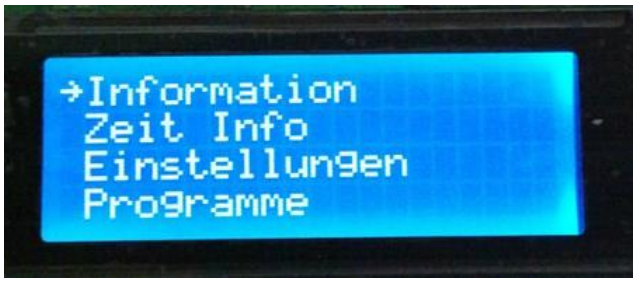
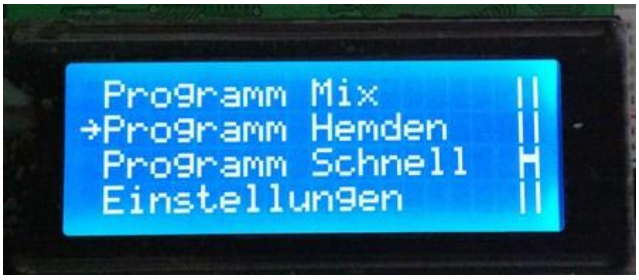
Dokumentation

LCDMenuLib




```
==========================================
================  Menu ====================
==========================================
(x) Information
( ) Time info
( ) Settings
( ) Program
```




```
==========================================
================  Menu ====================
==========================================
( ) Information
( ) Time info
( ) Settings
(x) Program
```

Doc - Version:	2016-03-28
Lib – Version:	2.1.0
Forum (german):	http://forum.arduino.cc/index.php?topic=73816.0
Download:	https://github.com/Jomelo/LCDMenuLib/releases
Issues (english / german):	https://github.com/Jomelo/LCDMenuLib/issues/

Inhaltsverzeichnis

1 Allgemein

Mit der LCDMenuLib (LCDML) kann ein Baum-Menü für ein beliebiges Display (zeilenbasiert / grafisch) oder für den serielleren Monitor erstellt werden. Es wird die Möglichkeit geboten Funktionseinheit logisch voneinander zu trennen. Dazu gibt es die Displayebene in der alle Ausgaben, die visualisiert werden, untergebracht sind und die Backendebene in der alle Abläufe, die im Hintergrund kontinuierlich ablaufen, liegen. Die Kommunikation zwischen den Ebenen erfolgt mittels globaler Variablen.

1.1 Displayebene

Auf der Displayeben wird das Menü angezeigt. Diese kann maximal 254 Menüelemente enthalten, welche sich in verschiedenen Ebenen anordnen. Sprich es gibt einen Stamm, von dem verschiedene Äste (Sub Menüs) abgehen. Ein Sub Menü existiert nur einmal im Menü und kann nicht mehreren Ästen zugeordnet werden. Dieses Modell wird auch Nested Set Model genannt. Der Stamm des Menüs wird beim Anlegen der Lib generiert, alle Äste sind konfigurierbar. Sobald in einer Ebene mehr Elemente als Zeilen im Display verwendet werden, wird ein Scroll Balken eingeblendet. Es gibt drei Trigger die den Inhalt einer Menüfunktion aktualisieren:

1. Die eingestellte Zeit der LCDML_BACKEND_menu Funktion
   UL steht für Unsigned Long und muss hier angegeben werden, da ansonsten ein Bereichsüberlauf der dahinterliegenden Variable auftritt. Die Zeit ist in den Beispielen möglichst groß gewählt um die anderen Trigger Möglichkeiten zeigen zu können
2. Durch drucken eines Buttons
3. Durch einstellen eines Triggers im Setup der Displayfunktion

Funktionseinheiten die in der Displayebene anagelegt werden bestehen **immer** aus drei Funktonen. Wird eine der folgenden Funktionen nicht angelegt führt die zu einer Fehlermeldung beim kompilieren.

1. Der Setup-Funktion, in dem Variablen initialisiert und der z.B. der oben beschriebene Trigger gesetzt werden kann. Solange die Menü Funktion aktive ist wird das Setup nicht erneut ausgeführt.

2. Der Loop-Funktion, in der die Displayinhalte aktualisiert werden
3. Der Loop-End-Funktion, die immer aufgerufen wird, wenn die Loop-Funktion verlassen wird.

1.2 Backendebene

In der Backeneben werden Funktionseinheiten die im Hintergrund kontinuierlich ablaufen sollen abgelegt.

1.3 Installation der Lib

Download: https://github.com/Jomelo/LCDMenuLib/releases

Lade die Lib unter den oben genannten Link herunter und entpacke das Archiv. Der Inhalt des Archivs kann mehrere Unterordnet beinhalte. Es wird nur der Ordner benötigt in dem sich direkt die *.c und *.h Dateien befinden. Dieser Ordner muss ins …/arduino/libraries/ Verzeichnis kopiert werden. Nach dem Einbinden der Lib muss, falls die Software gestartet war, die Software neugestartet werden. Neue Libraries werden von der Arduino Umgebung nur mit einem Neustart der Software übernommen.

