
MENWIZ Character LCD menu library for Arduino

MENWIZ 0.6.0 Quick Tour

© 2012 By Roberto Brunialti
roberto.brunialti@knowcastle.com

SUMMARY

SUMMARY	2
1. MENWIZ: A QUICK TOUR	3
1.1 Background	3
1.2 Lets go to the code, finally !	5
Library to include	5
Global variables	5
Code required to create the menu structure	6
Declare navigation devices (navButtons)	6
Few more lines to refine the example	7
How to debug (getErrorMessage)	7
All together now ! We can now assemble the whole example	7
1.3 “Advanced” functions	8
How to change the default behavior of menu terminal nodes (setBehaviour)	8
How to draw an entire formatted screen with one function (drawUsrScreen)	9
Temporized default screens (addSplash and addUsrScreen)	9
Internal variables and memory limits	9
How to use your input devices instead of standard digital buttons (addUsrNav)	10
How to save your MENWIZ variables to EEPROM (writeEeprom and readEeprom)	11
Use the internal variable sbuf to save space	12
MENWIZ change history	13

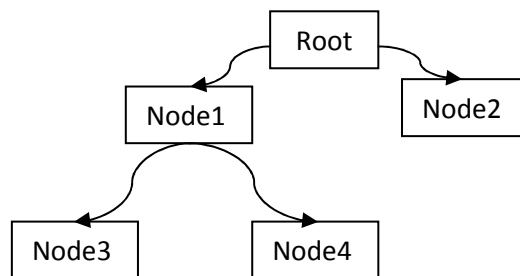
1. MENWIZ: A QUICK TOUR

1.1 Background

WARNING: This chapter is a little bit “theoretical”. You can skip it and pass directly to the second chapter. Nevertheless I suggest you to read it at some point, as it gives you the background perspective of the library and what you can expect from it now and in the future.

Technically we can define a menu as a not oriented acyclic graph, that is a hierarchical tree where all nodes are (sub)menu.

In MENWIZ all nodes are equal except one: the root. All the menu trees starts from a single node called root. There must be one and only one root node for each menu hierarchy (that is an instance of `menwiz` class in MENWIZ). Each node must declare its “parent node”, that is the ancestor node that must be traversed in order to reached the node itself. The parent node of a root is the root node itself. The root node must be declared as first node in MENWIZ.



In the above image “Root” is the parent node of “Node1”, and “Node1” is parent of “Node 3” and “Node 4”.

In MENWIZ each node is an instance of class `_menu` , even the root node. All nodes have **at least** one attribute: a label, that is the character string that likely you want to show on the LCD. In this example we assume label to be the text inside the node box (“Root”, “Node1”, ...).

All nodes within a menu tree are created using the following method of the class `menwiz` :

```
addMenu(qualifier, parent node, label);
```

In a menu structure some nodes are nothing else than containers of other child nodes. They have the only function to “organize” the different menu levels, with no contents other than the label and no specific behavior. In the example “Root”, and “Node1” are such a type of nodes.

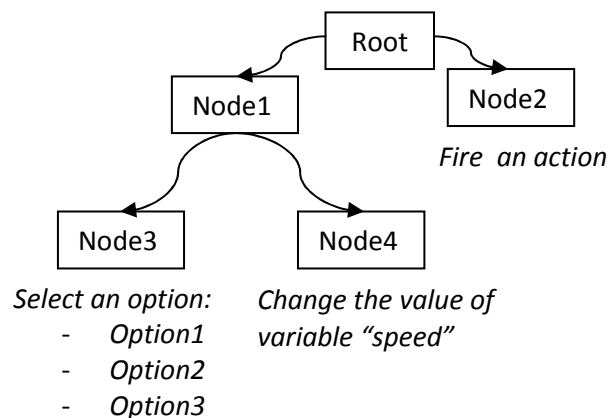
Any node having “child” nodes belongs to one of the following types (defined at creation time using addMenu method):

