

Parameters

printQueue	- pointer to a print queue object to allow the to use a print to output information
errorHanlder	- point to an error handler object to allow the NTP to raise errors Connects to NTP server to get current date and time and stores this locally

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

- NTPConnection.h
- NTPConnection.cpp

4.9 quantised_samples Struct Reference

Structure for holding quantised samples.

```
#include <sampling.h>
```

Public Attributes

- int **qsamples** [8]

4.9.1 Detailed Description

Structure for holding quantised samples.

This structure is primarily used for interfacing with the matrix display, and holds eight integer values.

Parameters

qsamples	eight index integer array for holding quantised samples. These values should be 0-16.

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

- sampling.h

4.10 sampler Class Reference

Public Member Functions

- sampler (ErrorHandler ∗OutputHandler)

  Default constructor.
- sampler (ErrorHandler ∗OutputHandler, float limits[6])

  Alternative constructor that overrides default limits.
- void sensorChange (char in)

  Method to change the sensor to a specific selection.
- void displayLimits ()

  Debug method to display the currently set alarm thresholds.
- sensor_type **get_current_sensor** ()

  Get method for returning sensor type.
- ∼**sampler** ()

  Default deconstructor for the sampler object.

Public Attributes

- quantised_samples **matrix_input**

  Holds quantised values to be passed to the matrix display.
- samples **internal_buffer** [8]

  Internal buffer to hold samples for the matrix.
- samples **sampleData**

  Single sample data to hold the latest sample.
- limits **threshold**

  limits object to store the current alarm thresholds

Private Types

- enum **sensor_type** { **TEMP** , **PRESSURE** , **LIGHT** }
- typedef void(∗ **funcPointer_t**) (void)

Private Member Functions

- void sample ()

  Main sampling method.
- void sampleflag ()

  ISR to raise the sample flag.
- void sensorflag ()

  ISR to flip the sensor flag.
- void quantise (sensor_type selectedSensor)

  Quantise the internal buffer to sixteen levels and send to the matrix.
- void **matrixInterface** ()

  Matrix interface method.This methods handles interfacing with the matrix, including quantising samples and storing them in an internal buffer, as well as making the appropriate calls to the matrix class to update it's internal buffer too.
- void thresholdCheck ()

  Method to check the incoming sample against the alarm threshold.

Private Attributes

- InterruptIn **BT_A**

  Button for controlling the matrix output sensor.
- sensor_type **currentSensor** = LIGHT

  Current sensor output, default is 'LIGHT'.
- Mutex **sampleLock**

  Mutex Lock to ensure thread safety on sample values.
- Ticker **sampleTick**

  Ticker interrupt to trigger sampling at once per second.
- Thread **sampleThread**
- Thread **matrixThread**

  Thread declarations.
- LEDMatrix **matrix**

  LED Matrix display for outputting sample bar graphs.
- uop_msb::EnvSensor **sensor**
- AnalogIn **LDR**
- ErrorHandler ∗ **EH**

  Error Handler.
- int **prevAlarmFlag** = 1

4.10.1 Constructor & Destructor Documentation

4.10.1.1 sampler() [1/2]

```
sampler::sampler (
            ErrorHandler * OutputHandler )
```

Default constructor.

This constructor will instantiate an instance of the sample module with the default alarm thresholds.

Parameters

OutputHandler	Pointer towards an error handler instance, to respond to errors. For more details see the ErrorHandler.h file.

4.10.1.2 sampler() [2/2]

```
sampler::sampler (
            ErrorHandler * OutputHandler,
            float limits[6] )
```

Alternative constructor that overrides default limits.

This constructor can be used if limit values other than the defaults are required.

Parameters

limits	Array of limits, upper and lower, temperature, pressure, light. Temperature in Celsius. Pressure in millibars. Light value as a 0->1 float value.
OutputHandler	Pointer towards an error handler instance, to respond to errors. For more details see the ErrorHandler.h file.

4.10.2 Member Function Documentation

4.10.2.1 displayLimits()

```
void sampler::displayLimits ( )
```

Debug method to display the currently set alarm thresholds.

This method is intended to be used for debugging purposes primarily.

Note

This method is not thread safe! It makes use of slow 'printf' calls and should not be called in any timing dependant contexts.

4.10.2.2 quantise()

```
void sampler::quantise (
             sensor_type selectedSensor )  [private]
```

Quantise the internal buffer to sixteen levels and send to the matrix.

This method quantises the internal buffer to sixteen levels, before sending the selected measurement (Temp, Pressure, Light), to the matrix display to be displayed graphically.

