MENU LIBRARY

TUTORIAL

WHAT IS THE MENU LIBRARY?

The Menu Library is a simple way to add a menu to your Sketches. Menus are useful if you want the person using your sketch (the user) to choose what **actions** needs to be done, and in what order.

HOW DOES IT WORK?

To use a menu you need three things:

- A way to display the menu (LCD, OLED Display, the Serial Monitor, …);
- A way to navigate the menu (usually a keypad);
- A system that can help you to display and navigate the menu (Menu Library).

Basically, you will setup your display device, setup your keypad and setup the menu. During the loop() part of your sketch, you will tell the Menu Library what key was last pressed, and, in return, the Menu Library will tell you what to print on your display (update your menu) and what action the user wants to be carried out.

SETTING UP YOUR DISPLAY DEVICE.

Please refer to the manufacturer of your device to learn how to do it properly. You should always run one of the example Sketches (Hello World) that comes with the LCD Library before attempting to use the Menu Library.

Example : the <LiquidCrystal.h> Library:

```
//Standard LCD using 6 pins========================================
//This is the library that comes with Arduino’s IDE
//See https://www.arduino.cc/en/Tutorial/HelloWorld to learn more
//Change the pin numbers to match those on your Arduino.
//-----------------------------------------------------------------

#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11,     5,   4,   3,   2);
//                 (RS, Enable, D4,  D5,  D6,  D7)
int lcdNumCols = 16;
int lcdNumLines = 2;
//-----------------------------------------------------------------
```

You will also need a routine that displays the menu:

```
//showMenu=============================================================
//Sends the current menu or submenu to your LCD.
//Updated only if needed.
//---------------------------------------------------------------------
void showMenu() {
  if (menu.LcdNeedsUpdate()) {
    lcd.clear();
    for (int i = 0 ; i < lcdNumLines ; i++) {
      lcd.setCursor(0,i);
      lcd.print(menu.lcdLine(i));
    }
    delay(100);
    menu.LcdUpdated();
  }
}//showMenu-------------------------------------------------------------
```

SETTING YOUR KEYPAD.

Please refer to the manufacturer of your device to learn how to do it properly. You should always run one of the example Sketches that comes with the Keypad Library before attempting to use the Menu Library.

Example : A keypad found at Adafruit™

```
#include <Keypad.h>
const byte ROWS = 4; //four rows
const byte COLS = 3; //three columns
char keys[ROWS][COLS] = {
   {'1','2','3'},
   {'4','5','6'},
   {'7','8','9'},
   {'#','0','*'}
};
 byte rowPins[ROWS] = {6, 7, 8, 9}; //connect to the row pinouts of the keypad
 byte colPins[COLS] = {10, 11, 12}; //connect to the column pinouts of the keypad

 Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);
```

You will also need to define what directional keys will be sent to the Menu Library:

- You can use characters: (as with the above keypad)
  ```
  #define UP '2'
  #define DOWN '8'
  #define LEFT '4'
  #define RIGHT '6'
  ```
- Or you can use integers: (for another keypad Library that returns integers)
  ```
  #define UP 1
  #define DOWN 2
  #define LEFT 3
  #define RIGHT 4
  ```

The directional keys are used to navigate the menu.

UP	Will select the item before the current item if it exists.
DOWN	Will select the item after the current item if it exists.
LEFT	Will select the parent of the item if it exists.
RIGHT	Will behave differently if the menu item has a submenu or not. If it has a submenu, it will select the first item of the submenu. Otherwise, the Menu Library will return an integer to say what action the user wants you to take. (No submenu means **action!**)

You will also be able to use UP, DOWN, LEFT and RIGHT keys when carrying the actions that the user has selected. (Outside the menu.)

SETTING YOUR MENU

Describing your menu is very easy. You construct a String with the items you want in you menu, one line per item.

Example :

```
const String menuItems =
"-READ:000"
"--SENSORS:000"
"---SENSOR A1:101"
"---SENSOR A2:102"
"--SWITCHES:000"
"---SWITCH PIN 4:103"
"---SWITCH PIN 5:104"
"-SET:000"
"--SERVO ARM:105"
"--SERVO BASE:106"
"-MOVE SERVOS:107";


Menu menu(menuItems); //Set up menu
```

The first line says that it is a constant (won't change) of type String (characters), that its name is "menuItems" and that it is equal to the lines that are below. (up to the semicolon ";")

Each line holds one menu item between quotes (") and contains:

- Dashes to specify the level of the item;
- The label of the item to be displayed on the LCD;
- A colon ":";
- A number between "000" and "999" to identify an **action** to be performed.
  (It has to be exactly 3 characters long and only be digits.)
  (use "000" to say : "I have a submenu.")