Der Beispiel Code kann über „Datei -> Beispiele -> LCDMenuLib -> …“ aufgerufen werden. Es gibt dort mehrere Beispiele. Um die Funktionalität der Lib zu überprüfen sollte das oberste Beispiel geladen und getestet werden. Das Beispiel ist im folgenden Kapitel erklärt.

1.4 Ablaufdiagramm

Diagramm folgt bald

2 Wichtiges

- Die Dokumentation bezieht sich immer auf die neuste Version.
- Sobald im Programm ein DELAY(x) mit x größer als 5 verwendet wird kann ein **nicht gewolltes** verhalten auftreten. Die Backendebene und die Buttons haben während Delay aktive ist keine Funktion.

4 Beispiels: „LCDML_002_lcd_liquidcrystal“

Mit dem ersten Beispiel soll gezeigt werden wie ein einfaches Menü mit mehreren Unterebenen aufgebaut sein kann.

4.1.1 Allgemeines zum Quellcode

Alle Makros / Funktion die mit LCDML_DISP_... beginnen beziehen sich auf das Menü sowie die Funktionen die darin aufgerufen werden (Displaysystem). Alle weiteren Makros / Funktionen die mit LCDML_BACK_... anfangen beziehen sich auf Abläufe die im Hintergrundlaufen (Backendsystem)

4.1.2 Include

Im Beispiel werden zuerst die benötigten Libraries eingebunden. Die Libs die benötigten werden hängen von der verwendeten Schnittstelle ab. Z.B. werden für I2C Displays die „Wire.h“ benötigt und für das DogLCD die „DogLCD.h“.

```
// include libs
#include <LiquidCrystal.h>
#include <LCDMenuLib.h>
```

4.1.3 LCDML Konstanten

Nach den Libraries werden die Konstanten für das Menü angelegt. Beim oben genannten Beispiel sollte hier alles so belassen werden wie es ist. Später kann hier der Initscreen (Standby Screen) aktiviert werden oder die Buttonpresstime (für Buttons) eingestellt werden. Mit der Button Press Time werden die Buttons entprellt.

```
// lib config
  #define _LCDML_DISP_cfg_button_press_time          200    // button press time in ms
  #define _LCDML_DISP_cfg_scrollbar                  1      // enable a scrollbar
  #define _LCDML_DISP_cfg_cursor                     0x7E   // cursor Symbol
```

4.1.4 Initialisierung des Displays / der Schnittstelle

Nach der Lib Konfiguration werden die Einstellungen vom Display gesetzt. Es müssen die Reihen (row) und Spalten (cols) angegeben werden und und das Objekt einschließlich der

Pins mit dem das Display angeschlossen ist. Im Beispiel werden an dieser Stelle noch die Custom Chars für den Scrollbalken angelegt.

```
// settings for lcd
  #define _LCDML_DISP_cols             20
  #define _LCDML_DISP_rows             4

  // lcd object
  // liquid crystal needs (rs, e, dat4, dat5, dat6, dat7)
  LiquidCrystal lcd(4,5,6,7,8,9);

  const uint8_t scroll_bar[5][8] = {
    {B10001, B10001, B10001, B10001, B10001, B10001, B10001, B10001}, // scrollbar top
    {B11111, B11111, B10001, B10001, B10001, B10001, B10001, B10001}, // scroll state 1
    {B10001, B10001, B11111, B11111, B10001, B10001, B10001, B10001}, // scroll state 2
    {B10001, B10001, B10001, B10001, B11111, B11111, B10001, B10001}, // scroll state 3
    {B10001, B10001, B10001, B10001, B10001, B10001, B11111, B11111}  // scrollbar bottom
  };
```

4.1.5 Steuerung

Folgt bald

4.1.6 Anlegen der Display Struktur

Als nächstes wird im Beispiel das Menü angelegt. „_LCDML_DISP_cnt“ bekommt die ID des letzten Menüelementes, sprich dieser Wert wird gesetzt wenn bekannt ist wie das Menü aussieht. Wird dieser Wert zu groß gewählt entstehen Fehlermeldungen. Wenn der Wert zu klein ist wird nicht das volle Menü angezeigt.

```
#define _LCDML_DISP_cnt    11 //ID des letzten Menü elementes
```

Danach wird mit der letzten Menü Element ID viele Variablen angelegt. Da dies nicht von Hand gemacht werden muss gibt es das Makro „LCDML_DISP_init(_LCDML_DISP_cnt);. Anschließend wird die Menüstruktur aufgebaut. Für die ersten drei Elemente sieht das wie folgt aus:

```
// (id, group, prev_layer_element, new_element_num, lang_char_array, callback_function)
LCDML_DISP_add (0  , _LCDML_G1  , LCDML_root , 1  , "Information" , LCDML_FUNC_information);
LCDML_DISP_add (1  , _LCDML_G1  , LCDML_root , 2  , "Time info"   , LCDML_FUNC_timer_info);
LCDML_DISP_add (2  , _LCDML_G1  , LCDML_root , 3  , "Settings"    , LCDML_FUNC);
```

Param 1 - Id: Der erste Wert im Makro ist die ID. Diese musst fortlaufend mit Null beginnend hochgezählt werden.

