SIMPLE MENU LIBRARY

TUTORIAL

What is the SimpleMenu Library?

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

Menu	→ Make action → ← Return to menu ←	Actions

How does it work?

To use a menu we need three things:

- A system that can help the user to navigate the menu (SimpleMenu Library).
- A way to display the menu (LCD, LED, the Serial Monitor, …);
- A way to navigate the menu (usually a keypad, but can be an encoder…);

Basically, we will setup the menu, setup the display device, and setup the keypad.

Setting up our menu

Describing our menu is very easy. We construct a string with the items we want in you menu, one line per item.

Example :

```
char menuItems[] =
"-READ:000"
"--SENSORS:000"
"---SENSOR A1:101"
"---SENSOR A2:102"
"--SWITCHES:000"
"---SWITCH PIN 2:103"
"---SWITCH PIN 3:104"
"-SET:000"
"--SERVO ARM:105"
"--SERVO BASE:106"
"-MOVE SERVOS:107";

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

The first line says that it is of type char (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.)
  (We use "000" to say: "I have a submenu.")

We don't forget to place a semi-colon “;” at the end of the last line.

Finally, we create an instance of the menu.

That’s it, we’re done!

After this, the SimpleMenu library knows that:

The MAIN MENU is:
```
"-READ:000"
"-SET:000"
"-MOVE SERVOS:107"
```

READ has a submenu:
```
"--SENSORS:000"
"--SWITCHES:000"
```

SENSORS has a submenu:
```
"---SENSOR A1:101"
"---SENSOR A2:102"
```

SWITCHES has a submenu:
```
"---SWITCH PIN 2:103"
"---SWITCH PIN 3:104"
```

SET has a submenu:
```
"--SERVO ARM:105"
"--SERVO BASE:106"
```

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 107):
```
"---SENSOR A1:101"
"---SENSOR A2:102"
"---SWITCH PIN 2:103"
"---SWITCH PIN 3:104"
"--SERVO ARM:105"
"--SERVO BASE:106"
"-MOVE SERVOS:107"
```

Setting up our display device.

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

Example : the <LiquidCrystal.h> Library: (It comes with the Arduino IDE)

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
#define LCD_COL 16
#define LCD_LINES 2
bool menuNeedsUpdate = true;
```

The “menuNeedsUpdate” variable will allow us to re-display the menu only when needed.

To display the menu on our display, we will have to setup a routine
This routine should be placed **before** the setup() section of our sketch.

This is what it could look like:

```
void displayMenu() {
  lcd.clear();
  for (int i = 0 ; i < LCD_LINES ; i++) {
    lcd.setCursor(0, i);
    lcd.print(menu.getLine(i));
  }
  menuNeedsUpdate = false;
}
```

In the setup() section of the sketch, we may have to initialise the lcd:

```
lcd.begin(LCD_COLS, LCD_LINES);
```

In the setup() section of the sketch, we have to tell SimpleMenu how many lines our display has:

```
menu.setDisplay(LCD_LINES);
```

Setting up our keypad.

There are numerous ways to use switches or encoders with an Arduino. The choice is ours. It is strongly recommended that all our switches are debounced. To prevent multiple keypresses, it is better to wait for the button to be released before reporting it to the Library.

The only thing that the SimpleMenu Library needs, is to be told when we want to move UP, DOWN, LEFT or RIGHT in the menu. In the setup of our sketch, we will add the following line:

```
menu.setKeypad(UP, DOWN, LEFT, RIGHT);
```

Those can be any whole numbers (1..255), but they have to be specified in that exact order.

How do we navigate the menu?

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 SimpleMenu Library will return an integer (whole number between 1 and 999) to say what action the user wants you to take. (No submenu means **action!**)

In the loop() section of our sketch, we will navigate the menu with this line:

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

More on this line of code on page 7.

Setting up the actions

The actions are the numbers we placed after the colon on each menu item in our menu.
Those will be used to act when this specific item is chosen. To do this, Arduino provides a construct that does just that. We will create the following routine:

```
void make(int action) {
    case 101: { your code; break; }
    case 102: { your code; break; }
    case 103: { your code; break; }
    case 104: { your code; break; }
    case 105: { your code; break; }
    case 106: { your code; break; }
    case 107: { your code; break; }
  }
}
```

Where is stated "your code", we can do the action needed by the menu item. One way to keep this clean, is to place a function call there:

```
void make(int action) {
    case 101: { displayAnalogPin(A0); break; }
    case 102: { displayAnalogPin(A1); break; }
    case 103: { displayDigitalPin(2); break; }
    case 104: { displayDigitalPin(3); break; }
    case 105: { setServoAngle(1); break; }
    case 106: { setServoAngle(2); break }
    case 107: { moveServos(); break; }
  }
}
```

And then to write the action codes in those functions

What are the methods that the SimpleMenu library provides?

Mandatory methods

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

void SimpleMenu::SimpleMenu(char *menuItems) (MANDATORY)

```
SimpleMenu menu(menuItems);
```

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

void SimpleMenu::setDisplayLines(byte displayLines)

```
menu.setDisplayLines(LCD_LINES);
```

This method has to be placed in the setup() section of our sketch. It tells the Menu Library how many lines our LCD has.
displayLines is the number of lines our LCD has;

void SimpleMenu::setKeypad(byte UP, byte DOWN, byte LEFT, byte RIGHT)

```
menu.setKeypad(1, 2, 3, 4);
```

This method has to be placed in the setup(). It tells the Menu Library what are the numbers (1..255) that we 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 method

This method will allow us to print the menu to our LCD

char* SimpleMenu::getLine(byte line)

```
menu.getLine(0);
```

We will use this method to display (update) the menu on our LCD. It returns the label of the menu item that we have to print on a specific line of the LCD. Typically, we would call this method for each line that we have on the LCD:

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

Menu managing method

This method is used in the loop() part of the sketch.

int SimpleMenu::getAction(byte key)

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

When this line is executed, there are 3 things that happen:

- it provides the Library which key was pressed;
- the Library updates the new menu item according to the key that was pressed;
- the Library returns the action that is tagged to the new menu item.