- root node; a root node is the first node to be created ; it is defined as root using the qualifier MW_ROOT at creation time; there is only one root node in a menu tree
- submenu, a node that has child and that is not a root node; it is defined as a submenu using the qualifier MW_SUBMENU at creation time

There is also an other type of node, as “Node2”, “Node3” and “Node4” in the example. That nodes have no “childs” (that is they are not parents of any other node). We call this kind of nodes “terminal nodes”. We

assume that once a user arrives (“navigates”) to a terminal node, he likely wants to make something more than simply going up and forth in a tree structure, for instance: selecting one of multiple options, setting/changing a variable value, running an action and so on.

In MENWIZ terminal nodes can be enriched with attributes and behaviours other than a simple label. Returning to the example, we want add some behaviors to our terminal nodes:



To reach our goal, any terminal node must have an associated user variable, in order to let the application (sketch code) be aware of the user interaction with the menu. This is done in MENWIZ binding a standard user variable to the terminal node: any change the user makes during menu interaction is available to the

So we can say that in MENWIZ any terminal node:

- must be explicitly declared as terminal node at creation time using the qualifier MW_VAR as argument.
- must be associated to a menu variable and binded to a user defined variable with the following method of class _menu:

```
addVar(variable type, binding variable, ...);
```

sketch code thru that variable but ,in the current version of MENWIZ¹, not vice-versa (any change to the variable value done inside the sketch is lost when you access the menu);

Currently MENWIZ supports the following menu variable types:

MW_LIST	a list of option to choose between
MW_BOOLEAN	a boolean value the user can toggle on/off
MW_AUTO_INT	an integer value, with min/max boundaries and increment/decrement step
MW_AUTO_FLOAT	a floating value, with min/max boundaries and increment/decrement step
MW_AUTO_BYTE	a byte value, with min/max boundaries and increment/decrement step
MW_ACTION	a user defined function to be called when the user push the enter button inside the menu terminal node

for any variable type there is a specific syntax of the method `addVar` :

```
void addVar(int, int*); // type MW_LIST
void addVar(int, int*, int, int, int); // type MW_AUTO_INT
void addVar(int, float*, float, float, float); // type MW_AUTO_FLOAT
void addVar(int, byte *,byte ,byte ,byte); // type MW_AUTO_BYTE
void addVar(int, boolean *); // type MW_BOOLEAN
void addVar(int, void (*f)()); // type MW_ACTION
```

All the above menu variables (except the MW_ACTION) have a user defined binded variable (second function argument) the sketch code can access.

1.2 Lets go to the code, finally !

Library to include

```
#include <Wire.h>
#include <LCD.h>
#include <LiquidCrystal_I2C.h>
#include <buttons.h>
#include <MENWIZ.h>
```

MENWIZ uses the “new” LiquidCrystal Library by Francisco Malpartida. This library supports I2c, 4, 8 wires and other lcd devices.

An other library needed by MENWIZ is the compact Buttons library by Franky.

Both of them are provided inside the library package and must be installed before to use MENWIZ.

Inside MENWIZ the Arduino pullup resistors for button pins are enabled. This should be enough for most commonly used buttons that should work correctly also without discrete external resistors. If you have unpredictable behavior with your buttons, you need to check if additional resistors are required.