Param 2 - Group: Als zweiter Wert folgt die Gruppe. Die Menüelemtne können in Gruppen eingeteilt werden. Die Gruppen können dann ausgeblendet werden z.B. um Funktion erst anzuzeigen nach dem eine Identifikation erfolgt ist oder für Funktionserweiterungen die erst später freigegeben werden sollen. In diesem Beispiel haben alle Element im Menü die Gleiche Gruppe.

Param 3 u. 4 - Prev_layer_element / new_element_num: Als drittes wird die Ebene in der sich das Element befinden soll angegeben. Mit LCDML_root ist die Hauptebene gemeint. Mit dem vierten Parameter wird die Reihenfolge in der zuvor angegeben Ebene festgelegt. Eine neue Untereben kann nur angelegt werden wenn zuvor das darüber liegende Element angelegt wurde. Z.B.

```
LCDML_DISP_add      (3 , _LCDML_G1  , LCDML_root_3      , 1  , "Change value" ,LCDML_FUNC);
```

Der Name der Ebene in der das „sub“ Element angelegt wird setzt sich aus der Hauptebene und der Position des Elementes für das Ein Sub Element erzeugt werden soll zusammen.

Param 5 - lang_char_array: Als fünften Parameter wird der Name der im Display für das Menüelement angezeigt wird angegeben. Hierbei muss beachtet werden, dass der Name zwei Zeichen kürzer sein muss als die Spalte im Display breit ist. Die Länge ist so definiert, dass vor dem Menüelement der Cursor gesetzt werden kann und hinter dem Namen der Scrollbalken Platz finden kann. Also bei einem 16x2 Display maximal 14 Zeichen und bei einem 20x4 maximal 18 Zeichen.

Param 6 - callback_function: Der letzte Parameter gibt die Callback Funktion an. Dies ist die Funktion die ausgeführt wird, sobald das Menüelement aufgerufen wird. Es macht keinen Sinn auf Menüs, die noch SubEbenen besitzen eine Funktion zu legen, diese würde nie ausgeführt werden. Für die Elemente, bei denen keine Funktion aufgerufen werden soll muss die Dummy-Funktion **„LCDML_FUNC“** eingetragen werden. Wenn ein Funktionsname im Menü eingetragen ist muss dazu auch eine Funktion existieren die aufgerufen werden kann, ansonsten entstehen Fehlermeldungen beim übersetzen

Nach dem Anlegen der Menüelemente wird mit dem Makro „LCDML_DISP_createMenu(_LCDML_DISP_cnt);“ das Menü angelegt.

4.1.7 Anlegen der Backend Struktur

Als nächstes folgt die Definition des Backends. Wie das Backendsystem im Detail funktioniert wird später erklärt. Zuerst wie schon eben beim Menü die letzte ID der Backendfunktion benötigt. Diese ist erst bekannt nachdem man alle Backend Funktionen angelegt hat. Bei einer zu großen ID treten Fehlermeldungen beim übersetzten auf. Bei einer zu kleinen ID wird irgendetwas nicht korrekt ausgeführt.

```
#define _LCDML_BACK_cnt    1  // last backend function id
```

Danach werden mit die benötigten Variablen angelegt mit:

```
LCDML_BACK_init(_LCDML_BACK_cnt);
```

Anschließend wird die Backensystemstruktur angelegt. Im Beispiel sieht das wie folgt aus:

```
// (id, interval_time, init_status, callback_function)
LCDML_BACK_new_timebased_static  (0 , ( 20UL )   , _LCDML_start  , LCDML_BACKEND_control);
LCDML_BACK_new_timebased_dynamic (1 , ( 1000UL ) , _LCDML_stop   , LCDML_BACKEND_menu);
LCDML_BACK_create();
```

Mit LCDML_BACK_new_timebased_**static** wird eine Backendfunktion angelegt deren Intervalzeit nicht mehr zur Laufzeit des Programms verändert werden kann. Mit LCDML_BACK_new_timebased_**dynamic** wird eine Funktion angelegt bei der die Intervall-

zzeit während des Programms noch verändert werden kann. Die dynamic Funktion benötigt 4Byte mehr an RAM als die _static Funktion.

