Phi_prompt Text-bsed User Interface for Arduino

Last reviewed on 11/4/2015
Dr. John Liu

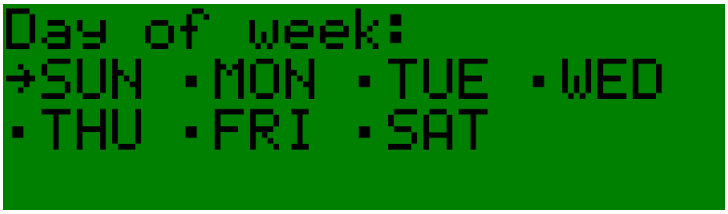
1. Introduction

Phi_prompt is the very first and best Arduino Text-based User interface with character LCDs and keypads. If you want to add interactive features to your Arduino projects, such as select lists or menus and number entry, regardless your programming skills, this library is for you. On one hand, it offers project developers a whole arsenal of user interactive functions customizable to many details. On the other hand, with simple functions, you can display an interactive menu as easily as writing one line of code:

response=simple_select_list("Day of week:\nSUN\nMON\nTUE\nWED\nTHU\nFRI\nSAT\n");

The rendering result will auto scale on 16x2 and 20x4 display as the following:

(a)



(b)

Fig. 1-1 Screen shoot of phi_prompt simple list. a) is the result on a 16*2 display. b) is the result of the same command on a 20*4 display.

The library offers Arduino project developers the freedom to focus on their projects and leave all user interaction coding to phi_prompt. Make phi_prompt the front panel of your project box so you can focus on what happens “inside the box”!

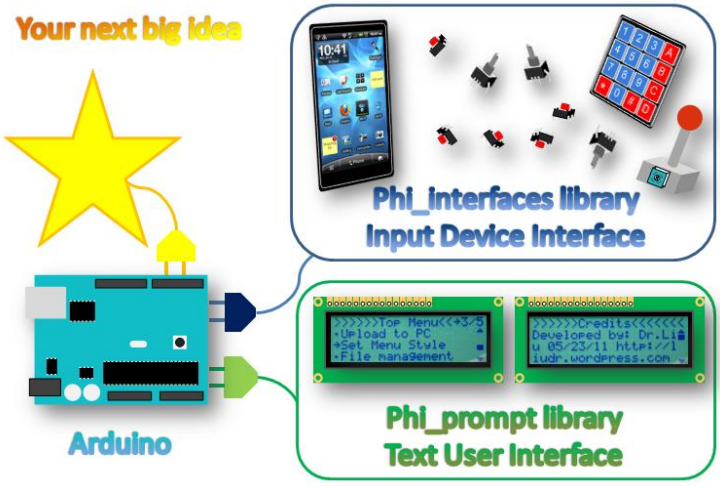


Fig. 1-2 Conceptual separation between your next project and user interfaces helps you focus on your idea and let phi_prompt handle the user interaction.

Utilizing my most recent input device class library phi_interfaces, phi_prompt library transparently collects user inputs from virtually any type of input hardware, such as buttons, matrix keypads, rotary encoders, analog buttons, 2-axis joysticks, serial keypads, and liudr keypads. The list is expanding.

Your project could use phi_prompt with any type of input combination, such as two rotary encoders with 6 buttons and a matrix keypad, all together. You may expand your keys any time you want without additional coding. You may replace one type of keys with another, such as replacing three buttons (up/down/enter) with a rotary encoder with clickable shaft. You may use as few as just two keys, one for select/up and one for confirm, to operate phi_prompt. The more keys you have on your keypad, the easier you can navigate and enter numbers and texts with phi_prompt.

You can easily start your project with virtually no input hardware and use Arduino IDE serial monitor as your serial keypad, and then decide on what buttons or knobs you want to use as inputs. You can also develop multiple interfaces, one to handle local inputs on Arduino, such as buttons or keypads, then you can add remote inputs that you can operate on your phone, PC, the internet, the same way you operate your project with those buttons. The possibility is limitless.

With the library, you will be able to show lists on the LCD. The user can browse the list and decide which item to choose. There are many features associated with lists since it is also used to generate multiple-level menus. You can set up menus of infinite levels. You can also display text areas with scroll bars to the right as an option, so you may display messages longer than the screen can display all at once. You can display YES/NO or OK dialogs, which automatically scale to fit your screen. You can also ask the user to enter information such as numbers, integers, floating point numbers, strings, password, etc. All you have to do as a developer is to call these functions and branch your program depending on their return values.

2. Main features

This powerful library was originally developed for my family of phi shields and later expended to support other compatible hardware. To obtain phi shield kits, visit my inmojo store:

http://www.inmojo.com/store/liudr-arduino-and-physics-gadgets/
http://liudr.wordpress.com/shields/phi-2-shield/

Here are the main features of the phi_prompt library:

- ☆ Compatible with Arduino IDE 1.6.0 and previous versions.
- ☆ Supports most input hardware and any number of local or remote interfaces.
- ☆ Support simple functions to do menus etc. with one line of code.
- ☆ Supports multiple-level menu. See fig. 2-1. See the menu example code how to construct a menu from a list of menu items. Use my template to save you time.
- ☆ Supports lists/menus with a variety of customizable features. See fig. 2-1 and 2-2.
- ☆ Supports multi-select lists to select more than one item on a list
- ☆ Supports dynamic contents that are constantly updated on the menu
- ☆ Supports in-menu-adjust features to adjust parameters without selecting an item
- ☆ Supports text areas to display long messages. You don't have to clip your messages in small chunks to display on 16X2 or 20X4 displays. Text areas do that for you. The user can use up/down keys to scroll up and down to read the entire message. This version has text_area() and text_area_P(), which displays message from SRAM or PROGMEM to LCD. See fig. 2-3.
- ☆ Scrolls menu/list items horizontally if a long item won't fit on a list. See fig. 2-4.
- ☆ Function to display texts automatically center-aligned. No need to count characters. See top side of fig. 2-1 thru 2-4.
- ☆ Supports number, integer entry, floating point number, and text entries. Easy to construct a password panel or ask user to enter a file name. Use my template to save time. See later sections for examples and function details.
- ☆ Lists and long messages can be stored in PROGMEM or SRAM for flexibility.
- ☆ Displays a vertical scroll bar anywhere you want. See fig. 2-1,-23 and 2-4.
- ☆ Displays yes/no and ok dialog that automatically scales to the size of your screen.
- ☆ Collect user button presses from all inputs with wait_on_escape(). You don't have to write your own code anymore. The return value tells you which button was pressed and indicates if a function key is pressed, in case you want to develop your own extension to the library.

