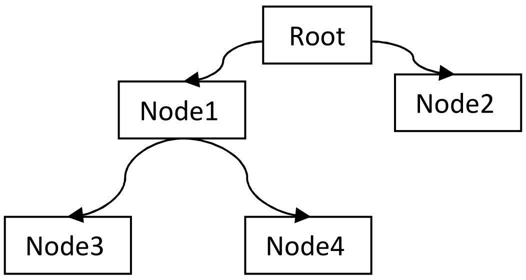
MENWIZ: A QUICK TOUR

Background

WARNING: This chapter is a little bit theoretical. You can skip it if you want to try the MENWIZ library AND learning by experience using the examples. Neverthless I suggest you to read it at some point, as it give you the background perspective of the library and what you can expect from it now and in the future.

Technically we can say that a menu in MENWIZ is simply 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 tree start from a single node called root. There must be one and only one root node in sketch. Why? Simple. Each node need to declare its “parent node”, that is the node that must be traversed in order to be reached. Each node except a root node that must be declared as first. All the nodes “departing” from root need to declare it as “parent”.



In the example above root is parent 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 one attribute: a label, that is the character string that appear on the LCD. In this example we assume label to be the text inside the node box (“Root”,”Node1”, …).

All nodes within a MENWIZ menu tree are created using the following method

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

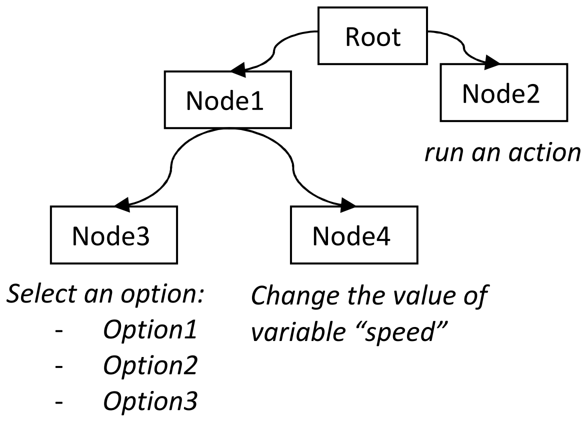
In a menu structure some node are nothing else than containers of other child nodes. They have the only function to “organize” the different menu levels, with no contents other than the label and no specific behavior. In the example “Root”, and “Node1” are such a type of nodes.

Any node having "child" nodes belongs to one of the following types (defined at creation time):

- 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 are other type of nodes. In the above example Node2, Node3 and Node4. 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 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 sketch code thru that variable and vice-versa (any change to the variable value done inside the sketch is available to the menu);

So we can say that in MENWIZ any terminal node:

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

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

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

All the above menu variables (except the `MW_ACTION`) have a user defined binded variable the user code can check and/or change.

Lets go to the code, finally !

Which library to include

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

Which global variables to create

```
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);
menwiz tree;
_menu *r,*s1,*s2; //ptr to nodes to be created (1 for each level)
//user defined variables
int list;
int speed;
```

Which code 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,&speed,0,120,10);
       s1=tree.addMenu(MW_VAR,r,"Node2");
         s1->addVar(MW_ACTION,myfunc);
```

Which code to navigate

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 function `navButtons(int,int,int,int,int,int);`

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

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

```
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 inser a trivial function writing to the serial terminal

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

All together now !

<t.b.d.>

MENWIZ changes history

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