Global variables

¹) To let the library be aware of changes happened outside itself (that is in the user sketch code) some extra memory space and code (computational resources) are required. I'm evaluating if it does not overcharge Arduino and consume too much precious memory

In this example I use a 20x4 lcd. The creation of the lcd object syntax depends from your device's interface (I2C, 4w, 8w ,...).

```
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);
menwiz tree;          //menwiz object
int list,sp=110;      // sp variable has 110 as default value
_menu *r,*s1,*s2;     //ptr to nodes to be created (1 for each level)
```

Code required to create the menu structure

```
r=tree.addMenu(MW_ROOT,NULL,"Root");
s1=tree.addMenu(MW_SUBMENU,r,"Node1");
s2=tree.addMenu(MW_VAR,s1,"Node3");
s2->addVar(MW_LIST,&list);
s2->addItem(MW_LIST,"Option1");
s2->addItem(MW_LIST,"Option2");
s2->addItem(MW_LIST,"Option3");
s2=tree.addMenu(MW_VAR,s1,"Node4");
s2->addVar(MW_AUTO_INT,&sp,0,120,10);
s1=tree.addMenu(MW_VAR,r,"Node2");
s1->addVar(MW_ACTION,myfunc);
```

Declare navigation devices (navButtons)

Menus navigaton needs a set of push buttons. MENWIZ let available to the user two options. The first requires 6 pin numbers (for the following buttons: up, down, left, right, escape, enter) to be passed to the following method of the class menwiz:

```
navButtons(up,down,left,right,escape,confirm);
navButtons(up,down,escape,confirm);
```

- up and down buttons allow to navigate menus and options;
- left and right buttons allow to increase/decrease variable values;
- escape button return back one level up without saving changes;
- return button acts as escape + changes savings.

The same function can be called with only four arguments (up,down,escape, enter). In this simpler interface schema, changes are not subject to confirmation, as they take effect immediately. To increment/decrement variables values use the up and down button.

There is also a third option: the user can provide its own callback routine if has more sophisticated input custom devices. The user provided function overload the internal one. This “advanced” option is out of the scope of this tutorial.

The line code to be inserted in the example is the long version (6 buttons), as the following (pin number is of course user defined):

```
tree.navButtons(9,10,7,8,11,12);
```

Few more lines to refine the example

The action fired under the menu node and labeled as “Node2” is part of the sketch. Let insert a trivial function writing to the serial terminal

```
void myfunc(){
  Serial.println("ACTION FIRED!");
}
```

How to debug (getErrorMessage)

It is strongly suggested, during debugging, to use the following function call after each MENWIZ function call in order to check if any error occurred during last MENWIZ library call:

```
int getErrorMessage(boolean fl);
```

the function is a method of class `menwiz`. It returns 0 if no errors occurred, an error code otherwise. If `fl` arg is equal to `true`, the function output error messages (if any) to the serial monitor. If `fl` is set to `false`, the function only returns the error code.

An other usefull function to check available sram memory is the following method of class `menwiz`:

```
int freeRam();
```

it returns the available sram bytes. You can used to check if unpredictable behaviours of your code are due to exhausted emory problems.

All together now ! We can now assemble the whole example

```
//The full code is in library example file Quick_tour.ino
#include <Wire.h>
#include <LCD.h>
#include <LiquidCrystal_I2C.h>
#include <buttons.h>
#include <MENWIZ.h>

// DEFINE ARDUINO PINS FOR THE NAVIGATION BUTTONS
#define UP_BOTTON_PIN      9
#define DOWN_BOTTON_PIN    10
#define LEFT_BOTTON_PIN    7
#define RIGHT_BOTTON_PIN   8
#define CONFIRM_BOTTON_PIN 12
#define ESCAPE_BOTTON_PIN  11

menwiz tree;
// create lcd obj using LiquidCrystal lib
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);

int list,sp=110;
```

```

void setup(){
  _menu *r,*s1,*s2;

  Serial.begin(19200);
  tree.begin(&lcd,20,4); //declare lcd object and screen size to menwiz lib

  r=tree.addMenu(MW_ROOT,NULL,"Root");
  s1=tree.addMenu(MW_SUBMENU,r,"Node1");
  s2=tree.addMenu(MW_VAR,s1,"Node3");
  s2->addVar(MW_LIST,&list);
  s2->addItem(MW_LIST,"Option1");
  s2->addItem(MW_LIST,"Option2");
  s2->addItem(MW_LIST,"Option3");
  s2=tree.addMenu(MW_VAR,s1,"Node4");
  s2->addVar(MW_AUTO_INT,&sp,0,120,10);
  s1=tree.addMenu(MW_VAR,r,"Node2");
  s1->addVar(MW_ACTION,myfunc);

  tree.navButtons(UP_BOTTON_PIN,DOWN_BOTTON_PIN,LEFT_BOTTON_PIN,RIGHT_BOTTON_PIN,ESCAPE_BOTTON_PIN,CONFIRM_BOTTON_PIN);
}

void loop(){
  tree.draw();
}

void myfunc(){
  Serial.println("ACTION FIRED");
}