- ☆ Showing off your project with professional-looking interfaces and control it locally and remotely!
- ☆ For details of currently supported input devices, please visit the phi_interfaces library page: http://liudr.wordpress.com/phi_interfaces/

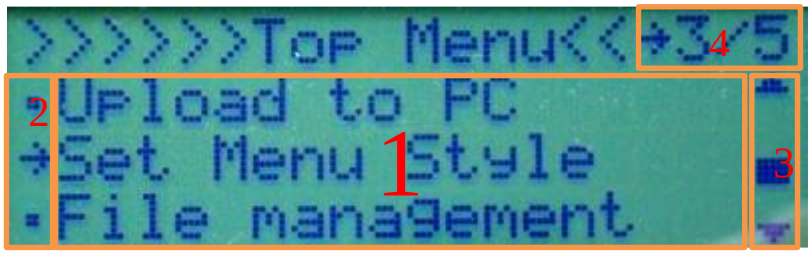


Figure 2-1. Menu/list example. 1. Main menu/list area displays 3 items in one column (can display any rows and columns combination). 2. Indicator of the highlighted item is the one in the middle (also auto center is enabled so the highlighted item is always displayed in the center. 3. Scroll bar indicates where you are at in the list. 4. Index style 2 tells the current and total number of menu/list items. The menu/list title is printed with center_text() so that the title is always in the center.

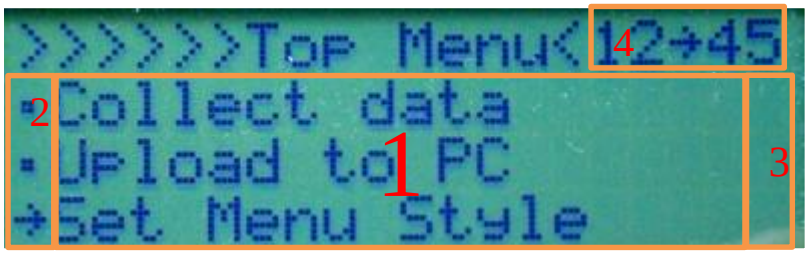
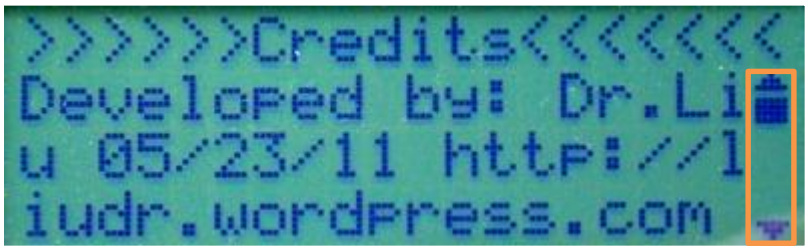
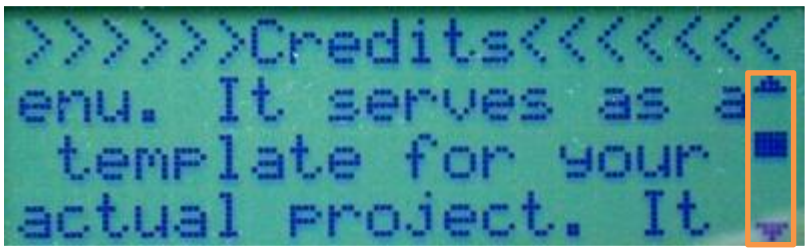


Figure 2-2. Menu/list example. 1. Main menu/list area displays 3 items in one column (can display any rows and columns combination). 2. Indicator of the highlighted item is the bottom one. Auto center is disabled so the menu/list is displayed page after page. Pressing up/down moves the arrow to the next item. 3. Scroll bar is disabled, saving one column. 4. Index style 1 tells the current item and a list of all items 1 thru 5. The right arrow indicates the highlighted item. The menu/list title is printed with center_text() so that the title is always in the center.

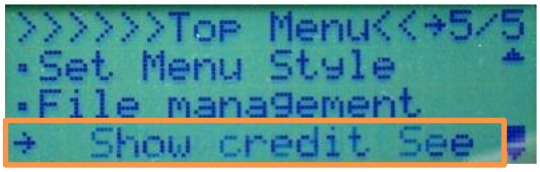


(a)

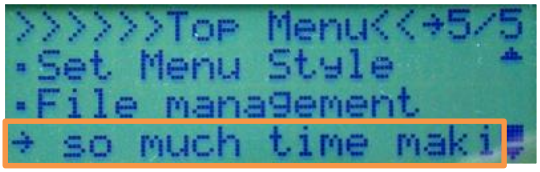


(b)

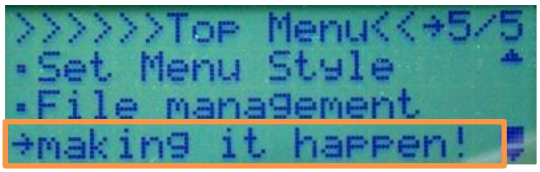
Figure 2-3. Example text area. (a) The text area occupies the 2nd to the 4th rows and all columns with a scroll bar on the right side. (b) The user can scroll up and down to see the entire text with up and down buttons. The orange boxes enclose the automatically updated scroll bar as the user scrolls the text. The "credits" was printed with center_text() so it appears in the middle of the display.



(a)



(b)



(c)

Figure 2-4. Scrolling item. The last item on the menu/list is too long for the limited space. If auto scroll is enabled, the item will automatically scroll to show the entire list. The (a), (b), and (c) show the entire item "Show credit See who has spent so much time making it happen!" The orange boxes show the scrolling item.

3. Setting up the library

1) Install the library

The phi_prompt library requires Arduino IDE 1.0 or up. Please download the latest version of Arduino IDE. The phi_prompt library also relies on phi_interfaces library. They are often released in one compressed file.
To install the library:

1. Unzip all files in a temporary folder
2. Cut the phi_prompt folder from this temporary folder and paste it inside your Arduino sketchbooks\libraries\ folder so you will have .\sketchbooks\libraries\phi_prompt\ folder and phi_prompt.cpp etc resides directly in this folder. Do not allow duplicate folder such as .\sketchbooks\libraries\phi_prompt\ phi_prompt\.
3. Cut the phi_interfaces folder from this temporary folder and paste it inside your Arduino sketchbooks\libraries\ folder so you will have .\sketchbooks\libraries\phi_ interfaces\ folder and phi_ interfaces.cpp etc resides directly in this folder. Do not allow duplicate folder such as .\sketchbooks\libraries\phi_ interfaces\ phi_ interfaces\.