Die Parameter der beiden Funktionen haben die gleiche Bedeutung:

Param 1 – id: Die ID muss mit 0 beginnend fortlaufend hochgezählt werden, sowie auch schon beim Menü. Die ID gibt die Reihenfolge an mit der die Intervallzeiten, der unterschiedlichen Einträge, abgefragt werden.

Param 2 – intervall_time [ms]: Die Zeit gibt an in welchem Intervall der Inhalt der Callback Funktion aufgerufen wird.

Param 3 – init_status: Der Init Status gibt an ob der Intervall-Timer einer Funktion gestartet oder gestoppt sein soll bei der Initialisierung des Programms. Der Status kann im weiteren Verlauf verändert werden.

Param 4 – callback_function: Gibt an welche Backendfunktion aufgerufen werden soll.

Im Beispiel ist die „LCDML_BACKEND_control“ Einheit immer aktiv. Über die Einheit wird das Menü gesteuert. Mit „LCDML_BACKEND_menu“ werden die Menüfunktionen aufgerufen. Diese Einheit wird erst gestartet wenn ein Menüpunkt aktiv ist und wieder gestoppt, wenn dieser beendet ist. Später dazu mehr.

4.1.8 Arduino - Setup

Im Setup wird die für dieses Beispiele benötigte Serielle Schnittstelle initialisiert. Darauf folgend wird die Gruppe 1, in der sich alle oben angelegten Menüelemente befinden aktiviert, sodass diese sichtbar ist. Danach werden die oben automatisch angelegten Variablen mit dem Makro „LCDML_setup(_LCDML_BACK_cnt);“ anhand der zuvor eingestellten Menükonfiguration initialisiert.

```
void setup()
{
  // serial init; only be needed
  Serial.begin(9600);   // start serial

  LCDML_DISP_groupEnable(_LCDML_G1); // enable group 1

  LCDML_setup(_LCDML_BACK_cnt);
}
```

4.1.9 Arduino - Loop

In der Loop Schleife befindet sich nur die LCDML_run(_LCDML_priority); Funktion, diese kümmert sich um den Aufruf der Funktion die im Backendsystem laufen. Dazu auch später mehr.

```
void loop()
{
  // this function must called here, do not delete it
  LCDML_run(_LCDML_priority);
}
```

4.1.10 Backendeinheit – Controll

Für die Backendeinheit „LCDML_BACKEND_control“, sowie für alle anderen Backendeinheiten werden drei Funktionen benötigt:

```
// ********************************************************************
void LCDML_BACK_setup(LCDML_BACKEND_control)
// ********************************************************************
{
}
```

Die erste Funktion LCDML_BACK_setup …. ist das Setup dieser Funktion und wird immer nur einmalig ausgeführt. Man kann mit speziellen Funktionen das erneute Ausführen erzwingen. Dazu später mehr.

```
boolean LCDML_BACK_loop(LCDML_BACKEND_control)
{
  LCDML_CONTROL_serial();

  return true;
}
```

LCDML_BACK_loop ist die Loop Funktion der Backend Einheit. Diese wird in dem zuvor definierten Zeitintervall aufgerufen. Das „return true“ ist für die Prioritätenverteilung der Intervallzeiten zuständig. Dazu später mehr.

```
void LCDML_BACK_stable(LCDML_BACKEND_control)
{
}
```

LCDML_BACK_stable wird nur ausgeführt, wenn eine Backendfunktion mit „stable_stop“ angehalten wird. Hier kann Quellcode eingefügt werden der einen stabilen Zustand des Automaten erlaubt.

In der Backendloop Funktion wird die Ansteuerungsfunktion des Menüs aufgerufen. Je nachdem welche Funktion mit „_LCDML_CONTROL_cfg“ aktiviert ist lässt sich das Menü mit verschiedenen Möglichkeiten ansteuern.

Für die Serielle Schnittstelle sieht diese wie folgt aus:

```
#if(_LCDML_CONTROL_cfg == 0)
// settings
  # define _LCDML_CONTROL_serial_enter           'e'
  # define _LCDML_CONTROL_serial_up              'w'
  # define _LCDML_CONTROL_serial_down            's'
  # define _LCDML_CONTROL_serial_left            'a'
  # define _LCDML_CONTROL_serial_right           'd'
  # define _LCDML_CONTROL_serial_quit            'q'
// ********************************************************************
// setup
void LCDML_CONTROL_setup()
{
}
// ********************************************************************
// loop
void LCDML_CONTROL_loop()
{
  // check if new serial input is available
  if (Serial.available()) {
    // read one char from input buffer
    switch (Serial.read())
    {
      case _LCDML_CONTROL_serial_enter:  LCDML_BUTTON_enter(); break;
      case _LCDML_CONTROL_serial_up:     LCDML_BUTTON_up();    break;
      case _LCDML_CONTROL_serial_down:   LCDML_BUTTON_down();  break;
      case _LCDML_CONTROL_serial_left:   LCDML_BUTTON_left();  break;
      case _LCDML_CONTROL_serial_right:  LCDML_BUTTON_right(); break;
      case _LCDML_CONTROL_serial_quit:   LCDML_BUTTON_quit();  break;
      default: break;
    }
  }
}
```

