MENWIZ character LCD menu library for Arduino

MENWIZ V 1.2.x Quick Tour

roberto.brunialti@knowcastle.com

SUMMARY

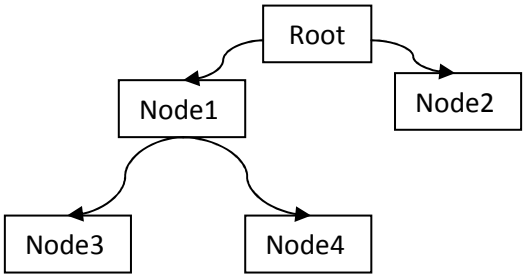
1. MENWIZ: A QUICK TOUR

1.1 Backgrounds

WARNING: This chapter is a little bit "theoretical". You can skip it and pass directly to the second chapter. Neverthless 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(int qualifier, _menu *parent, __FlashStringHelper* label);
```

The `__FlashStringHelper*` is simply the argument of macro F()

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.

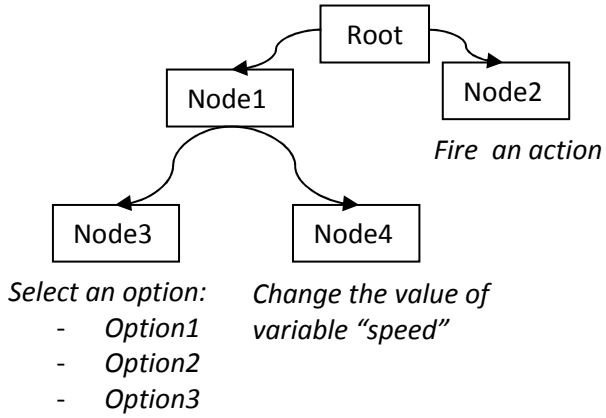
Any node having “child” nodes belongs to one of the following types (defined at creation time using `addMenu` method):

- root note; 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 also available to

So we can say that in MENWIZ any terminal node:

- must be esplicitly decla red as terminal nodeusing 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, ….);
```

the sketch code thru the binded variable itself but ,in the current version of MENWIZ[1], it is not a two way: any variable change done inside the sketch (after `addVar` declaration) is lost when you access again the menu.

Currently MENWIZ supports the following menu variable types:

`MW_LIST`	a list of option to choose between (the option index is 0!)
`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 confirm button inside the menu terminal node

to declare MENWIZ variables use method `addVar` :

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

In all the above menu variables (except `MW_ACTION`) the second method argument is the binded variable the sketch code can access. The third, fourth and fifth aguments of the numeric variables are the min, max and (auto) increment values.

1.2 Lets go to the code, finally !

Libraries to include

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

The following libraries have to be included in the sketch:

- "new" LiquidCrystal Library by Francisco Malpartida. This library supports I2c, 4, 8 wires and other lcd devices. the library is a drop replacement for the standard lib LiquidCrystal e LiquidCrystal_I2C. The latest version can be found at https://bitbucket.org/fmalpartida/new-liquidcrystal/wiki/Home.
- Buttons compact library by Franky (see also *"1.4 How to save memory space*." chapter)

[1] *) To let the library be aware of changes occourred outside its own code (that is in the user sketch code) some extra memory space and/or code (computational resources) are required. I'm evaluating if it does not overcharge Arduino and comsume too much precious memory*

- EEPROM library (one of Arduino the builtin libraries); (see also "*1.4 How to save memory space*" chapter)

The first two libraries are provided inside the MENWIZ zip file and must be installed before the use of 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

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 (`addMenu`, `addItem`, `addVar`)

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

WARNING: Please note the use of `F()` macro. Starting from version 1.0.0 all the label strings used in the `addMenu` and `addItem` methods are of type `__FlashStringHelper*` (this type is forced implicitly by the `F()` macro) instead of `char*` of the previous pre-release versions. The use of `F()` results in a significant Ram memory saving.

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:`

```
  void menwiz: (int up, int down, int left, int right,int escape, int
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 saving.

The same function can be called with only four arguments.

```
void navButtons(int up,int down,int escape,int confirm);
```

In this simpler interface schema, up and down buttons will both navigate and increase/decrease values. 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);
```

There is also a third option: the user can provide its own callback routine in order to use input custom devices. The user provided function "overload" the internal one. For details please see chapter "How to use your input devices instead of standard digital buttons".