Parameters

selectedSensor	Stores what measurement is to be output onto the matrix display. The method only quantises the desired measurement values.

4.10.2.3 sample()

```
void sampler::sample ( )  [private]
```

Main sampling method.

This method contains the majority of the sampler's methodality. After being awoken by the ticker, it reads the sensor values in a thread safe manner, checking the values against the set thresholds. If any of these thresholds are broken, the appropriate error flag is raised. The sample data is quantised and then sent to the matrix.

4.10.2.4 sampleflag()

```
void sampler::sampleflag ( )  [private]
```

ISR to raise the sample flag.

This interrupt service routine is triggered by a ticker attached in the class constructor, and fires every second waking the sampler up.

4.10.2.5 sensorChange()

```
void sampler::sensorChange (
             char in )
```

Method to change the sensor to a specific selection.

Called by the serial input, this method raises flags in the matrix thread in order to signal a sensor change is desired. If the input is unrecognised, it instead toggles the sensor to the next selection, the same as when Button A is pressed.

Parameters

in	char input, either T/P/L for temperature, pressure, or light respectively.

4.10.2.6 sensorflag()

```
void sampler::sensorflag ( )  [private]
```

ISR to flip the sensor flag.

The sensor flag indicates that the desired output sensor has been changed.

4.10.2.7 thresholdCheck()

```
void sampler::thresholdCheck ( )  [private]
```

Method to check the incoming sample against the alarm threshold.

This method compares the latest sample to the tresholds held in the thresholds struct. Should any of these thresholds be crossed, the appropriate error is sent to the error handler.

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

- sampling.h
- sampling.cpp

4.11 samples Struct Reference

Structure for holding the sample data.

```
#include <sampling.h>
```

Public Attributes

- float **temp**
- float **pressure**
- float **LDR**

4.11.1 Detailed Description

Structure for holding the sample data.

The samples struct holds three values, one from each sensor, taken at 10 second intervals.

Parameters

temp	Float value to hold a temperature reading.
pressure	Float value to hold a pressure reading.
LDR	Float value to hold a light level reading.

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

- sampling.h

4.12 SerialIn Class Reference

Thread-safe Serial Input class.

```
#include <SerialIn.h>
```

Public Member Functions

- SerialIn (CustomQueue ∗printQueue, sampler ∗serialSamples, bufferClass ∗serialBuffer)

    Construct the Serial In object.
- void SerialListener ()

    Main Functionality of the Serial In class.
- void **SerialInstructions** ()

    Prints onto the serial terminal the methods that can be called from the serial terminal.
- void **SerialTest** ()

    Test code to check the serial inputs and outputs are working correctly.
- void **Help** ()

    Prints onto the serial terminal the instructions required to use the various functions.

Private Attributes

- CustomQueue ∗ **pQ**

    Pointer to the custom queue for status outputs.
- sampler ∗ **serialSampler**

    Pointer to the sampler to change threshold limits.
- bufferClass ∗ **serialBuff**

    Pointer to the buffer to flush, get samples, and number of samples.
- Thread **SerialWatcher**

    Thread for the main Serial Inputs.

4.12.1 Detailed Description

Thread-safe Serial Input class.

Instructions for use outputted to the serial terminal on start-up

4.12.2 Constructor & Destructor Documentation

4.12.2.1 SerialIn()

```
SerialIn::SerialIn (
             CustomQueue * printQueue,
             sampler * serialSamples,
             bufferClass * serialBuffer )
```

Construct the Serial In object.

This constructor must be given pointers to the custom print queue, sampler, and buffer. This allows the class to communicate with the sampler, buffer, and print queue. This class will not function without these three arguments and no other constructor is provided.

```
SerialIn serialInput(&printQueue, &SampleModule, &mainBuffer);
```

Parameters

pintQueue	- pointer to a print queue object to allow the use of a print to output information
serialSamples	- pointer to the sampler object to allow changes to the input thresholds
serialBuffer	- pointer to the buffer object to allow communication between the buffer and serial input Instantiates the Serial In class, outputs the serial instructions, and starts the SerialWatcher thread.

4.12.3 Member Function Documentation

4.12.3.1 SerialListener()

```
void SerialIn::SerialListener ( )
```

Main Functionality of the Serial In class.

Takes the Serial Data and deicdes what to do from the basic set of instructions. The appropriate action is then applied. This function is a blocking function because of the scanf. As it is in its own thread, it does inhibit the other operations.