4.1.11 Displayfunktionen

Jede Displayeinheit die im Menü angelegt wird besteht aus drei Funktionen, ähnlich wie beim Backendsystem. Diese Funktionen trennen die Einheit in logische Funktionen:

- **LCDML_DISP_setup** wird ausgeführt, wenn aus dem Menü eine Funktion gestartet wird. Danach erst wieder nachdem die Funktion beendet wurde.
- **LCDML_DISP_loop** hingegen der ersten LCDMenuLib Versionen läuft die Loop Funktion nicht mehr kontinuierlich. Die Loop Funktion ist nun von Triggern abhängig. Jedes betätigen eines Buttons löst ein Trigger aus der den Inhalt dieser Funktion aktualisiert. Zudem kann im Setup ein Trigger gesetzt werden in dessen Zeitintervall diese Funktion aufgerufen wird.
  In der Loopfunktion muss die Endbedingung z.B. durch Buttons der Funktion festgelegt werden.
- **LCDML_DISP_loop_end** wird ausgeführt nachdem die Loop-Schleife beendet wurde. Die Funktion erlaubt es alle Displayfunktionsvariablen in einen stabilen Zustand zu setzen

Dieser Aufbau muss zwingend verwendet werden damit keine Fehler beim übersetzten der Lib entstehen. Zudem wird durch diesen Aufbau eine hoffentlich ordentliche Programmstruktur zu verwenden.

Im Beispiel werden drei Displayfunktionen verwendet:

Die erste Funktion „LCDML_FUNC_information“ zeigt einen Text an. Der Text wird einmalig im Setup geschrieben. Danach wird in der Loop Schliefe nur überprüft ob die Funktion beendet werden soll.

```
// ********************************************************************
void LCDML_DISP_setup(LCDML_FUNC_information)
// ********************************************************************
{
  // setup function
  lcd.setCursor(0, 0);
  lcd.print(F("Um Funktion zu"));
  lcd.setCursor(0, 1);
  lcd.print(F("schliessen eine"));
  lcd.setCursor(0, 2);
  lcd.print(F("Taste druecken oder"));
  lcd.setCursor(0, 3);
  lcd.print(F("Back Taste verwenden"));
}

void LCDML_DISP_loop(LCDML_FUNC_information)
{
  // loop function, can be run in a loop when LCDML_DISP_triggerMenu(xx) is set
  // the quit button works in every DISP function without any checks; it starts the loop_end
function

  if(LCDML_BUTTON_checkAny()) { // check if any button is presed (enter, up, down, left,
right)
    // LCDML_DISP_funcend calls the loop_end function
    LCDML_DISP_funcend();
  }
}

void LCDML_DISP_loop_end(LCDML_FUNC_information)
{
  // this functions is ever called when a DISP function is quit
  // you can here reset some global vars or do nothing
}
```

Die zweite Funktion „LCDML_FUNC_timer_info“ lässt einen Zähler von 10 bis 0 zählen und schließt sich dann von selbst. Der Zähler arbeitet im Sekundentakt. Zur Veranschaulichung ist im Setup ein Trigger auf 100 ms gesetzt. Sprich es wird 10 Mal pro Sekunde überprüft ob eine Sekunde um ist. In real wäre das Resourcenverschwendung. Des Weiteren wird in der Loopschleife der Initscreentimer zurückgesetzt. Dies verhindert, dass dieser angezeigt wird.

```
// ********************************************************************
uint8_t g_func_timer_info = 0;  // time counter (global variable)
unsigned long g_timer_1 = 0;    // timer variable (globale variable)
void LCDML_DISP_setup(LCDML_FUNC_timer_info)
// ********************************************************************
{
  // setup function
  lcd.print(F("x sec warten")); // print some content on first row
  g_func_timer_info = 10;       // reset and set timer
  LCDML_DISP_triggerMenu(100);  // starts a trigger event for the loop function every 100
millisecounds
}

void LCDML_DISP_loop(LCDML_FUNC_timer_info)
{
  // loop function, can be run in a loop when LCDML_DISP_triggerMenu(xx) is set
  // the quit button works in every DISP function without any checks; it starts the loop_end
function

  // this function is called every 100 millisecounds

  // this timer checks every 1000 millisecounds if it is called
  if((millis() - g_timer_1) >= 1000) {
    g_timer_1 = millis();
    g_func_timer_info--;                // increment the value every secound
    lcd.setCursor(0, 0);                // set cursor pos
    lcd.print(g_func_timer_info);       // print the time counter value
  }

  // reset the initscreen timer
  LCDML_DISP_resetIsTimer();

  // this function can only be ended when quit button is pressed or the time is over
  // check if the function ends normaly
  if (g_func_timer_info <= 0)
  {
    // end function for callback
    LCDML_DISP_funcend();
  }
}

void LCDML_DISP_loop_end(LCDML_FUNC_timer_info)
{
  // this functions is ever called when a DISP function is quit
  // you can here reset some global vars or do nothing
}
```