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 (the function name is the one we declared in the `addVar` call):

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

How to debug (`getErrorMessage, freeRam`)

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

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

the function is a method of class `menwiz`. It returns 0 if no errors occourred, 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.

Error code	Description
100	Too many items. Increment MAX_MENU
130	Invalid buttons number: allowed 4 or 6
200	Undefined Root node
205	Too many items. Increment MAX_OPTXMENU
110	MW_VAR menu type required
120	Bad 1st arg in addVar

300	Undefined variable type
310	Unknown behaviour
410	Behavihour available only with 6 buttons. Ignored
900	Out of memory

Table 1.Error codes and description

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

```
int freeRam();
```

it returns the available sram bytes. It can be used to check the free memory when your program has unpredictable behaviours.

All together now ! We can now assemble the whole example

```
//The full code is in library example Quick_tour
#include <Wire.h>
#include <LCD.h>
#include <LiquidCrystal_I2C.h>
#include <buttons.h>
#include <MENWIZ.h>
#include <EEPROM.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,F("Root"));
    s1=tree.addMenu(MW_SUBMENU,r, F("Node1"));
      s2=tree.addMenu(MW_VAR,s1, F("Node3"));
        s2->addVar(MW_LIST,&list);
        s2->addItem(MW_LIST, F("Option1"));
        s2->addItem(MW_LIST, F("Option2"));
        s2->addItem(MW_LIST, F("Option3"));
      s2=tree.addMenu(MW_VAR,s1, F("Node4"));
        s2->addVar(MW_AUTO_INT,&sp,0,120,10);
```

```
    s1=tree.addMenu(MW_VAR,r, F("Node2"));
      s1->addVar(MW_ACTION,myfunc);

tree.navButtons(UP_BOTTON_PIN,DOWN_BOTTON_PIN,LEFT_BOTTON_PIN,RIGHT_BOTTON_PIN,E
SCAPE_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/look of menu nodes (`setBehaviour`)

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

```
  void setBehaviour(byte behaviour, boolean value);
```

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

The following behaviors (see table 2) are implemented in the current version of MENWIZ. Be carefull to apply this method to the proper object (defined in the "Apply to" column of the table) to avoid unpredictable results at run time.

Behavior arg	Apply to	Description	Default
MW_MENU_INDEX	class menwiz	This option enable/disable the menu index. The index shows the current selected item and the total items.	**true**
MW_MENU_COLLAPSED	class _menu of type MW_ROOT, MW_SUBMENU	This option is available only for the 6 buttons mode. It does not take effect with 4 buttons mode. This option let view and modify the values of variable subitems directly inside a menu	**false**
MW_SCROLL_HORIZONTAL	_var of type MW_LIST	By default the list items are scrolled vertically. If you set `MW_SCROLL_HORIZONTAL= true` list items are scrolled horizontally. This can be usefull on two lines LCD **Warning:** when this behavior is set=true, it set `MW_LIST_2COLUMNS` = false and `MW_LIST_2COLUMNS` =false	**false**
MW_LIST_2COLUMNS	_var of type MW_LIST	By default the list items are displayed into a single column. When set equal to `true` this option enable the two columns display mode, where up to two options per row are displayed. The max number of options is <= (rows*2-2); **Warning:** when this behavior is set=true, it set `MW_LIST_3COLUMNS` = false and `MW_SCROLL_HORIZONTAL` =false	**false**

		Warning: the max option number for MW_LIST is defined by MAX_OPTXMENU literal	
MW_LIST_3COLUMNS	_var of type MW_LIST	By default the list items are displayed into a single column. When set equal to true this option enable three columns display mode, where up to three options per row are displayed. The max number of options is <= (rows*3-3); **Warning:** when this behavior is set=true, it set MW_LIST_2COLUMNS = false and MW_SCROLL_HORIZONTAL =false **Warning:** the max option number for MW_LIST is defined by MAX_OPTXMENU literal	**false**
MW_ACTION_CONFIRM	_var of type MW_ACTION	By default when an action is selected, a "Confirm to run" request is prompted on the LCD (if set false the associated user callback is fired immediately, without confirmation)	**true**

Table 2. setBehaviour method

An example of working code using setBehaviour method is following:

```
s1=menu.addMenu(MW_VAR,r,F("TEST ACTION"));
   s1->addVar(MW_ACTION,act);
   s1->setBehaviour(MW_ACTION_CONFIRM,false);
```

How to display an entire formatted screen with just 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 to 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.

Create a splash screen (addSplash)

It is possible to create a splash screen, that is 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 do create a splash screen is the following:

```
void menwiz::addSplash(char *str, int msecs);
```

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

User default screen (`addUsrScreen`)

When the menus are no longer used, after a certain number of seconds it is possible to show a user defined screen until any navigation button is pushed again. This feature is usefull , for instance, when a sketch need to continuosly show values from sensors and the menu use is a rare event.
The method to create a default user screen is the following:

```
void menwiz::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".* This error does not halt the program, simply the menus exceeding the max value are ignored.

```
#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".* This error does not halt the program, simply the options exceeding the max value are ignored.

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

MENWIZ use the Buttons library to manage standard pushbuttons (each one using 1 Arduino pin). If you want to use your own device (for instance anal buttons, rotary encoders etc.) to replace the internal functions you need to write your own function and to declare it to MENWIZ library using addUsrNav method.

```
void addUsrNav(int (*f)(), int nb)
```

where f is your function and nb is the number of buttons emulated by f. The only allowed values for nb are 6 or 4.

If you use addUsrNav you have hot to use the navButtons function.

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_BTNULL;
  }
```

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

```
// BUTTON CODES
// -----------------------------------------------------------------------
#define MW_BTNULL      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_BTNULL if no button has been pushed since last call.

The user defined function, (same as the replaced built-in `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 `MW_BTNULL` is returned). Any other return value (or no explicit return value) makes the library behavior unpredictable.

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 writeEeprom();
void 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

1.4 How to save memory space

Here following there are some tips to reduce program footprint and get more memory for your sketch

Use the internal variable `sbuf`

If you need a buffer using sprint function carefully use the internal `sbuf` char buffer (its size, dynamically allocated, is equal to the result of this expression: rows*colums+rows.

The usage of such an internal buffer will save some amount of memory.

Do not use `sprint` function to save memory space

Since version 1.1.0 MENWIZ does'nt use the `sprintf` function to save memory space. In fact `sprintf` usage will add 1.52 Kbyte to the compiled sketch. In order to save the above mentioned memory space, even the user must avoid to use `sprintf` function. Use `strcpy`, `strcat` functions instead.

Comment `#define EEPROM_SUPPORT` if you do not use EEPROM based function

if you do not need the support of EEPROM features (that is you do *not* write or read EEPROM memory and you do not use the MENWIZ methods to save values to EEPROM) you can comment `the #define EEPROM_SUPPORT` line in MENWIZ.h to save some memory space (about 0,9 KB)
After commenting you, of course, cannot use any more the following methods:

```
void     writeEeprom();
void     readEeprom();
```

and you do not need anymore to include the following files in your sketch:

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

Comment `#define BUTTON_SUPPORT` if you do not use standard buttons

If you do not need the internal support for standard buttons and you are using the `addUsrNav` method you can save about 1,2 KB of progmem commenting the line #define BUTTON_SUPPORT in file MENWIZ.h.
After commenting you, of course, cannot use any more the following methods:

```
void     navButtons(int,int,int,int);
```

and you do not need anymore to include the following files in your sketch:

```
#include <buttons.h>
```

MENWIZ change history

Ver 1.2.0

Solved bugs

Solved few bug occourring when a root menu is declared as collapsed menu in ver 1.1.0 .

Internal changes

It is possible to disable button.h support in order to save space.

New and modified functions

New behaviours added: MW_MENU_INDEX (applies to menwiz objects).

The getVer() function is now declared as #define pseudofunction, outside menwiz class.

Ver 1.1.0

Internal changes

Sprint is not used anymore to save memory space

New functions

New behaviours added: MW_MENU_COLLAPSED (applies to _menu objects)

Ver 1.0.0

Internal changes

The "label" args of `addMenu` and `addItem` methods are now of type `__FlashStringHelper*` instead of `char*` as in the previous release

New functions

Nerw behaviours added:

MW_LIST_2COLUMNS (applies to _menu objects)
MW_LIST_3COLUMNS (applies to _menu objects)

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 nember 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 uswer 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_BTNULL=NO BUTTON).
The callback is invocated 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