Don't forget to place a semi-colon ";" at the end of the last line.

Finally, create an instance of the menu.

That's it, you're done!

After this, the menu library knows that:

The MAIN MENU is:

```
"-READ:000"
"-SET:000"
"-MOVE SERVOS:107"
```

READ has a submenu:

```
"--SENSORS:000"
"--SWITCHES:000"
```

SET has a submenu:

```
"--SERVO ARM:105"
"--SERVO BASE:106"
```

SENSORS has a submenu:

```
"---SENSOR A1:101"
"---SENSOR A2:102"
```

SWITCHES has a submenu:

```
"---SWITCH PIN 4:103"
"---SWITCH PIN 5:104"
```

The menu items that have submenus are (numbered 000):

```
"-READ:000"
"--SENSORS:000"
"--SWITCHES:000"
"-SET:000"
```

The menu items that spark actions are (numbered 101 to 108):

```
"---SENSOR A1:101"
"---SENSOR A2:102"
"---SWITCH PIN 4:103"
"---SWITCH PIN 5:104"
"--SERVO ARM:105"
"--SERVO BASE:106"
"-MOVE SERVOS:107"
```

To have your Sketch do the actions selected by the user, you will use a switch:

```
void make(int action) {
  switch (action) {
    case 101: { readPin(A1, ANALOG); break; }
    case 102: { readPin(A2, ANALOG); break; }
    case 103: { readPin(4, DIGITAL); break; }
    case 104: { readPin(5, DIGITAL); break; }
    case 105: { angleServoArm = changeValue(angleServoArm); break; }
    case 106: { angleServoBase = changeValue(angleServoBase); break; }
    case 107: { servoArm.write(angleServoArm); servoBase.write(angleServoBase); break; }
  }
}
```

DO I HAVE TO DO SOMETHING SPECIAL IN THE SETUP() PART OF THE SKETCH?

Yes, you have three things to do:

- Set up your LCD :
  `lcd.begin(lcdNumCols, lcdNumLines);`
- Tell the Menu Library about your LCD :
  `menu.defineLcd(lcdNumCols, lcdNumLines);`
- Tell the Menu Library what you intent to send for the four directional keys (in that exact order):
  — You can send characters: `menu.mapKeyChar(UP, DOWN, LEFT, RIGHT);`
  — Or, you can send integers: `menu.mapKeyInt(UP, DOWN, LEFT, RIGHT);`

DO I HAVE TO DO SOMETHING SPECIAL IN THE LOOP() PART OF THE SKETCH?

Yes. You have to : (Read the remarks)

```
void loop() {
  showMenu();                      //Update the LCD with the menu.
  char key = keypad.getKey();      //Read the key.
  int action = menu.update(key);   //Read the action tagged to the (new) current menu item.
  if (action > 0) {                //If we need to act:
    make(action);                    //make it.
    menu.done();                     //We are done with this action... Go back to the menu.
  }
}
```

WHAT ARE THE METHODS THAT THE MENU LIBRARY PROVIDES?

MANDATORY METHODS

Those methods have to be called to use the Menu Library.

void Menu::menu(String menuItems) (MANDATORY)

```
Menu menu(menuItems);
```

The first method is called a constructor, which creates the menu. It has to be placed **before** setup().
Menu is the name of the constructor;
menu is the name that you will use to refer to the Menu Library throughout your sketch;
menuItems is the name that you gave to the String containing your menu.

void Menu::defineLCD(int columns, int rows) (MANDATORY)

```
menu.defineLcd(20, 4);
```

This method has to be placed in the setup(). It tells the Menu Library how many columns and how many lines your LCD has.
columns is the number of columns your LCD has;
rows is the number of rows your LCD has;

void Menu::mapKeyChar(char UP, char DOWN, char LEFT, char RIGHT) (MANDATORY)
OR...
void Menu::mapKeyInt(int UP, int DOWN, int LEFT, int RIGHT) (MANDATORY)

```
menu.mapKeyChar(UP, DOWN, LEFT, RIGHT);
menu.mapKeyInt(UP, DOWN, LEFT, RIGHT);
```

This method has to be placed in the setup(). It tells the Menu Library what are the characters or integers that you will send to the library if one of the directional keys is pressed. They have to be given in that exact order: UP, DOWN, LEFT and RIGHT.