Please refer to the following snapshot (windows):

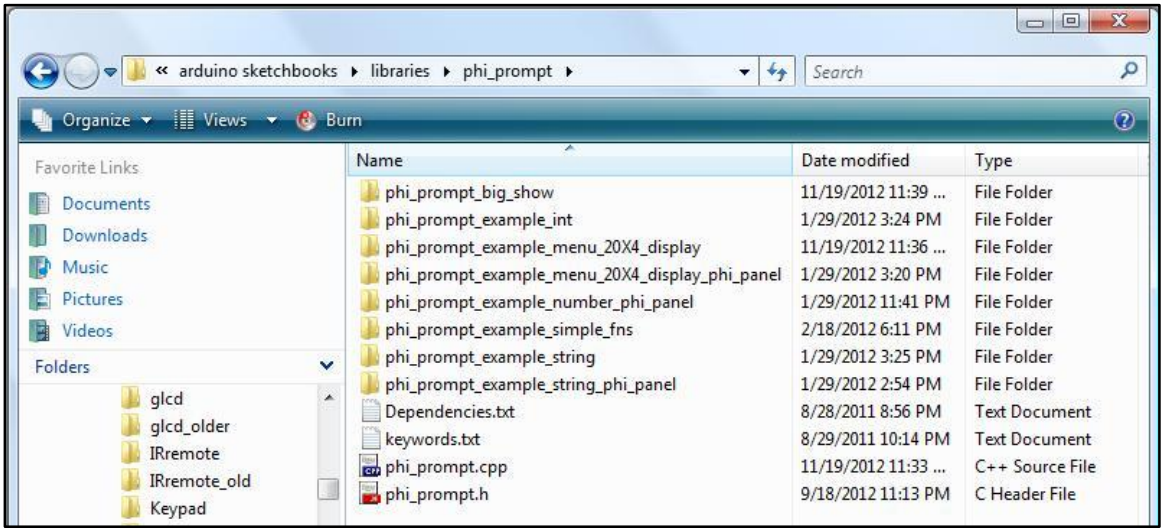


Fig. 3-1 Folder for phi_prompt

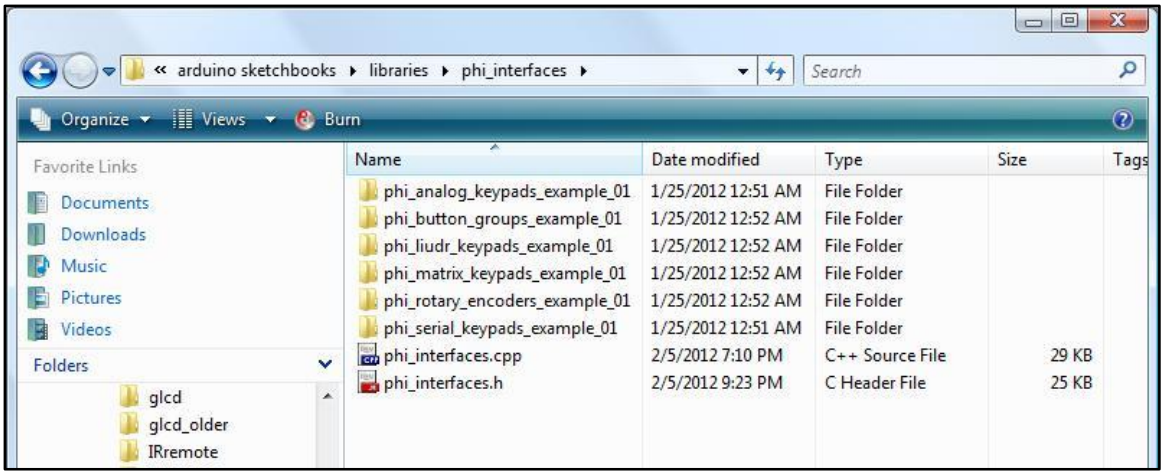


Fig. 3-2 Folder for phi_interfaces

2) Set up code in Arduino

In order for the library to function properly, it needs the following for initialization:

- ☆ The LCD object to display to and its dimensions
- ☆ Set up input devices
- ☆ Establish a list of these input devices
- ☆ Establish a list of keys for function keys such as up/down/left/right/enter/escape

There are two ways to do this setup, simple way (not much freedom but quick), and standard way (full freedom but requires more coding).

Simple setup

Three simple setup functions are provided with this release, one that sets up the library with 6 push buttons, and the other two that set up the library for phi-2 shields and phi-3 shields.

For LCD and 6 push buttons:

```
LiquidCrystal lcd(LCD_RS,LCD_EN,LCD_D4,LCD_D5,LCD_D6,LCD_D7);
void setup()
{
simple_setup_6_buttons(&lcd, columns, rows, btn_u, btn_d, btn_l, btn_r, btn_b, btn_a);
}
```

Simply replace the place holders for the lcd pins, then provide lcd columns, and rows (say 16, and 2) and digital pins to u/d/l/r/b/a. Now you are ready to call the library.

For phi-2 shields:

```
void setup()
{
        simple_setup_phi_2();
}
```

Simply including the above line in your project, then you are ready to call the library.

For phi-3 shields (to be released):

```
void setup()
{
        simple_setup_phi_3();
}
```

Simply including the above line in your project, then you are ready to call the library.