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

- SerialIn.h
- SerialIn.cpp

4.13 SevenSegmentDisplay Class Reference

Public Member Functions

- SevenSegmentDisplay ()

  Default constructor.
- void clear ()

  Method to clear the displays.
- void clear (int number)

  Alternative definition for the clear method.
- void setDigit (int digit, int number)

  Method to set a single digit of the display.
- void setNumber (int number)

  Method to set both digits of the display.
- void test ()

  Test method.
- void operator= (int number)

  Operator overload for the set number funtion.

Private Attributes

- BusOut **digits**

  digits for 7 segment display
- BusOut **display**

  selects which 7 segment to display from
- DigitalOut **output_enable**

  Output enabled ACTIVE LOW.

4.13.1 Constructor & Destructor Documentation

4.13.1.1 SevenSegmentDisplay()

```
SevenSegmentDisplay::SevenSegmentDisplay ( )
```

Default constructor.

This constructor initialises a Seven Segment Display object with the correct pin assignments for the MSB V2 board.

4.13.2 Member Function Documentation

4.13.2.1 clear() [1/2]

```
void SevenSegmentDisplay::clear ( )
```

Method to clear the displays.

Calling the method without any arguments will clear both digits, completely clearing the display.

4.13.2.2 clear() [2/2]

```
void SevenSegmentDisplay::clear (
             int number )
```

Alternative definition for the clear method.

When passing an argument to the method it will clear one specific digit of the display.

Parameters

number	integer specifying which digit to clear.

4.13.2.3 operator=()

```
void SevenSegmentDisplay::operator= (
             int number )
```

Operator overload for the set number funtion.

This operator overload allows the display to be set using the following syntax:

```
LEDMatrix = number;
```

This streamlines the code within considerably, and makes the code much more readable.

4.13.2.4 setDigit()

```
void SevenSegmentDisplay::setDigit (
             int digit,
             int number )
```

Method to set a single digit of the display.

This method takes two arguments, and sets the selected digit to the value of the argument passed.

Parameters

digit	Which digit of the display to change.
number	The number that the selected digit should be set to.

4.13.2.5 setNumber()

```
void SevenSegmentDisplay::setNumber (
            int number )
```

Method to set both digits of the display.

This method sets the display to be equal to the integer value passed to it.

4.13.2.6 test()

```
void SevenSegmentDisplay::test ( )
```

Test method.

This method tests the display by incrementing through different values and displaying them on the display.

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

- SevenSegmentDisplay.h
- SevenSegmentDisplay.cpp

Chapter 5

File Documentation

5.1 buffer.h File Reference

buffer and sd card header file Author - Noah Harvey

```
#include "CustomQueue.h"
#include "sampling.h"
#include <FATFileSystem.h>
#include <InterruptIn.h>
#include <Mutex.h>
#include <SDBlockDevice.h>
#include <Semaphore.h>
#include <ctime>
#include <mbed.h>
#include <uop_msb.h>
```

Classes

- class bufferClass

  buffer class with sd card functionality and dedicated threads for writig to the buffer flushing the sd card
- struct bufferClass::liveData

  Structure for holding sample data, sampled every 10s, and time and data that the data was obtained.

Macros

- #define buffer_size 100

  macro so define the size of the buffer
- #define **SDDetect** PF_4

  define sd card detecting pin

5.1.1 Detailed Description

buffer and sd card header file Author - Noah Harvey

5.1.2 Macro Definition Documentation

5.1.2.1 buffer_size

```
#define buffer_size 100
```

macro so define the size of the buffer

define buffer size

5.2 buffer.h

Go to the documentation of this file.

```
#ifndef __FIFO_BUFFER__
#define __FIFO_BUFFER__

#include "CustomQueue.h"
#include "sampling.h"
#include <FATFileSystem.h>
#include <InterruptIn.h>
#include <Mutex.h>
#include <SDBlockDevice.h>
#include <Semaphore.h>
#include <ctime>
#include <mbed.h>
#include <uop_msb.h>

using namespace std;

#define buffer_size 100
// define the pin for the sd card
#define SDDetect PF_4

class bufferClass {

private:
  typedef void (*funcPointer_t)(void);
  // mutex locks, tickers, timers,
  Mutex bufferLock;
  Mutex timeLock;
  Ticker bufferWriteTick;
  Ticker bufferFlushTick;
  Timer t;
  time_t timestamp;
  InterruptIn SDDetector;

  sampler *bSamp;
  ErrorHandler *BEH;
  CustomQueue *PQ;

  void writeFlag();
  Thread writeThread;

  void flushFlag();
  Thread flushThread;

  void writeBufferAuto();
  unsigned int newIDX =
      buffer_size - 1;
  unsigned int oldIDX = buffer_size - 1;

  void whenToFlush();
  float flushTiming;
  float hourPassed = 59 * 60;

public:
  bufferClass(sampler *buffersampler, ErrorHandler *bufferEH,
              CustomQueue *bufferPQ);

  struct liveData {
    float LDR;
    float temp;
    float pressure;
    char *realTime;
  };

  samples sampleData;                // sampled values, every 10s
  liveData buffer[buffer_size];
  liveData dataRecord;
  liveData printRecord[buffer_size];

  void fetchLatestRecord();

  void flushBuffer();
  int runFlush = 1;

  void bufferCount();

  void printBufferContents();
  int runPrint = 1;

  void initSD();
  bool SDMount;

  void SDRemoved();

  void flashGreen(); // for showing that the sd card is being flushed

  // destructor
  ~bufferClass();
};

#endif
```