LCD MANAGING METHODS

Those methods will allow you to print the menu to your LCD

String Menu::lcdLine(int line)

```
menu.lcdLine(0);
```

You will use this method to display (update) the menu on your LCD. It returns the label of the menu item that you have to print on a specific line of the LCD. Typically, you would call this method for each line that you have on the LCD to obtain the part of the menu that your LCD can display:

```
lcd.clear();
for (int i = 0 ; i < lcdNumLines ; i++) {
  lcd.setCursor(0,i);
  lcd.print(menu.lcdLine(i));
}
```

The label of the menu item is preceded by a caret “>” if it is the current item. It is preceded by a space otherwise.

bool Menu::LcdNeedsUpdate()

```
if (menu.LcdNeedsUpdate()) { update the lcd… }
```

You will use this method to find out if you need to update the LCD. This could happen if the user clicked one of the directional keys or if you used the LCD while carrying out an action.

void Menu::LcdUpdated()

```
LcdUpdated();
```

You will use this method to tell the Menu Library that you are finished updating the LCD with the menu items.

void Menu::UpdateLcd()

```
LcdUpdated();
```

You will use this method to tell the Menu Library that you will have to redraw the menu on the LCD in the next loop. This has the same effect that “menu.done();”

void Menu::done()

```
menu.done();
```

You will use this method to tell the Menu Library that you are finished carrying an action. This will raise the flag that will make “LcdNeedsUpdate()” return true the next time you call it.

MENU MANAGING METHOD

You will use this method tell the Menu Library that a directional key has been pressed (and to specify which one). When the Menu Library returns, it will also tell you what action has to be carried out by your sketch according to the selected menu item. If it returns zero (0), you have no action to carry. If it returns a non-zero value, you will have to carry out the action corresponding to this value.

int Menu::update(char key)   OR   int Menu::update(int key)

```
key = keypad.getKey();
action = menu.update(key);
```

You can pass either a character (char) or an integer (int) to the method. You don't have to worry which one to use; the Menu Library will sort it out. (Actually, it's Arduino's IDE that does the job.)

OTHER USEFULL METHODS

These methods will allow you to retrieve some information from the Menu Library.

String Menu::getCurrentLabel()

```
lcd.print(menu.getCurrentLabel());
```

You will use this method to get the label associated to the current menu item.

int Menu::itemNumber(String label)

```
ptr = menu.itemNumber("SWITCH PIN 5");
```

You will use this method to find the number of the **first** item that has the label "label". If you have more than one menu item with the same label, only the first one will be returned. Use it in conjunction with the next method to set the current item to a specific item in the menu. Keep reading…

void Menu::selectItem(int number)

```
menu.selectItem(menu.itemNumber("MOVE SERVOS")); \\Sets "MOVE SERVOS" as the current menu item
```

You will use this method in conjunction with the previous method, to set the current item to a specific item in the menu, using its label.

void Menu::reset()

```
menu.reset();
```

You will use this method to set the first item of the menu as the current item.

int Menu::getAction()

```
action = menu.getAction();
```

You will use this method when you need to know what is the action corresponding to the current item, without using the "menu.update()" method.