Standard setup

This way of setup gives you all the freedom with the choice of buttons, rotary encoders, etc. To prepare for library initialization, you need the following code before the setup(), which applies to phi-2 shields:

```
#define LCD_RS 8              ///< Arduino pin connected to LCD RS pin
#define LCD_EN 9              ///< Arduino pin connected to LCD EN pin
#define LCD_D4 7              ///< Arduino pin connected to LCD D4 pin
#define LCD_D5 6              ///< Arduino pin connected to LCD D5 pin
#define LCD_D6 2              ///< Arduino pin connected to LCD D6 pin
#define LCD_D7 3              ///< Arduino pin connected to LCD D7 pin
#define lcd_rows 2            ///< Specify the height of the LCD.
#define lcd_columns 16        ///< Specify the width of the LCD.

LiquidCrystal lcd(LCD_RS,LCD_EN,LCD_D4,LCD_D5,LCD_D6,LCD_D7);

#define total_buttons 6       ///< The total number of push buttons in a buttons group
#define btn_u 5               ///< I/O pin for a button
#define btn_d 10              ///< I/O pin for a button
#define btn_l 11              ///< I/O pin for a button
#define btn_r 4               ///< I/O pin for a button
#define btn_b A0              ///< I/O pin for a button
#define btn_a A1              ///< I/O pin for a button

byte pins[]={btn_u,btn_d,btn_l,btn_r,btn_b,btn_a};  ///< The digital pins for buttons
char mapping[]={'U','D','L','R','B','A'};          ///< This is a list of names for each button.

// The following lines instantiate several keypads.
```

```
phi_button_groups* my_btns= new phi_button_groups(mapping, pins, total_buttons);
///< This instantiates a button group to sense the buttons.

// The following adds all available keypads as inputs for phi_prompt library
multiple_button_input * keypads[]={my_btns, 0}; ///< Phi-2 shield keypad, 0 termination.

// The following sets up function keys for phi_prompt library
char up_keys[]={"U"};          ///< All keys that act as up. Must use double quotations.
char down_keys[]={"D"};        ///< All keys that act as down. Must use double quotations.
char left_keys[]={"L"};        ///<All keys that act as left. Must use double quotations.
char right_keys[]={"R"};       ///< All keys that act as right. Must use double quotations.
char ent_keys[]={"B"};         ///< All keys that act as enter. Must use double quotations.
char esc_keys[]={"A"};         ///< All keys that act as escape. Must use double quotations.
char * function_keys[]={up_keys,down_keys,left_keys,right_keys,ent_keys,esc_keys};

void setup()
{
        init_phi_prompt(&lcd, keypads, function_keys, lcd_columns, lcd_rows, '~');
}
```

As you can see, there are five sections color coded.

The purple section has definitions of the LCD pins, dimension, and the instantiate an LCD object. This is done by supplying all the pins that are connected to the LCD in the correct order. There is no need to change this section once set.

The orange section establishes a phi_button_groups object with information such as where the buttons are connected, what character each button press returns and the actual code to instantiate the phi_button_groups object *my_btns*. There is no need to change this section once set. You may add more input devices here such as rotary encoders etc.

The green section is an array that contains all input devices. You provide the address of *my_btns* to it so the object can later be sensed by the library for inputs. In case you have other input devices such as a rotary encoder, you include the address of that object in this array. The array must terminate with 0.

The blue section is a series of arrays that set up what key presses represent which function keys such as up/down/left/right/enter/escape. If phi_prompt sees these key presses, it does special functions depending on what it is displaying. Say if you pressed a button that corresponds to up, in a menu, then you move your highlight to one item up. These keys should correspond to the return values generated from actual button pushes. Say if one button push returns 'U' and you can include 'U' in the up_keys so that any button that generates 'U' is treated as the up function key and produces the desired result of an up key press. This 'U' will not be received by the caller but translated to phi_prompt_up and returned to the caller (most of the time the wait_on_escape function).

The red section is the initialization. This initialization function supplies phi_prompt library the liquid crystal object *&lcd*, input keypads *keypads*, and function key names *function_keys*. Also supply the column and row of the lcd *lcd_columns*, *lcd_rows*, and indicator as '>'. You can also use '\x7e', which is a right arrow. When all of the above sections are present, you may call the library initialization in setup():

Once set up, you may use the library any time. In the following sections, you will learn how to invoke the library, first with simple functions, then with standard functions.

4. Using simple functions

This section is for you if you are just starting off with programming or if you want to enjoy simple ways to program.

1) Display a menu:

This might be your most desired feature at the simplest possible way. Say you want to display day of the week in a menu, all you need to do, once you initialized the library is this:

response=simple_select_list("Day of week:\nSUN\nMON\nTUE\nWED\nTHU\nFRI\nSAT\n");

The rendering result will auto scale on 16x2 and 20x4 display as the following:

(a)

(b)

Fig. 4-1 Screen shoot of phi_prompt simple list. a) is the result on a 16*2 display. b) is the result of the same command on a 20*4 display.

Each entry is separated by a '\n'. The first entry is considered the title of the menu. The rest are menu entries. There is also a '\n' in the end to terminate the command.

This menu is displayed without the developer ever worried with the LCD size or else. It occupies the entire LCD screen and usually forms multiple columns of choices as needed. The return value is 0-6 representing the seven days of the week. Notice the item number is zero based. If the return value is 255, the user has pressed escape. **There is a 32 item maximum for this simple function. You may modify the library to make it larger but please help yourself with debugging as well.**

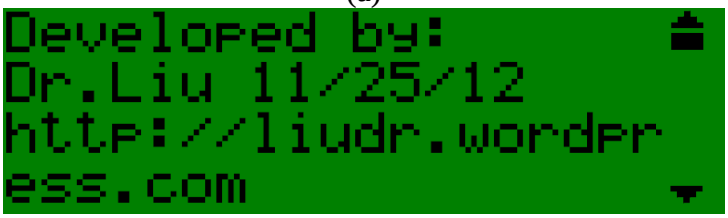
If you know of any easier way to create an interactive menu, you should inform me ☺

2) Display a long scrollable text area

If you want to display an introduction that is longer than a few lines, which requires the user to scroll up and down to read, and then dismiss, this is for you:

simple_text_area("Developed by:\nDr.Liu 11/25/12\nhttp://liudr.wordpress.com\nThis is an example.\nPress Confirm to continue");

(a)



(b)

Fig. 4-2 Screen shoot of phi_prompt simple text area. a) is the result on a 16*2 display. b) is the result of the same command on a 20*4 display.

This function is blocking, that is, trapping Arduino execution until the user presses a key to dismiss the message. While blocking, the function scrolls the message if the user presses up/down/left/right buttons to scroll up/down one line or one page. Notice this message is too long for even 20*4 display. Also notice the '\n' starts a new line. You can easily organize the message with new lines. In this message you don't have worry about how the message fits on your screen. The function automatically uses the size of your screen to display the message. So if you change to a 16*2 display you still see this message displayed correctly, except on a smaller window. The user is able to use up/down to scroll the message to read the entire message. A scroll bar is also displayed on the right side.

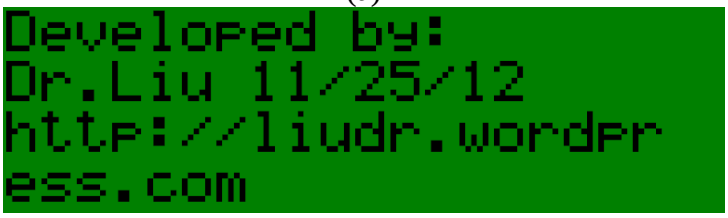
3) Display a long formatted message

If you just want to render a long multi-line message and immediately return control to Arduino, this is for you. This function is thus non-blocking. Except for not being able to sense buttons or displaying scroll bar, the rendering result is the same. See example below:

simple_formatted_msg("Developed by:\nDr.Liu 11/25/12\nhttp://liudr.wordpress.com\nThis is an example.\nPress Confirm to continue");

Notice that the message is too long to display on one screen and since this function doesn't scroll, only the first portion is displayed.