Die Dritte Funktion „LCDML_FUNC_p2“ zeigt ein Beispiel wie innerhalb einer Menüfunktion mit Buttons gearbeitet werden kann. Hier muss der Button „links“ oder „oben“ dreimal gedruckt werden.

```
// ********************************************************************
uint8_t g_button_value = 0; // button value counter (global variable)
void LCDML_DISP_setup(LCDML_FUNC_p2)
// ********************************************************************
{
  // setup function
  // print lcd content
  lcd.setCursor(0, 0);
  lcd.print(F("press left or up"));
  lcd.setCursor(0, 1);
  lcd.print(F("count: 0 of 3"));
  // Reset Button Value
  g_button_value = 0;
}

void LCDML_DISP_loop(LCDML_FUNC_p2)
{
  // loop function, can be run in a loop when LCDML_DISP_triggerMenu(xx) is set
  // the quit button works in every DISP function without any checks; it starts the loop_end
function

  if (LCDML_BUTTON_checkAny()) // check if any button is pressed (enter, up, down, left,
right)
  {
    if (LCDML_BUTTON_checkLeft() || LCDML_BUTTON_checkUp()) // check if button left is pressed
    {
      LCDML_BUTTON_resetLeft(); // reset the left button
      LCDML_BUTTON_resetUp(); // reset the left button
      g_button_value++;

      // update lcd content
      lcd.setCursor(7, 1); // set cursor
      lcd.print(g_button_value); // print change content
    }
  }

  // check if button count is three
  if (g_button_value >= 3) {
    // end function for callback
    LCDML_DISP_funcend();
  }
}

void LCDML_DISP_loop_end(LCDML_FUNC_p2)
{
  // this functions is ever called when a DISP function is quit
  // you can here reset some global vars or do nothing
}
```

5 Displaysystem – Funktionen / Makros

Im folgenden befindet sich eine Liste mit den unterstützten Funktionen und Makros.

5.1 Konstanten

```
LCDML_root
```

```
LCDML_FUNC
```

```
LCDML_FUNC_back
```

5.2 Makros

```
LCDML_BUTTON_enter()
LCDML_BUTTON_up()
LCDML_BUTTON_down()
LCDML_BUTTON_left()
LCDML_BUTTON_right()
LCDML_BUTTON_quit()
```

```
LCDML_BUTTON_checkAny()
LCDML_BUTTON_checkEnter()
LCDML_BUTTON_checkUp()
LCDML_BUTTON_checkDown()
LCDML_BUTTON_checkLeft()
LCDML_BUTTON_checkRight()
```

```
LCDML_BUTTON_resetAll()
LCDML_BUTTON_resetEnter()
LCDML_BUTTON_resetUp()
LCDML_BUTTON_resetDown()
LCDML_BUTTON_resetLeft()
LCDML_BUTTON_resetRight()
```

```
LCDML_DISP_setup(disp_func_name)
LCDML_DISP_loop(disp_func_name)
LCDML_DISP_loop_end(disp_func_name)
```

```
LCDML_DISP_groupEnable(g)
LCDML_DISP_groupDisable(g)
```

```
LCDML_DISP_triggerMenu(intervall_time_in_ms)
```

```
LCDML_DISP_jumpToFunc(name)
LCDMenuLib_getElementName(var, element_id) // überprüfen
LCDMenuLib_getElementNameChecked(var, element_id, check) // überprüfen
LCDML_DISP_funcend();
```

5.3 Funktionen

```
LCDML.getLayer();
```

```
LCDML.getFunction();
```

```
LCDML.getCursorPos();
```

```
LCDML.goBack();
```

```
LCDML.goRoot();
```

```
LCDML.display();
```

6 Backendsystem

6.1 Einleitung

Oft ist es so, dass große Programme irgendwann unübersichtlich werden. Ab diesen Punkt hilft einem nur noch Abstraktion des Codes und eine Zerlegung auf mehrere Dateien (Tabs oder Libs). Um diesem Prozess zu vereinfachen habe ich eine Lib geschrieben die Threads generiert. Jeder Thread kann zeitbasiert (timebased) oder eventbasiert (eventbased) angelegt werden. Bei den zeitbasierten Threads wird nochmal zwischen Threads mit statischer und dynamischer Ausführzeit unterschieden. Bei den dynamischen Threads kann die Zeit zur Laufzeit des Programmes noch verändert werden. Wenn mehrer Threads angelegt sind, müssen diese nicht mehr einzeln in der Loop Funktion aufgerufen werden. Dazu gibt es eine Run-Funktion die automatisch, entweder der Reihenfolge nach oder Priorität orientiert die einzelnen Threads aufruft.