```

1.3 “Advanced” functions

How to change the default behavior of menu terminal nodes (setBehaviour)

It is possible to change the behavior of some menu terminal nodes. In order to simplify the code interface there is only one method in the class `_menu`:

```
setBehaviour(byte behavihour, boolean value);
```

where `behaviour` is the behavior to activate/deactivate (use only the defined literal!) and `value` is the toggling value (true/false).

In the current version of MENWIZ are implemented the following behaviors;

Object	Behavior literal	Description	Default
Var of type MW_LIST	MW_SCROLL_HORIZONTAL	The item list is scrolled vertically by default (if set <code>true</code> it is scrolled horizontally). This can be usefull on two lines LCD	false
Var of type MW_ACTION	MW_ACTION_IMMEDIATE	By default when an action is selected, a “Confirm to run” request is prompted on the LCD (if set <code>false</code> the associated user callback is fired immediately, without confirmation)	true

How to draw an entire formatted screen with one function (drawUsrScreen)

```
void drawUsrScreen(char *str);
```

This is a method of class `menwiz`. `str` argument is a string containing all the multiline text to be displayed on the LCD. Each display line inside `str` must be terminated by char `0x0A ('\n')`. This method provide the user with a quick way to write an entire LCD screen (the lib will manage space padding, cursor position and string length checking). This function can be used in any point of the sketch code. Remember that the persistence of the text on LCD is within a single call of method `draw()`. A new call to the method `draw()` will overwrite the LCD.

Example:

```
drawUsrScreen("Test user screen\nline1\nline2\n\n");
```

The above call let the lcd display the four line user defined screen. The last line is empty.

Temporized default screens (addSplash and addUsrScreen)

MENWIZ allows the user to define two optional temporized “default” screen:

Splash screen

the one to be shown at startup time for a certain amount of seconds. It is asynchronous, that is during the splash screen the sketch can execute other code. The method of the class `menwiz` is as following:

```
void addSplash(char *str, int msec);
```

`str` argument is a string containing all the multiline text to be displayed on the LCD. Each display line inside `str` must be terminated by char `0x0A ('\n')`. The argument `msec` contains the splash screen duration in millisecs. The method manages space padding, cursor position and string length checking).

Default screen:

the one to be shown after a certain number of seconds since the last user’s menu interaction and until any interaction with the navigation buttons. It is usefull , for instance, when a sketch need to continuously show values from sensors and the menu use is a rare event. The method of the class `menwiz` is as following:

```
void addUsrScreen(void (*f)(), unsigned long elapsed);
```

`f` argument is the user defined void function (callback) called after `elapsed` millisecs from the last interaction with the menu. Inside `f` callback the user can read sensor values, perform its own task and compose its own screen. The callback is fired once for each `draw()` method call, allowing fast data refreshing to be displayed.

It is usefull to use the method `drawUsrScreen` to display a formatted screen inside the `f` callback.

Internal variables and memory limits

In order to limit the allocated memory amount, the library preallocates some array able to manage up to a maximum number of menu items (nodes) and/or options or submenus.