5.3 CustomQueue.h File Reference

CustomQueue class header file.

```
#include "mbed.h"
#include "uop_msb.h"
#include "string.h"
#include <string>
```

Classes

- class CustomQueue

  Event Queue wrapper class.

5.3.1 Detailed Description

CustomQueue class header file.

This class contains an event queue with a wrapper. This wrapper ensures that the queue's thread is initialised and that the queue is dispatched before being used by any of the other classes. Several of the classes have status prints within their constructors, so a pre-main initialisation is critical to capture all status messages.

Author - Jack Pendlebury

5.4 CustomQueue.h

Go to the documentation of this file.

```
#ifndef __CUSTOM_QUEUE__
#define __CUSTOM_QUEUE__

#include "mbed.h"
#include "uop_msb.h"
#include "string.h"
#include <string>


class CustomQueue{
    private:

    public:
    CustomQueue();
    EventQueue custom;
    Thread QUEUE_THREAD;
};

#endif
```

5.5 ErrorHandler.h File Reference

Error Handler class header file.

```
#include "DigitalOut.h"
#include "SevenSegmentDisplay.h"
#include "Thread.h"
#include "mbed.h"
#include "Mutex.h"
#include <uop_msb.h>
#include "CustomQueue.h"
```

Classes

- class ErrorHandler

  Thread-safe error handler class.

Macros

- #define BUZZER_ENABLE 0

  Macro to control the buzzer usage.
- #define **BUFFER_FULL** 0x110

  FULL BUFFER - SEVERITY CRITICAL.
- #define **BUFFER_LOCK_TIMEOUT** 0x111

  BUFFER LOCK TIMEOUT - CRITICAL.
- #define **TIMER_LOCK_TIMEOUT** 0x112

  TIMER LOCK TIMEOUT - CRITICAL.
- #define **EMPTY_FLUSH** 0x013

  FLUSH WHEN EMPTY - WARNING.
- #define **BUFFER_FLUSH_TIMEOUT** 0x114

  FAILED FLUSH - CRITICAL.
- #define **MOUNT_ERROR** 0x120

NO SD CARD MOUNTED - CRITICAL.

- #define **NO_SD_FILE** 0x121
  NO FILE TO WRITE TO - CRITICAL.
- #define **UNMOUNTED_FLUSH** 0x122
  SD CARD SLOT EMPTY, CANNOT FLUSH - CRITICAL.
- #define **T_LOWER** 0x430
  Lower temperature threshold exceeded.
- #define **T_UPPER** 0x431
  Upper temperature threshold exceeded.
- #define **P_LOWER** 0x433
  Lower pressure threshold exceeded.
- #define **P_UPPER** 0x434
  Upper pressure threshold exceeded.
- #define **L_LOWER** 0x435
  Lower light threshold exceeded.
- #define **L_UPPER** 0x436
  Upper light threshold exceeded.
- #define **NO_NETWORK_INTERFACE** 0x540
  NO NETWORK INTERFACE CONNECTED/FOUND.
- #define **CONNECTION_ERROR** 0x541
  COULD NOT CONNECT TO SERVER.
- #define **TIME_ERROR** 0x542
  FAILED TO GET TIME FROM NTP SERVER.
- #define **NO_IP_ADDRESS** 0x543
  COULD NOT GET IP ADDRESS.
- #define **LISTENER_ERROR** 0x544
  LISTENER ERROR.
- #define **FLAG_CLEAR_ERROR** 0x299
  Flag clear error, immediate reset.
- #define **ALL_CLEAR** 0x999
  All clear from modules. Clears current error code.