Allgemein gilt für das Thread Management
Für jeden Thread werden zwei Bit benötigt. In diesen wird gespeichert ob der Thread gestartet/gestoppt ist und ob der Therad zurückgesetzt werden soll. Für den ersten Thread der angelegt wird, werden zwei Byte reserviert. Jeder weitere Thread bis zum achten greift auf diese zwei Byte zurück. Erst der neunte Thread reserviert erneut zwei Byte.
1 - 8 => zwei Byte
9 - 16 => zwei weitere Byte ...
Diese Speicherbelegung trifft auf eventbasierte Threads zu.

zeitbasierter Thread mit statischer Zeit
Auf Basis der allgemeinen Belegung, werden hier vier weitere Bytes im Ram belegt

zeitbasierter Thread mit dynamischer Zeit
Auf Basis der allgemeinen Belegung, werden hier acht weitere Bytes im Ram belegt

Initialisierung von Threads:

6.2 Funktionen / Makros / Konstanten

// konstants

```
_LCDML_no_priority
_LCDML_priority
```

```
_LCDML_stop
_LCDML_start
_LCDML_stable
```

// Makros

```
LCDML_BACK_create
```

Aufbau eines Threads:
Jeder Thread besteht aus einer SETUP und LOOP Funktion. Die Setup Funktion wird einmal zu Beginn aufgerufen. Anschließend wird ausschließlich die Loop Funktion ausgeführt. Ein Thread kann gestartet und gestoppt werden. Mit dem Reset Befehl setzt man einen Thread zurück, sprich beim nächsten Aufruf wird die Setup-Funktion erneut aufgerufen. Jeder Thread kann mit "isRun" überprüft werden, ob dieser gestartet oder gestoppt ist. Zudem gibt es die Möglichkeit
Befehle:

```
LCDML_BACK_setup(name)
LCDML_BACK_loop(name)
LCDML_BACK_stable(name)
```

```
LCDML_BACK_init
```

```
LCDML_run
```

```
LCDML_BACK_new_timebased_static(id, init_time, status, name)
LCDML_BACK_new_timebased_dynamic(id, init_time, status, name)
LCDML_BACK_new_eventbased(id, name)
```

```
LCDML_BACK_start(name)
LCDML_BACK_stop(name)
LCDML_BACK_stopStable(name)
LCDML_BACK_reset(name)
LCDML_BACK_restart(name)
```

```
LCDML_BACK_event_start(name)
LCDML_BACK_event_reset(name)
LCDML_BACK_event_restart(name)
```

```
LCDML_BACK_all_start()
LCDML_BACK_all_stop()
LCDML_BACK_all_reset()
LCDML_BACK_all_restart()
```

Um Kontrollfunktionen bei vielen Threads zu vereinfachen, können Gruppen von Threads angelegt werden. Jede Gruppe kann einzeln gestartet, gestoppt, zurückgesetzt oder zurückgesetzt und gestartet werden.
Befehle:

```
LCDML_BACK_group(thread_name)
LCDML_BACK_group_init(group_name, thread_cnt)
LCDML_BACK_group_start(group_name)
LCDML_BACK_group_stop(group_name)
LCDML_BACK_group_reset(group_name)
LCDML_BACK_group_restart(group_name)
```

```
LCDML_BACK_signal(id, name)
LCDML_BACK_signal_init(cnt)
LCDML_BACK_signal_set(name)
LCDML_BACK_signal_get(name)
LCDML_BACK_signal_reset(name)
```

```
LCDML_BACK_isRun(name)
```

```
LCDML_BACK_dynamic_setLoopTime(name, time)
LCDML_BACK_dynamic_getLoopTime(name)
LCDML_BACK_dynamic_setDefaultTime(name)
LCDML_BACK_dynamic_restartTimer(name)
LCDML_BACK_dynamic_timeToZero(name)
```

Zur Vereinfachung gibt es auch die Möglichkeit die Setup- oder Loop- Funktion eines Threats direkt aufzurufen. Dafür wurden folgende Befehle eingeführt:

// direct call

```
LCDML_BACK_call(name)
LCDML_BACK_call_setup(name)
LCDML_BACK_call_loop(name)
LCDML_BACK_call_stable(name)
```

Aus alter Beschreibung:

Da es Event- Zeitbasierte Threads gibt wird der Ablauf wie im folgenden Bild generiert. Der Thread der dort eingezeichnet ist, wurde vom Ablauf her weiter oben schon beschrieben.

Bild

Dieser Ablauf der nun Beschrieb wurde ist in den gesamten Ablauf eingebettet. Der gesamte Ablauf kann der Reihenfolge nach oder mittels einer Priorität durchlaufen werden.