Those limits can be modified by the user, changing some literals in the MENWIZ.h file. Any change to the predefined values affects the memory usage.

```
#define MAX_MENU 15
```

This literal define the max number of nodes. It is equal to the maximum number of call to the addMenu methods. When the method addMenu is called a number of times greater than MAX_MENU value, the function *getErrorMessage(true)* return the value 100 and the following message is sent to the serial terminal: "E100-Too many items. Increment MAX_MENU".

```
#define MAX_OPTXMENU 5
```

This literal define the max number of options (see addItem method) within an option list and the max number of submenus (child nodes) of a single node (see addMenu method with MW_SUBMENU ARG). If the above methods are used a number of times greater than MAX_OPTXMENU value, the function *getErrorMessage(true)* return the value 105 and the following message is sent to the serial terminal: "E105-Too many items. Increment MAX_OPTXMENU".

```
#define MAX_BUFFER 84
```

This literal defines the internal LCD max buffer sizes. The value must be equal or greater than the value calculated as following: MAX_BUFFER >= LCD columns x LCD rows + rows. The default value is able to manage a LCD up to 4 rows of 20 characters each.

How to use your input devices instead of standard digital buttons (addUsrNav)

if you want to use your own device to replace the standards buttons managed by MENWIZ and Buttons libraries (and declared with navButtons functions) you need to write your own function and to declare it to MENWIZ library using addUsrNav method.

The user defined function will replace the following internal one:

```
int menwiz::scanNavButtons() {
    if (btx->BTU.check() == ON) {
        return MW_BTU;
    }
    else if (btx->BTD.check() == ON) {
        return MW_BTD;
    }
    else if (btx->BTL.check() == ON) {
        return MW_BTL;
    }
    else if (btx->BTR.check() == ON) {
        return MW_BTR;
    }
    else if (btx->BTE.check() == ON) {
        return MW_BTE;
    }
    else if (btx->BTC.check() == ON) {
        return MW_BTC;
    }
    else
        return MW_BTNNULL;
}
```

The user defined function must return one of the following integer values, defined in MENWIZ.h (always use the literals instead of the values, as values can be changed in new MENWIZ versions):

```
// BUTTON CODES
// -----
#define MW_BTNNULL      30    //NOBUTTON
#define MW_BTU          31    //UP
#define MW_BTD          32    //DOWN
#define MW_BTL          33    //RIGTH
#define MW_BTR          34    //LEFT
#define MW_BTE          35    //ESCAPE
#define MW_BTC          36    //CONFIRM
```

The returned integer code represent the last pushed button, if any, or MW_BTNNULL if no button has been pushed since last call.

The user defined function, as the internal `scanNavButtons`, is called once for every time the method `menwiz::draw` is called.

The returned code will activate the behavior associated to the pushed button (or no behaviour if no button has been pushed).

Resuming

in case of any custom device (as analog button or any other) you must:

- write your own function in the sketch (the name is up to the user)
- the function must return one of the 7 values above, depending on the pushed button (or the simulated ones)
- the function must be declared to MENWIZ with the method `addUsrNav`

WARNING: the user defined function simulating buttons have to return pushed button codes just once (that is the function must “clear” the internal status) same as with standard digital buttons! otherwise the library assumes multiple button pushes, one for each user function call....

How to save your MENWIZ variables to EEPROM (`writeEeprom` and `readEeprom`)

If your program need to save in non volatile EEPROM memory the values of the `_var` variables , you can use the following methods of the class `menwiz`

```
void menwiz::writeEeprom();
```

```
void menwiz::readEeprom();
```

WARNING: to use the above functions you need to add the following line to your sketch:

```
#include <EEPROM.h>
```

This is a break to the backward compatibility (version 0

In case of necessity, you can save some memory (about 800 bytes of program and 30 bytes of Ram) you can disable the EEPROM support commenting the following line in file MENWIZ.h

```
#define EEPROM_SUPPORT
```

Of course disabling the EEPROM support the above methods will not be available to the user. If the EEPROM support is disabled you have not to include the EEPROM.h file in the sketch.