5.5.1 Detailed Description

Error Handler class header file.

Author - Jack Pendlebury

5.5.2 Macro Definition Documentation

5.5.2.1 BUZZER_ENABLE

```
#define BUZZER_ENABLE 0
```

Macro to control the buzzer usuage.

set to 0 when using networking functionality to avoid a hardware fault. Pre-processor directives will replace buzzer calls with queued print dispatches to the serial output.

5.6 ErrorHandler.h

Go to the documentation of this file.

```

#ifndef H_ERROR_HANDLER
#define H_ERROR_HANDLER
#include "DigitalOut.h"
#include "SevenSegmentDisplay.h"
#include "Thread.h"
#include "mbed.h"
#include "Mutex.h"
#include <uop_msb.h>
#include "CustomQueue.h"
using namespace uop_msb;
#define BUZZER_ENABLE 0


// ERROR CODE FORMAT
// TWELVE BITS
//   11->8   Severity Code
//   7->4    Module Code
//   3->0    Error ID
// SEVERITY CODES
//  0x0 - WARNING
//  0x1 - CRITICAL
//  0x2 - FATAL
//  0x4 - ENV_ERROR
//  0x5 - NET_ERROR
//  0x9 - ALL CLEAR

//error codes - 10s

#define BUFFER_FULL              0x110
#define BUFFER_LOCK_TIMEOUT      0x111
#define TIMER_LOCK_TIMEOUT       0x112
#define EMPTY_FLUSH              0x013
#define BUFFER_FLUSH_TIMEOUT     0x114

//sd card errors - 20s - All Critical
#define MOUNT_ERROR              0x120
#define NO_SD_FILE               0x121
#define UNMOUNTED_FLUSH          0x122

//ENV_ERRORS - 30s
#define T_LOWER 0x430
#define T_UPPER 0x431
#define P_LOWER 0x433
#define P_UPPER 0x434
#define L_LOWER 0x435
#define L_UPPER 0x436

//networking errors - 40s
#define NO_NETWORK_INTERFACE     0x540
#define CONNECTION_ERROR         0x541
#define TIME_ERROR               0x542
#define NO_IP_ADDRESS            0x543
#define LISTENER_ERROR           0x544

//error handler errors
#define FLAG_CLEAR_ERROR 0x299

#define ALL_CLEAR 0x999

class ErrorHandler {
    private:
    enum errorSeverity{
    WARNING = 0x0,
    CRITICAL = 0x1,
    FATAL = 0x2,
    BUFF_FULL = 0x3,
    ENV_ERR = 0x4,
    NET_ERROR = 0x5,
    CLEAR = 0x9
    };
    typedef void(*funcPointer_t)(void);
    DigitalOut yellowLED = TRAF_YEL1_PIN;
    DigitalOut redLED = TRAF_RED1_PIN;
     InterruptIn override_button;
     #if BUZZER_ENABLE == 1
     Buzzer buzz;
     char note = 'C';
     #endif

     SevenSegmentDisplay errorDisplay;

    Mutex flagLock;
    CustomQueue* queue;

    int flag_value = 0;
    int currentErrorSeverity = 0;
    int alarm_status=0;
    int prevAlarmFlag=0;
    void clear_all();

    void alarm_override();

    public:

    ErrorHandler(CustomQueue* errorQueue);
    ~ErrorHandler();
    void error_thread();

    void setErrorFlag(int errorCode);


    Thread ERROR_THREAD_NAME;
};

#endif
```

5.7 hardware.hpp

```
#ifndef H_HARDWARE
#define H_HARDWARE

#define TRAF_GRN1_PIN PC_6
#define TRAF_YEL1_PIN PC_3
#define TRAF_RED1_PIN PC_2

//OPEN DRAIN
#define TRAF_GRN2_PIN PC_9
#define TRAF_YEL2_PIN PC_8
#define TRAF_RED2_PIN PC_7

#endif
```

5.8 HTTP_Server.h File Reference

HTTP Server class header file.

```
#include "EthernetInterface.h"
#include "TCPSocket.h"
#include <string>
#include "Thread.h"
#include "mbed.h"
#include "sampling.h"
#include "uop_msb.h"
#include "CustomQueue.h"
#include "ErrorHandler.h"
```

Classes

- class HTTP_server

Macros

- #define **HTTP_STATUS_LINE** "HTTP/1.0 200 OK"

    shows that the responce to the broswer was successful - must return a payload
- #define **HTTP_HEADER_FIELDS** "Content-Type: text/html; charset=utf-8"

    Provides required imformation about te responce.
- #define HTTP_MESSAGE_BODY

    Body of the HTML message the will make up the webpage.
- #define HTTP_TEMPLATE

    Structure of the HTTP responce being sent to the broswer.