(a)



(b)

Fig. 4-2 Screen shoot of phi_prompt simple formatted message. a) is the result on a 16*2 display. b) is the result of the same command on a 20*4 display.

4) Get input from the user numbers or texts

At some point in your program you will need to ask user for numeric parameters or maybe a name. Here is how to do it the simple way:

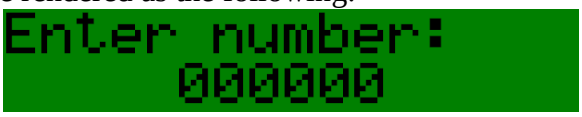
```
char msg[]="Hello World";
simple_input_panel(msg, 'A', 'z', 1)
```

This simple input panel will first display the content of msg and move the cursor to the first character. The user may edit the content by up/down keys and navigate with left/right keys. If you have a numeric or serial keypad the user can enter text or numbers by directly pushing the keys. Pressing enter or going beyond the end of the input are both treated as confirm. The returned text is contained in msg so the original msg is modified, be warned. If you include a numeric msg such as char msg[]="000000"; then you can use it to ask user for numbers. The second and third parameters are the allowed lowest and highest characters. Having 'A' and 'z' means each character is allowed to change between A and z. The last parameter is option, 0 for default and 1 to include 0-9 as valid inputs even if they are not in the allowed lowest or highest characters.

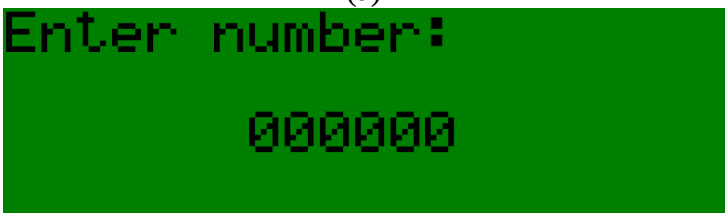
If you want a prompt for the input, you may clear screen and print the prompt on row 0. The simple input panel prints on row 1 for 2-row display or row 2 for 4-row display. It won't call lcd.clear so your prompt will be shown during the entire entry. The function is blocking until the user confirms the entry.

```
char msg[]="000000";
lcd.clear();
lcd.print("Enter number:");
simple_input_panel(msg, '0', '9', 1)
```

The above code will be rendered as the following:



(a)



(b)

Fig. 4-3 Screen shoot of phi_prompt simple input panel. a) is the result on a 16*2 display. b) is the result of the same command on a 20*4 display.

5) More simple functions

Interested in more control over these simple functions, such as adding scroll bar to a list, centering choice of a list, auto scrolling list items that are too long etc.? The following functions will do these tricks for you and require not much explanation. Use 1 for enabling and 0 for disabling. **Make sure you call before you call the simple functions.**

```
void simple_select_list_center_choice(byte b);      ///< Enable or disable center_select list choice.
void simple_select_list_scroll_bar(byte b);         ///< Enable or disable scroll bar.
void simple_select_list_auto_scroll(byte b);        ///< Enable or disable auto scroll long item.
```

The following are useful functions to extract the title and individual items of a list. It is useful to be able to print a message such as "You have chosen slow and painful", where slow and painful is a list entry. The function get_simple_list_item gets the list entry and returns it in buf. Same goes for get_ simple_list_title.

```
byte get_simple_list_title(char msg[], char buf[]);     ///< Extracts the title from a simple list msg. The list has to observe the same format as the list for simple_select_list: "Menu_title\nItem1\nItem2\nLast_item\n".
byte get_simple_list_item(char msg[], char buf[], byte item_num);       ///< Extracts the "item_num" item from a simple list msg. The list has to observe the same format as the list for simple_select_list: "Menu_title\nItem1\nItem2\nLast_item\n".
```

6) Even more simple functions

Wondering what some undocumented simple functions (what? Everything is well-documented in the source) can do for you? Some of these functions are servicing the simple functions you call and aren't useful to your simple quest for interactivity. Some other functions are written specifically to allow mulit-select menu, dynamic menu contents and in-menu-adjust features. These are advanced features, which you can sneak peak here:

http://liudr.wordpress.com/2012/04/15/phi_prompt-advanced-features-video-demo/

There is no simple nature to these dynamic features and none of them can be written into a library call so these features are reserved for those experienced with programming and willing to purchase a license of "Phi_prompt User Interface Professional Development Kit". Here is a link:

http://liudr.wordpress.com/libraries/phi_prompt-ui-pdk/

Professional programming help and support is available to these license holders and Arduino fans in general.

http://www.inmojo.com/store/liudr-arduino-and-physics-gadgets/item/software-help/
Here is a static representation of the dynamic features of the phi_prompt UI PDK:

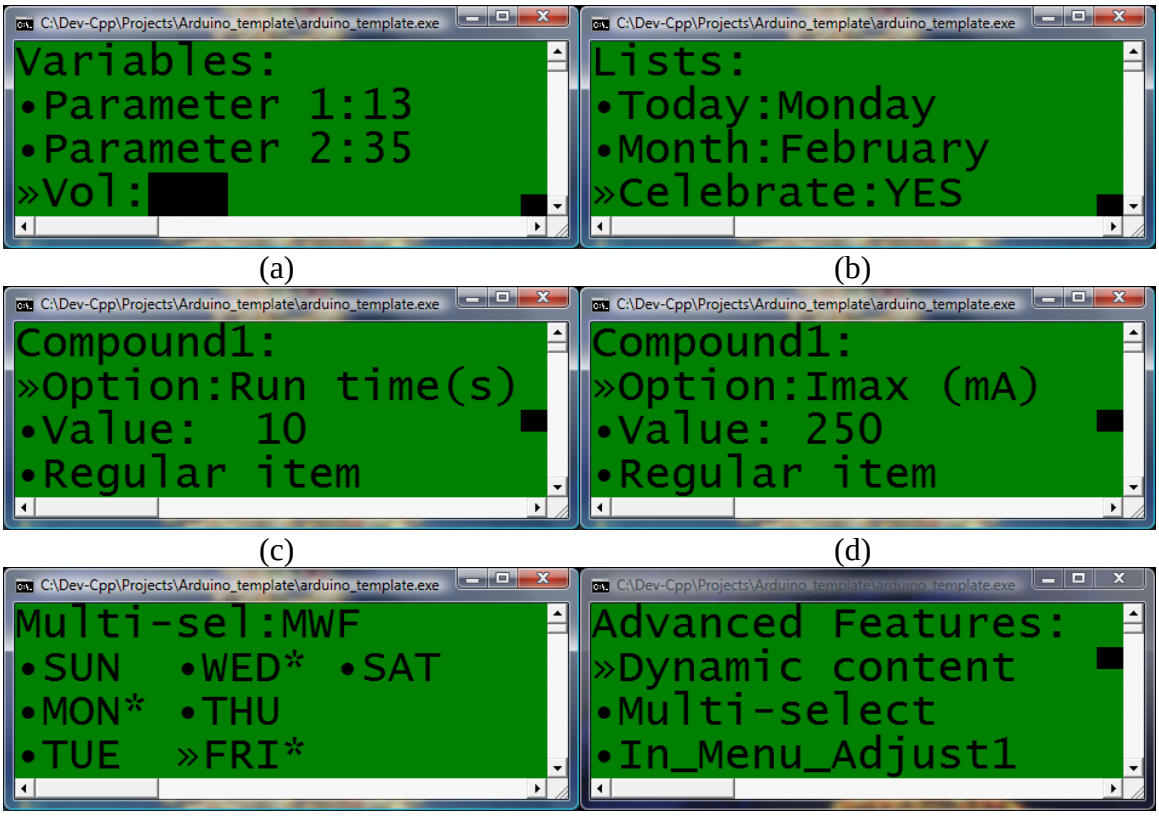


Fig. 4-4 Dynamic features of phi_prompt UI PDK. a) Lines 2,3 and 4 feature in-menu-adjust numeric parameters. b) Lines 2,3 and 4 feature in-menu-adjust lists. (c) Compound in-menu-adjust. (d) Compound in-menu-adjust next parameter. (e) Multi-select list with selection indicator. (f) Demo menu.

a) Lines 2,3 and 4 feature in-menu-adjust numeric parameters. Pressing left/right while the item is highlighted will adjust, say, parameter 1 from 0 to 13 or parameter 2 from 0 to 35, or Vol from 0 to 15 with an analog bar graph indicating volume. You don't have to select the item and then adjust, just highlight it and adjust while you are in the menu.

b) Lines 2,3 and 4 feature in-menu-adjust lists. Again, move cursor to highlight an item and press left/right to cycle through each list, day of week, month, or yes/no.

(c) Compound in-menu-adjust. Here Option line is in-menu-adjust and currently takes the value of Run time(s). Correspondingly the value line displays the value of Run time(s). You may adjust this value if you want.

(d) Compound in-menu-adjust next parameter. Here the user has cycled to the next parameter on option line and now the parameter is Imax(mA). The value line now indicates Imax(mA) value. You may adjust this value if you want.

(e) Multi-select list with selection indicator. Every time you select an item, it is marked with a star on the right. Besides that, a condensed list of the day of the week is also dynamically displayed, as MWF at the moment. It will change into Weekdays and Daily, or Weekends if these conditions are met.

(f) Demo menu. Just the main menu of the demonstration.

5. Using standard functions

Besides simple functions, the library features a full arsenal of standard full-featured functions. The following are example codes for various tasks you may find yourself up against in your projects. All the code snippets require proper initialization codes and to reside in the loop(). The common initialization code is the following:

```
#define lcd_rows 2
#define lcd_columns 16

//Phi-2 shield buttons
#define btn_u 5
#define btn_d 10
#define btn_l 11
#define btn_r 4
#define btn_b 14
#define btn_a 15

//Phi-2 shield LCD pin setting
#define LCD_RS 8
#define LCD_EN 9
#define LCD_D4 7
#define LCD_D5 6
#define LCD_D6 2
#define LCD_D7 3

#include <LiquidCrystal.h>
#include <WProgram.h>
#include <phi_buttons.h>
#include <phi_prompt.h>

phi_buttons btn_1(btn_u, LOW);
phi_buttons btn_2(btn_d, LOW);
phi_buttons btn_3(btn_l, LOW);
phi_buttons btn_4(btn_r, LOW);
phi_buttons btn_5(btn_b, LOW);
phi_buttons btn_6(btn_a, LOW);

LiquidCrystal lcd(LCD_RS,LCD_EN,LCD_D4,LCD_D5,LCD_D6,LCD_D7); // Create
the lcd object

phi_buttons *btns[]={&btn_1,&btn_2,&btn_3,&btn_4,&btn_5,&btn_6}; // Actual
buttons definitions are in the defs.h

void setup()
{
```

```
  lcd.begin(lcd_columns, lcd_rows);
  init_phi_prompt(&lcd,btns,lcd_columns, lcd_rows, '~'); // Supply the liquid crystal
object and the phi_buttons objects. Also supply the column and row of the lcd, and
indicator as '>'. You can also use '\x7e', which is a right arrow.
}
```

The above code properly initializes the LCD, all buttons, and the phi_prompt library. You will need to download phi_buttons library (http://liudr.wordpress.com/phi_buttons/). You are strongly suggested to obtain a phi-2 shield (http://liudr.wordpress.com/phi-2-shield/) to run the code. If you intend to run the code on your own hardware instead, please change the buttons and LCD pin definitions to suit your hardware. In case you don't have 6 push buttons, you may use *btn_null* in place of the actual pins to declare a null button, which is a dummy. You can afford to assign null to button 2, which is down button, then button 6 the escape button, then button 5, the enter button. This way you can operate with 3 buttons only, using the right button as enter. Since all lists and inputs wrap around, the up button is enough.

1) Input an integer

In lots of projects, you would want to interact with Arduino on the fly or in the field by entering numbers to Arduino, say motor speed, angle, Morse code speed etc. You can use the input_integer() function. Here is an example: You want a number between 0 and 20 with increment of 2 (basically 0,2,4,6,8, etc.)

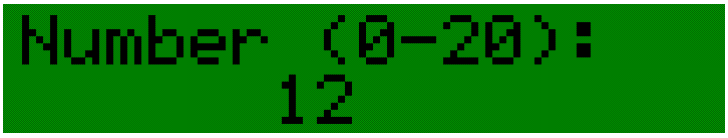


Figure 4-1. Sample screen for inputting a number between 0 and 20.

```
int user_input=10; // This is the storage for the integer
phi_prompt_struct myIntegerInput; // This struct stores information for library functions
myIntegerInput.ptr.i_buffer=&user_input; // Pass the address of user_input to the library.
After library function call, user input will be stored in this variable. Note the use of "&".
myIntegerInput.low.i=0; // Lower limit. The number wraps to 20 when decreased from 0.
myIntegerInput.high.i=20; // Upper limit. The number wraps to 0 when increased from
20.
myIntegerInput.step.i=2; // Step size. You will get 0, 2, 4, 6, 8, etc if you set it to 2.
myIntegerInput.col=7; // Display the number at column 7
myIntegerInput.row=1; // Display the number at row 1
myIntegerInput.width=2; // The number occupies 2 character space.
myIntegerInput.option=0; // Option 0, space pad right, 1, zero pad left, 2, space pad left.
lcd.clear(); // Clear the lcd.
lcd.print("Number(0-20):"); // Prompt user for input
input_integer(&myIntegerInput); // This calls the library function. The initial number will
be displayed first and the functions waits for the user to press up/down to change the
number and enter to confirm, after which it stores the new number in user_input. Notice
the "&" in front of the myIntegerInput struct.
```

After the above code is executed, the user's final choice of the number resides in the user_input.