EXAMPLE

```
#include <Menu.h>

////////////////////////////////////////////////////////////////////////////////////////////THE LCD
#include <Wire.h>
#include <LiquidTWI.h>
LiquidTWI lcd(0);      //Or the I2C address you have setup your backpack
int lcdNumCols = 20;
int lcdNumLines = 4;

////////////////////////////////////////////////////////////////////////////////////////////THE SWITCHES
#include <Keypad.h>
const byte ROWS = 4; //four rows
const byte COLS = 3; //three columns
char keys[ROWS][COLS] = {
   {'1','2','3'},
   {'4','5','6'},
   {'7','8','9'},
   {'#','0','*'}
};
byte rowPins[ROWS] = {46, 47, 48, 49}; //connect to the row pinouts of the keypad
byte colPins[COLS] = {50, 51, 52};     //connect to the column pinouts of the keypad

Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);

#define UP '2'
#define DOWN '8'
#define LEFT '4'
#define RIGHT '6'

////////////////////////////////////////////////////////////////////////////////////////////THE MENU
const String menuItems =
"-READ:000"
"--SENSORS:000"
"---SENSOR A1:101"
"---SENSOR A2:102"
"--SWITCHES:000"
"---SWITCH PIN 4:103"
"---SWITCH PIN 5:104"
"-MOTOR:000"
"--START:105"
"--STOP:106"

Menu menu(menuItems); //Set up menu

////////////////////////////////////////////////////////////////////////////////////////////THE SKETCH
#define ANALOG 1
#define DIGITAL 0
#define PINMOTOR 6

////////////////////////////////////////////////////////////////////////////////////////////ACTIONS 101 to 104
void readPin(int pin, int pinType) {
  char key = '0';
  lcd.clear();
  lcd.setCursor(0,0);                                    //On the first row
  lcd.print(menu.getCurrentLabel());                     //Print the menu label.
  while(key != LEFT && key != RIGHT) {                   //Exit on LEFT or RIGHT.
    lcd.setCursor(0, 1);                                   //On the second row
    lcd.print("    ");                                     //Erase the old value.
    lcd.setCursor(0, 1);                                   //On the second row
    if (pinType == DIGITAL)                                //If DIGITAL:
      lcd.print(digitalRead(pin) ? "HIGH" : "LOW");          //Print HIGH or LOW.
    else                                                   //If not,
      lcd.print(analogRead(pin));                            //Print value (0:1023).
    key = keypad.getKey();                                 //Read the keypad.
    delay(100);                                            //Reduce flickering.
  }
}

////////////////////////////////////////////////////////////////////////////////////////////ACTION SELECT
void make(int action) {
  switch (action) {
    case 101: { readPin(A1, ANALOG); break; }
    case 102: { readPin(A2, ANALOG); break; }
    case 103: { readPin(4, DIGITAL); break; }
    case 104: { readPin(5, DIGITAL); break; }
    case 105: { digitalWrite(PINMOTOR, HIGH); break; }
    case 106: { digitalWrite(PINMOTOR, LOW); break; }
  }
}
```

```
//////////////////////////////////////////////////////////////////////////////////////////////MENU to LCD
void showMenu() {
  if (menu.LcdNeedsUpdate()) {
    lcd.clear();
    for (int i = 0 ; i < lcdNumLines ; i++) {
      lcd.setCursor(0,i);
      lcd.print(menu.lcdLine(i));
    }
    delay(100);
    menu.LcdUpdated();
  }
}

void setup() {
  lcd.begin(lcdNumCols, lcdNumLines);                  //Setup your LCD.
  menu.defineLcd(lcdNumCols, lcdNumLines);             //Describe your LCD to the Menu Library.
  menu.mapKeyChar(UP, DOWN, LEFT, RIGHT);              //Tell the Menu Library what your keys are.

  pinMode(4,INPUT_PULLUP);
  pinMode(5,INPUT_PULLUP);
  pinMode(PINMOTOR, OUTPUT)
}

void loop() {
  showMenu();                         //Update the LCD with the menu.
  char key = keypad.getKey();         //Read the key.
  int action = menu.update(key);      //Read the action taged to the curent menu item.
  if (action > 0) {                   //If we need to act:
    make(action);                       //make it.
    menu.done();                        //We are done with this action... Go back to the menu.
  }
}
```

IS THERE ANYTHING ELSE I SHOULD KNOW?

Yes. It is regarding the amount of memory that the sketch's variables occupy. When you check or upload your sketch, Arduino's IDE tells you how much memory you use for your variables. In order to let you decide how many menu items you want, the Menu Library allocates just the amount of memory necessary to store information about each of your menu item. This is done without Arduino's IDE knowing about it.

It is called dynamic memory allocation. You don't have to know anything about this technique, but you need to know that your sketch will use more memory for its variables than what Arduino reports.

How much more? The Menu Library uses 7 bytes per menu item, plus another 7 bytes for its own use. So, if your menu has 10 items you have to add (10 x 7) + 7 = 77 bytes to the amount of dynamic memory that Arduino reports to you.

If you have loads of dynamic memory (a Mega or the like) you can have menus with as many items as you wish. If you are tight on dynamic memory, you should take this into account.

I sincerely hope that the Menu Library will help you in your projects.
Jacques Bellavance
j-bellavance@videotron.ca