5.8.1 Detailed Description

HTTP Server class header file.

Author - Luke Waller

5.8.2 Macro Definition Documentation

5.8.2.1 HTTP_MESSAGE_BODY

```
#define HTTP_MESSAGE_BODY
```

Value:

```
""                                     \
"<html>" "\r\n" \
"<h1 style=\"text-align: center;\">ELEC351 Coursework webpage - Group D</h1>" "\r\n" \
"   <p style=\"text-align: center;\">Date and Time: {{1}}</p>" "\r\n" \
"<h2 style=\"text-align: center;\">Peripherals</h2>"   "\r\n" \
"   <table style=\"text-align: center; table-layout: fixed; width: 100%;\">""\r\n" \
"       <tbody>" "\r\n" \
"       <tr>"                                                              "\r\n" \
"           <th style=\"border: 1px solid black;\">POT</th>"               "\r\n" \
"           <th style=\"border: 1px solid black;\">LDR</th>"               "\r\n" \
"           <th style=\"border: 1px solid black;\">PRESSURE</th>"          "\r\n" \
"           <th style=\"border: 1px solid black;\">TEMPERATURE</th>"       "\r\n" \
"       </tr>"                                                             "\r\n" \
"       <tr>"                                                              "\r\n" \
"           <td style=\"border: 1px solid black;\">{{2}}</td>"             "\r\n" \
"           <td style=\"border: 1px solid black;\">{{3}}</td>"             "\r\n" \
"           <td style=\"border: 1px solid black;\">{{4}}</td>"             "\r\n" \
"           <td style=\"border: 1px solid black;\">{{5}}</td>"             "\r\n" \
"       </tr>"                                                             "\r\n" \
"       </tbody>"                                                          "\r\n" \
"   </table>"                                                              "\r\n" \
"</html>"                                                                  "\r\n"
```

Body of the HTML message the will make up the webpage.

5.8.2.2 HTTP_TEMPLATE

```
#define HTTP_TEMPLATE
```

Value:

```
                      HTTP_STATUS_LINE "\r\n"   \
                      HTTP_HEADER_FIELDS "\r\n" \
                      "\r\n"                    \
                      HTTP_MESSAGE_BODY "\r\n"
```

Structure of the HTTP responce being sent to the broswer.

5.9 HTTP_Server.h

Go to the documentation of this file.

```

#ifndef __HTTP_SERVER__
#define __HTTP_SERVER__

#include "EthernetInterface.h"
#include "TCPSocket.h"
#include <string>
#include "Thread.h"
#include "mbed.h"
#include "sampling.h"
#include "uop_msb.h"
#include "CustomQueue.h"
#include "ErrorHandler.h"
using namespace uop_msb;


#define HTTP_STATUS_LINE "HTTP/1.0 200 OK"
#define HTTP_HEADER_FIELDS "Content-Type: text/html; charset=utf-8"
#define HTTP_MESSAGE_BODY ""                                     \
"<html>" "\r\n" \
"<h1 style=\"text-align: center;\">ELEC351 Coursework webpage - Group D</h1>" "\r\n" \
"   <p style=\"text-align: center;\">Date and Time: {{1}}</p>" "\r\n" \
"<h2 style=\"text-align: center;\">Peripherals</h2>"   "\r\n" \
"   <table style=\"text-align: center; table-layout: fixed; width: 100%;\">""\r\n" \
"       <tbody>" "\r\n" \
"       <tr>"                                                               "\r\n" \
"           <th style=\"border: 1px solid black;\">POT</th>"                "\r\n" \
"           <th style=\"border: 1px solid black;\">LDR</th>"                "\r\n" \
"           <th style=\"border: 1px solid black;\">PRESSURE</th>"           "\r\n" \
"           <th style=\"border: 1px solid black;\">TEMPERATURE</th>"        "\r\n" \
"       </tr>"                                                              "\r\n" \
"       <tr>"                                                               "\r\n" \
"           <td style=\"border: 1px solid black;\">{{2}}</td>"              "\r\n" \
"           <td style=\"border: 1px solid black;\">{{3}}</td>"              "\r\n" \
"           <td style=\"border: 1px solid black;\">{{4}}</td>"              "\r\n" \
"           <td style=\"border: 1px solid black;\">{{5}}</td>"              "\r\n" \
"       </tr>"                                                              "\r\n" \
"       </tbody>"                                                           "\r\n" \
"   </table>"                                                               "\r\n" \
"</html>"                                                                   "\r\n"

#define HTTP_TEMPLATE HTTP_STATUS_LINE "\r\n"   \
                      HTTP_HEADER_FIELDS "\r\n" \
                      "\r\n"                    \
                      HTTP_MESSAGE_BODY "\r\n"

class HTTP_server{

    private:
    EthernetInterface network;
    TCPSocket socket;
    TCPSocket* client_socket;
    SocketAddress address;

    sampler* webDataSampler;

    samples sampledData;

    public:
    HTTP_server(CustomQueue* printQueue, ErrorHandler* errorHandler, sampler* webSampler);
    void HTTP_server_thread(void);


    Thread HTTP_thread;
};

#endif
```