2) Text input with text panel

In some projects, you may want to have a text panel where the user types up a short string for file names, password etc. You can use the input_panel() to get user input with various options on what characters are allowed.

Figure 4-2. Sample screen of input panel.

```
phi_prompt_struct myTextInput; // This struct stores information for library functions.
char file_name[]="AAAAAAAA.TXT"; // This buffer stores the content of the text
panel.
myTextInput.ptr.msg=file_name; // Assign the text buffer address
myTextInput.low.c='A'; // Text panel valid input starts with character 'A'.
myTextInput.high.c='Z'; // Text panel valid input ends with character 'Z'.
myTextInput.width=12; // Length of the input panel is 12 characters.
myTextInput.col=2; // Display input panel at column 2
myTextInput.row=1; // Display input panel at row 1
myTextInput.option=1; // Option 1 includes 0-9 as valid characters. Option 0, default.
lcd.clear(); // Clear the lcd
lcd.print("File name:"); // Prompt user for input
input_panel(&myTextInput); // User input is stored in file_name. Notice the "&".
```

The user text input is stored in file_name after the function returns.

3) Floating number inputs

Sometimes you need to input some floating point numbers with input_float().I wrote the floating number input from text input panel, talking about reusing codes. The floating number input allows you to enter a number with a FIXED number of digits before and after the decimal point. You may restrict the input to be positive, negative, or both. There is no step or min/max for floating number input. I found it simply too much work to implement such limits, which may or may not be useful.

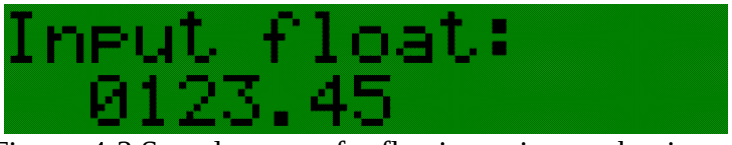


Figure. 4-3 Sample screen for floating point number input.

```
  phi_prompt_struct myFloatInput; // This struct stores information for library functions.
  float myFloatNumber=123.45; // Initial value. User input will be stored here afterward.
  myFloatInput.ptr.f_buffer=&myFloatNumber; // Notice the “&”.
  myFloatInput.step.c_arr[0]=2; // 2 digits after decimal
  myFloatInput.step.c_arr[1]=4; // 4 digits before decimal
  myFloatInput.col=2; // Display input panel at column 2
  myFloatInput.row=1; // Display input panel at row 1
  myFloatInput.option=2; // 0: only positive numbers allowed, 1: only negative numbers
allowed, 2: both positive and negative numbers are allowed.
  lcd.clear(); // Clear the lcd
  lcd.print("Input float:"); // Prompt user for input
  input_float(&myFloatInput); // Notice the “&”.
```

The above code initializes the floating point input and stores the user input in variable myFloatNumber. The number of digits before decimal includes the negative sign so if you want 3-digit negative numbers you will allow “4 digits before decimal”.

4) Interactive lists

Interactive lists can be made with select_list().

Figure 4-4. Sample screen of a 2 by 2 list with arrow/dot enabled.

```
//First, define the complete list in PROGMEM to save RAM. Please follow the example
closely.
PROGMEM prog_char pasta_00[]="Spaghetti";
PROGMEM prog_char pasta_01[]="Rotelle";
PROGMEM prog_char pasta_02[]="Rotini";
PROGMEM prog_char pasta_03[]="Fettuccine";
PROGMEM prog_char pasta_04[]="Lasagne";
PROGMEM prog_char pasta_05[]="Penne";
PROGMEM prog_char pasta_06[]="Rigatoni";
PROGMEM const char *pasta_items[]=
{pasta_00,pasta_01,pasta_02,pasta_03,pasta_04,pasta_05,pasta_06};

int user_choice=0; // This will store the item number (0-6) the user chooses.
phi_prompt_struct myListInput; // This struct stores information for library functions.
myListInput.ptr.list=(char**)&pasta_items; // Please follow closely the syntax..
myListInput.low.i=0; // Default item highlighted on the list is #0, the first item on the list.
myListInput.high.i=6; // Last item on the list is size of the list - 1.
myListInput.width=7; // Width of each list item. Longer items are truncated or scrolled.
```

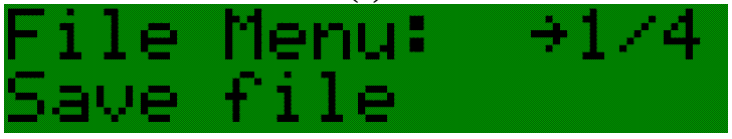
```
myListInput.col=0; // Display the list at column 0
myListInput.row=0; // Display the list at row 0
myListInput.step.c_arr[0]=2; // Display the list as two rows
myListInput.step.c_arr[1]=2; // Display the list as two columns
myListInput.option=1; // See render_list() for details of how to set this option.

PROGMEM prog_char pasta_00[]="Spaghetti";
PROGMEM prog_char pasta_01[]="Rotelle";
PROGMEM prog_char pasta_02[]="Rotini";
PROGMEM prog_char pasta_03[]="Fettuccine";
PROGMEM prog_char pasta_04[]="Lasagne";
PROGMEM prog_char pasta_05[]="Penne";
PROGMEM prog_char pasta_06[]="Rigatoni";
PROGMEM const char *pasta_items[]=
{pasta_00,pasta_01,pasta_02,pasta_03,pasta_04,pasta_05,pasta_06};
```

Notice that select_list() doesn't know the address of user_choice thus it stores the user choice in low.i for you to retrieve. The struct's only pointer is used to point to the list items. There are many options you can choose to make your list very nice. Lists with shorter items, such as day of the week, would benefit from multiple-column list, while lists with many items may use an index display. Lists with long items may enable scroll to display the entire items. You also may center your choice on the list so the list moves but the highlighted item is always in the middle. This works nicely on a larger display. All options are listed under render_list().