Bild

der Reihenfolge nach (A):
Bei jedem Thread wird der Reihenfolge nach überprüft ob dieser gestartet wurde und ob bei zeitbasierten Threads die Wartezeit abgelaufen ist oder bei eventbasierten Threads das ein Event gestartet wurde. Wenn keine Bedingung erfüllt ist, wird der nächste Thread überprüft. Ansonsten wird der Thread ausgeführt.

mittels Priorität (B)
Es findet die gleiche Überprüfung wie schon unter "der Reihenfolge nach (A)" beschrieben statt. Der Unterschied liegt in der Überprüfung der Bedingungen. Wenn keine Bedingung erfüllt ist, passiert das gleich wie unter (A). Wenn der Thread ausgeführt wird, beginnt anschließend die Schleife von neuem. Je nach Rückgabewert der Thread-Loop-Funktion. Ist der Rückgabewerte "null" oder "false" startet die Schleife die die einzelnen Threads aufruft neu. Wenn der Wert "true" ist, beginnt die Prozedur von neuem.

Das System kann man so verstehen:

Bils

6.3 Versteckte Funktion zur Task-Auswahl

Der nachfolgende Code befindet sich innerhalb der Lib hinter einem Makro versteckt. Damit in diesem Code kein Fehlerdurch Veränderungen bzw. Verwirrung beim verstehen entsteht wird dieser Code von der Lib bereitgestellt.

```
#define LCDML_BACK_create()                                                 \
    void LCDML_BACK_setup(LCDML_BACKEND_menu)                               \
    {                                                                       \
        g_LCDML_BACK_lastFunc = LCDML.getFunction();                        \
        if (g_LCDML_DISP_functions_loop_setup[g_LCDML_BACK_lastFunc] ==
LCDML_FUNC_loop_setup) {                                                    \
            bitSet(LCDML.control, _LCDML_control_funcend);                  \
        }                                                                   \
        else if (g_LCDML_BACK_lastFunc != _LCDML_NO_FUNC) {                 \
            LCDML_lcd_menu_clear();                                         \
            LCDML_BUTTON_resetAll();                                        \
            g_LCDML_DISP_functions_loop_setup[g_LCDML_BACK_lastFunc]();     \
        }                                                                   \
    }                                                                       \
    boolean LCDML_BACK_loop(LCDML_BACKEND_menu)                             \
    {                                                                       \
        if (LCDML.getFunction() != _LCDML_NO_FUNC) {                        \
            g_LCDML_DISP_functions_loop[LCDML.getFunction()]();             \
        }                                                                   \
        return true;                                                        \
    }                                                                       \
    void LCDML_BACK_stable(LCDML_BACKEND_menu)                              \
    {                                                                       \
        if (g_LCDML_BACK_lastFunc != _LCDML_NO_FUNC) {                      \
            g_LCDML_DISP_functions_loop_end[g_LCDML_BACK_lastFunc]();       \
            g_LCDML_BACK_lastFunc = _LCDML_NO_FUNC;                         \
            LCDML_lcd_menu_clear();                                         \
            LCDML.display();                                                \
            LCDML_lcd_menu_display();                                       \
            LCDML_BUTTON_resetAll();                                        \
            LCDML.function = _LCDML_NO_FUNC;                                \
            bitClear(LCDML.control, _LCDML_control_funcend);                \
            if(g_lcdml_jump_func != _LCDML_NO_FUNC) {                       \
                LCDML.jumpToElement(g_lcdml_jump_func);                     \
                LCDML_DISP_update_menu();                                   \
                g_lcdml_jump_func = _LCDML_NO_FUNC;                         \
            }                                                               \
            if(bitRead(LCDML.control, _LCDML_control_go_root)) {            \
                LCDML.goRoot();                                             \
                LCDML.display();                                            \
                LCDML_lcd_menu_display();                                   \
            }                                                               \
        }                                                                   \
    }
```

Dieser Code erklärt wieso im BACKEND Tab der Beispiele nur eine Backendeinheit angelegt ist. Die Zweite steht in einem Makro.

7 Known Errors

7.1 Wrong number

```
In file included from D:\Programme\arduino-1.6.7-
windows\hardware\arduino\avr\cores\arduino/Arduino.h:28:0,

                 from
C:\Users\nilsf\AppData\Local\Temp\buildefd2c9e57b647fda1f0605a716ef886e.tmp\sketch\LCDML_205_b
ackend.ino.cpp:1:

D:\Programme\arduino-1.6.7-windows\libraries\LCDMenuLib/LCDMenuLib_makros.h:160:50: error:
redefinition of 'const char g_LCDML_DISP_lang_3_var []'

   const char g_LCDML_DISP_lang_ ## name ##_var[] PROGMEM = { content }; \
```

Nummerierung im Menü ist falsch. Eine Zahl ist Doppelt g_LCDML_DISP_lang_**3**_var