5.10 LEDMatrix.h File Reference

class header file.

```
#include "DigitalOut.h"
#include "InterfaceDigitalOut.h"
```

```
#include "PinNames.h"
#include "uop_msb.h"
#include "mbed.h"
#include "SPI.h"
```

Classes

- class LEDMatrix

  Thread-safe LED Matrix class.

5.10.1 Detailed Description

class header file.

Author - Luke Waller

5.11 LEDMatrix.h

Go to the documentation of this file.

```

#ifndef __LED_MATRIX__
#define __LED_MATRIX__

#include "DigitalOut.h"
#include "InterfaceDigitalOut.h"
#include "PinNames.h"
#include "uop_msb.h"
#include "mbed.h"
#include "SPI.h"


class LEDMatrix{
    private:
    SPI matrix_spi;
    DigitalOut matrix_spi_cs;
    DigitalOut matrix_spi_oe;
    int samples[8];

    public:
    Thread MatrixThread;
    LEDMatrix();
    void clear();
    void writeMatrix(int RHC, int LHC, int ROW);
    void plot();
    void test();
    void update(int updatedSamples[]);
    void matrixThread();

};
#endif
```

5.12 main.cpp File Reference

Main file for Group D ELEC351 Coursework - Jack Pendlebury, Noah Harvey, Luke Waller.

```
#include "mbed.h"
#include "sampling.h"
#include "ErrorHandler.h"
#include "buffer.h"
#include "NTPConnection.h"
#include "CustomQueue.h"
#include "SerialIn.h"
#include "HTTP_Server.h"
```

Functions

- int **main** ()

Variables

- int **demo_mode** = 0

  Set to 1 to enable demo mode, this will execute code to induce several errors, before resetting the board with a simulated fatal error.
- CustomQueue **printQueue**
- sampler SampleModule & **EH**
- bufferClass mainBuffer & **SampleModule**

5.12.1 Detailed Description

Main file for Group D ELEC351 Coursework - Jack Pendlebury, Noah Harvey, Luke Waller.

5.13 NTPConnection.h File Reference

NTP Connection class header file.

```
#include "CustomQueue.h"
#include "ErrorHandler.h"
#include "NTPClient.h"
#include <ctime>
```

Classes

- class NTPConnection

  NTP Connection class.

Macros

- #define **HTTP_SERVER_USED** 1

  Macro to control the HTTP Server usage Set Macro to 0 if HTTP Server not being used, otherwise set to 1 Pre-processer directives will eiother try to connect the NTP server or use existing connection.

5.13.1 Detailed Description

NTP Connection class header file.

Author - Luke Waller

5.14 NTPConnection.h

Go to the documentation of this file.

```
#ifndef __NTP_CLIENT__
#define __NTP_CLIENT__

#include "CustomQueue.h"
#include "ErrorHandler.h"
#include "NTPClient.h"
#include <ctime>

#define HTTP_SERVER_USED 1

class NTPConnection{

    private:
    NetworkInterface* NTPInterface;
    CustomQueue* printQueue;
    ErrorHandler* errorHandler;


    public:
    time_t timestamp;
    NTPConnection(CustomQueue* printQueue, ErrorHandler* errorHandler);
    time_t getTime();

};

#endif
```

5.15 sampling.h File Reference

Sampler class header file.

```
#include <mbed.h>
#include <Mutex.h>
#include <uop_msb.h>
#include "LEDMatrix.h"
#include "ErrorHandler.h"
```

Classes

- struct samples

  Structure for holding the sample data.
- struct quantised_samples

  Structure for holding quantised samples.
- struct limits

  Structure for holding the alarm thresholds.
- class sampler

Macros

- #define **SAMPLE_INTERVAL** 10s

5.15.1 Detailed Description

Sampler class header file.