Understanding the loop()

First, we will update the lcd with the current menu (or sub-menu). The Boolean "menuNeedsUpdate" was created when we did the setting up the display device, so we can use it here. If it is true, we will display the menu. (The displayMenu() routine that we made there took care to make it false on exit).

Then we will read the keypad. The button pressed has to return a number between 0 and 255. Let's say that 0 means no key pressed, 1=UP, 2=DOWN, 3=LEFT and 4=RIGHT. We will do the next steps only if there is a key pressed (key > 0).

> We send the key to the Library and it returns an action. 0=NO ACTION and 1..999=ACTION. We will do the next step only if there is an action to undertake.
>
> > Now we call the routine that makes all the action
>
> We have to remember that the menu has to be updated.

```
void loop() {
  if (menuNeedsUpdate) {                 //If the menu needs to be updated
    displayMenu();                         //Update it
  }
  byte key = buttonPressed();           //Read the keypad
  if (key > 0) {                        //If a key was pressed
    int action = menu.getAction(key);     //Alert the Library of a keypress and get the action
    if (action > 0) {                       //If there is an action to undertake
      make(action);                         //Do it
    }
    menuNeedsUpdate = true;               //Menu needs to be updated
  }
}
```

Example

```
/*
 * MenuV3.ino
 * Author: Jacques Bellavance
 * Released: August 10, 2017
 *
*/
//==========================================================================================SETUP THE MENU
#include "SimpleMenu.h"
char menuItems[] =
"-READ:000"
"--SENSORS:000"
"---SENSOR A0:101"
"---SENSOR A1:102"
"--SWITCHES:000"
"---SWITCH PIN 4:103"
"---SWITCH PIN 5:104"
"-SET:000"
"--SERVO ARM:105"
"--SERVO BASE:106"
"-MOVE SERVOS:107";
SimpleMenu menu(menuItems);     //Make it real

//===========================================================================================SETUP THE LCD
#include <Wire.h>
#include <LiquidTWI.h>
#define lcdLines 4
#define lcdColumns 20
bool menuNeedsUpdate = true;
LiquidTWI lcd(0);

//The display routine======================
void displayMenu() {
  lcd.clear();
  for (int i = 0 ; i < lcdLines ; i++) {
    lcd.setCursor(0, i);
    lcd.print(menu.getLine(i));
  }
  menuNeedsUpdate = false;
}//----------------------------------------

//========================================================================================SETUP THE KEYPAD
#define pinUP 3
#define pinDOWN 4
#define pinLEFT 2
#define pinRIGHT 5

#define UP 1
#define DOWN 2
#define LEFT 3
#define RIGHT 4

#include <EdgeDebounce.h>
EdgeDebounce btnUP(pinUP, PULLUP);
EdgeDebounce btnDOWN(pinDOWN, PULLUP);
EdgeDebounce btnLEFT(pinLEFT, PULLUP);
EdgeDebounce btnRIGHT(pinRIGHT, PULLUP);

//The keypad routine====================================================
//Return only when the button is released
int buttonPressed() {
  if (btnUP.closed()) { while(btnUP.closed()) {;} return UP; }
  if (btnDOWN.closed()) { while(btnDOWN.closed()) {;} return DOWN; }
  if (btnLEFT.closed()) { while(btnLEFT.closed()) {;} return LEFT; }
  if (btnRIGHT.closed()) { while(btnRIGHT.closed()) {;} return RIGHT; }
  return 0;
}//---------------------------------------------------------------------
```

```
//==========================================================================================SETUP THE ACTIONS

//The action routine======================================
  switch (action) {
    case 101: {} // { your code; break; }
    case 102: {} // { your code; break; }
    case 103: {} // { your code; break; }
    case 104: {} // { your code; break; }
    case 105: {} // { your code; break; }
    case 106: {} // { your code; break; }
    case 107: {} // { your code; break; }
  }
}//--------------------------------------------------------

//==================================================================================================SETUP
void setup() {
  Serial.begin(9600);                         //Start Serial
  lcd.begin(lcdColumns, lcdLines);            //Start LCD
  menu.setDisplayLines(lcdLines);             //Set the number of lines of the display
  menu.setKeypad(UP, DOWN, LEFT, RIGHT);      //Set the values of  the directional keypad
}

//===================================================================================================LOOP
void loop() {
  if (menuNeedsUpdate) {                 //If the menu needs to be updated
    displayMenu();                         //Update it
  }
  byte key = buttonPressed();           //Read the keypad
  if (key > 0) {                        //If a key was pressed
    int action = menu.getAction(key);     //Alert the Library of a keypress and get the action
    Serial.println(action);
    if (action > 0) {                       //If there is an action to undertake
      make(action);                         //Do it
    }
    menuNeedsUpdate = true;               //Menu needs to be updated
  }
  //You can add more code here.
  //If your keypad gets to be unresponsive,
  //it is probably because you have too much stuff going on here.
  //buttonPress() should be called at least every 500ms to feel responsive.
}
```

Is there anything else I should know?

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

It is called dynamic memory allocation. we don’t have to know anything about this technique, but we need to know that our 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 our menu has 10 items, we have to add (10 x 7) + 7 = 77 bytes to the amount of dynamic memory that Arduino reports to us.

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

I sincerely hope that the SimpleMenu Library will help you in your projects.
Jacques Bellavance