Other programming tips

Use the internal variable sbuf to save space

If you need a buffer using sprint function carefully use the internal sbuf char buffer (its size is equal to MAX_BUFFER value in MEMWIZ.h, usually set to 84), it will save some amount of memory.

MENWIZ change history

Ver 0.6.0

New functions

```
void menwiz::writeEeprom();  
void menwiz::readEeprom();  
void _menu::setBehaviour(byte behavihour, boolean value);
```

Ver 0.5.3

Internal changes

Minor internal changes e bug finxing in the examples

Ver 0.5.0

Changes to existing functions

```
void navButtons(int up, int down, int esc, int enter);
```

method of class `menwiz`. Now MENWIZ works with only 4 buttons also (you can use both way: the old one with 6 buttons and the new one with only 4). Each argument is the Arduino pin used by the related button.

Remember:

[Up] button in variable context: increment the variable value

[Down] button in variable context: decrement the variable value

In other context up/Down buttons acts as usual (screen scrolling).

ALLOWED USER DEFINED BUTTON MANAGEMENT CALLBACK (`addUsrNav`) MUST STILL RETURN 6 VALUES (BUTTONS)!

Ver 0.4.1

Changes to existing functions

```
void addVar(int,float *,float,float,float);
```

method of class `_menu`. now MENWIZ supports variables of floating point type (`MW_AUTO_FLOAT`). The variables are displayed with a number of decimal digits set by `MW_FLOAT_DEC` global variable (default=1). The syntax is the same as integer type (`MW_AUTO_INTEGER`).

Example:

```
float gp;  
menu.addVar(MW_AUTO_FLOAT,&gp,11.00,100.00,0.5);
```

the above call create a variable of type float, binded to sketch variable gp, ranging between 11,0 and 100,0, with increment of 0,5

```
void addVar(int,byte *,byte,byte,byte);
```

method of class `_menu`. now MENWIZ supports now also variables of byte type (MW_AUTO_BYTE). The syntax is the same as integer type (MW_AUTO_INTEGER).

Example:

```
byte gp;  
menu.addVar(MW_AUTO_BYTE,&gp,0,255,1);
```

the above call create a variable of type byte, binded to sketch variable gp, ranging between 1,0 and 255, with increment of 1

Internal changes

added the global variable `MW_FLOAT_DEC` setting the number of decimal digits of floating variables (default=1);

Ver 0.3.0 CHANGES

Changes to existing functions

```
void addSplash(char * str, int millisecs);
```

method of class `menwiz`. `Str` passed to the function use `\n` (0x0A) character as line delimiter instead of previous character `'#'`

New functions

```
void addUsrNav(int (*f)());
```

method of class `menwiz`. `f` is the user defined navigation routine (callback). The user can use any device other than buttons to overwrite the internal routine. The callback **must** return an int code for any pushed "button" (MW_BTU=UP, MW_BTD=DOWN, MW_BTL=LEFT, MW_BTR=RIGHT, MW_BTE=ESCAPE, MW_BTC=CONFIRM, MW_BTNUL=NO BUTTON).

The callback is invoked on each call to the method `draw`. The used device(s) must be declared and initialized inside the sketch by the user. The callback is in charge of device debouncing (if any).

void drawUsrScreen(char *str);

method of class `menwiz`. It quick draw LCD screen with the contents of the argument string. Each line to be shown in the LCD is terminated by char 0x0A ('\n') inside the argument string. This method provide the user with the quick way to write an entire LCD screen (the lib will manage space padding, cursor position and string length checking).

Example:

```
menu.drawUsrScreen("Test user screen\nline1\nline2\n\n");
```

The above call let the lcd display the four line user defined screen. The last line is empty.

int getErrorMessage(boolean fl);

method of class `menwiz`. if `fl` is `true`, the function write a full error message to the default serial terminal, otherwise return error code only