Author - Jack Pendlebury

5.16 sampling.h

Go to the documentation of this file.

```

#ifndef H_SAMPLING
#define H_SAMPLING
#include <mbed.h>
#include <Mutex.h>
#include <uop_msb.h>
#include "LEDMatrix.h"
#include "ErrorHandler.h"
//#include "buffer.h"

#define SAMPLE_INTERVAL 10s

struct samples{
    float temp, pressure, LDR;
};

struct quantised_samples{
    int qsamples[8];
};

struct limits{
    float t_upper,t_lower,p_upper,p_lower,l_upper,l_lower;
    limits(){
        t_upper = 35;
        t_lower = 15;
        p_upper = 1050;
        p_lower = 950;
        l_upper = 0.8;
        l_lower = 0.2;
    }
    limits(float limits[6]){
        t_upper = limits[0];
        t_lower = limits[1];
        p_upper = limits[2];
        p_lower = limits[3];
        l_upper = limits[4];
        l_lower = limits[5];
    }

    void bind(float limits[6]){
        t_upper = limits[0];
        t_lower = limits[1];
        p_upper = limits[2];
        p_lower = limits[3];
        l_upper = limits[4];
        l_lower = limits[5];
    }

    void bind_upper(float limits[3]){
        t_upper = limits[0];
        p_upper = limits[1];
        l_upper = limits[3];
    }

    void bind_lower(float limits[3]){
        t_lower = limits[0];
        p_lower = limits[1];
        l_lower = limits[3];
    }
};

class sampler {
    private:
    typedef void(*funcPointer_t)(void);
    enum sensor_type{
        TEMP,
        PRESSURE,
        LIGHT,
    };
    InterruptIn BT_A;
    sensor_type currentSensor = LIGHT;
    //float limits[6];
    Mutex sampleLock;
    Ticker sampleTick;
    Thread sampleThread,matrixThread;
    LEDMatrix matrix;
    //bufferClass sampleBuffer;
    uop_msb::EnvSensor sensor;
    AnalogIn LDR;
    ErrorHandler* EH;

    int prevAlarmFlag = 1;

    void sample();

    void sampleflag();

    void sensorflag();

    void quantise(sensor_type selectedSensor);

    void matrixInterface();

    void thresholdCheck();

    public:
    sampler(ErrorHandler* OutputHandler);

    sampler(ErrorHandler* OutputHandler, float limits[6]);

    quantised_samples matrix_input;
    samples internal_buffer[8];
    samples sampleData;
    limits threshold;


    void sensorChange(char in);

    void displayLimits();

    sensor_type get_current_sensor();

    ~sampler();


};



#endif
```

5.17 SerialIn.h File Reference

Serial Input class header file.

```
#include "CustomQueue.h"
#include "mbed.h"
#include "sampling.h"
#include "buffer.h"
```

Classes

- class SerialIn

  Thread-safe Serial Input class.

5.17.1 Detailed Description

Serial Input class header file.

Author - Luke Waller

5.18 SerialIn.h

Go to the documentation of this file.

```
#ifndef __SERIAL_IN__
#define __SERIAL_IN__

#include "CustomQueue.h"
#include "mbed.h"
#include "sampling.h"
#include "buffer.h"

class SerialIn{
    private:
    CustomQueue* pQ;
    sampler* serialSampler;
    bufferClass* serialBuff;
    Thread SerialWatcher;


    public:
    SerialIn(CustomQueue* printQueue, sampler* serialSamples, bufferClass* serialBuffer);
    void SerialListener();
    void SerialInstructions();
    void SerialTest();
    void Help();
};

#endif
```

5.19 SevenSegmentDisplay.h File Reference

Seven Segment Display class header file.

```
#include "mbed.h"
#include "mbed_retarget.h"
#include "mbed_wait_api.h"
```

Classes

- class SevenSegmentDisplay

5.19.1 Detailed Description

Seven Segment Display class header file.

Author - Luke Waller

5.20 SevenSegmentDisplay.h

Go to the documentation of this file.

```
#ifndef __Seven_Segment_Display__
#define __Seven_Segment_Display__
#include "mbed.h"
#include "mbed_retarget.h"
#include "mbed_wait_api.h"

class SevenSegmentDisplay{
    private:
```

```
    BusOut digits;
    BusOut display;
    DigitalOut output_enable;

    public:
    SevenSegmentDisplay();
    void clear();
    void clear(int number);
    void setDigit(int digit, int number);
    void setNumber(int number);
    void test();
    void operator=(int number);
};

#endif
```

Index