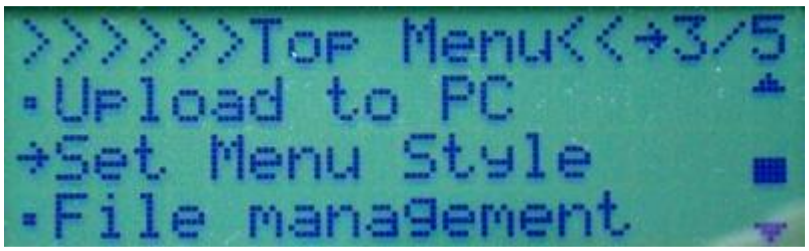
5) Multi-level menu

Nothing compares with a nice interactive menu that will make your project just perfect as a stand-alone project that can take a user through various functions under layers of menus like an MP3 player. A menu also helps organize your code so they don't tangle together like a bowl of pasta when improperly cooked. You can construct a menu with any number of levels or layers with select_list(). With a 20X4 display, you can really take advantage of all the rendering options the library provides.

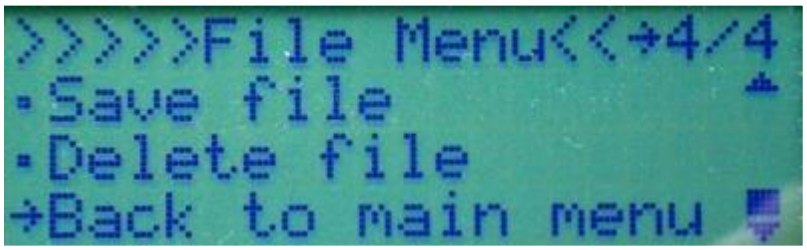
(a)



(b)



(c)



(d)

Figure 4-5. Sample screens of (a) top menu and (b) a sub menu on a 16X2 display and screen shots of (c) top menu and (d) a sub menu on a 20X4 display.

This example is more extensive. **The complete code is provided with the library release as phi_prompt_example_menu either 16X2 or 20X4.** It is a complete template for an interactive data logger. The logger has one main menu with the following items:

1. Collect data
2. Upload to PC
3. Set menu style
4. File management (sub menu)
5. Show credit

The sub menu has the following items:

1. Load file
2. Save file
3. Delete file
4. Main menu (return to main menu)

There can be as many level of menus as you like, as long as you follow the example.

<u>This is the main menu:</u>

```
int menu_pointer_1=0; // This stores the menu choice the user made.
phi_prompt_struct myMenu; // This structure stores the main menu.
select_list(&myMenu); // Use the select_list to ask the user to select an item of the list.
menu_pointer_1=myMenu.low.i; // Get the item number selected by the user.
switch (menu_pointer_1) // See which menu function to run
```

```
{

  case 0:
  top_menu_function_1();
  break;

  case 1:
  top_menu_function_2();
  break;

  case 2:
  top_menu_function_3(&myMenu); // This is the setting function that changes how
  menu is rendered
  break;

  case 3:
  sub_menu_1(); // This is actually the sub menu.
  break;

  case 4:
  top_menu_function_5(); // If this were a sub level menu, this last one should return to
  previous level by using "return".
  break;

}
```

This is the sub menu:

```
int menu_pointer_1=0;
phi_prompt_struct myMenu; // This structure stores the sub menu.
select_list(&myMenu);
menu_pointer_1=myMenu.low.i;
switch (menu_pointer_1)
{

    case 0:
    sub_menu_1_function_1();
    break;

    case 1:
    sub_menu_1_function_2();
    break;

    case 2:
    sub_menu_1_function_3();
    break;
```

```
    case 3: // This option returns to previous level.
    lcd.clear();
    return; // This returns to previous menu level. Only sub menus should have this unless
    the top level menu also returns to something.
    break;

}
```

6) Text area

A lot of times you will need to display long messages such as introduction, instruction, and credit to a user on limited display space. The easiest way to display long messages is to use a text area. The text area automatically wraps text and lets user scroll text up and down. Two versions exist, one displays long messages in SRAM, one displays long messages in PROGMEM. There is an example:

```
PROGMEM prog_char long_msg_p[]="Developed by: Dr.Liu 05/23/11
http://liudr.wordpress.com This is just a mock-up of an actual data acquisition system ";
phi_prompt_struct myLongMsg;
myLongMsg.ptr.msg=long_msg_p; // Assign the address of the text string to the pointer.
myLongMsg.low.i=0; // Text starts at position 0.
myLongMsg.high.i=strlen_P(long_msg_p);
myLongMsg.step.c_arr[0]=3; // Text area spans 3 rows
myLongMsg.step.c_arr[1]=19; // Text area spans 19 columns
myLongMsg.col=0; // Display the text area starting at column 0
myLongMsg.row=1; // Display the text area starting at row 1
myLongMsg.option=1; // Option 0, no scroll bar, option 1, scroll bar on right.
text_area_P(&myLongMsg);
```

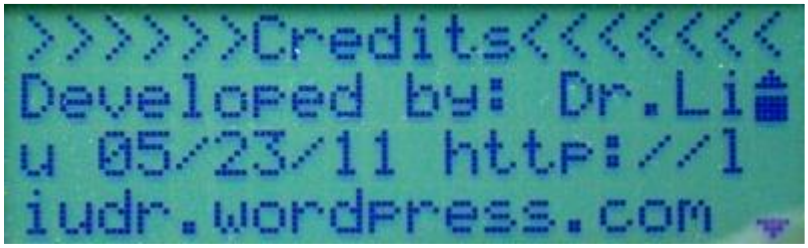


Figure 4-6. Text area occupying 19 columns and 3 rows. The top line doesn't belong to the text area.

6. Library struct

The struct phi_prompt_struct is used to pass parameters to library functions in order to tell the library function where to display what with which options. It is defined as the following:

```
union buffer_pointer
{
  int *i_buffer;
  float * f_buffer;
  char ** list;
  char* msg;
  PGM_P msg_P;
};

union four_bytes
{
  int i;
  long l;
  float f;
  byte b;
  char c;
  char c_arr[4];
};

struct phi_prompt_struct
{
 buffer_pointer ptr;
 four_bytes low;
 four_bytes high;
 four_bytes step;
 byte col;
 byte row;
 byte width;
 int option;
 void (*update_function)(phi_prompt_struct *);
};
```

The two unions are defined to save memory. You don't have to be concerned with how to use them. They're not used outside the struct.

1) buffer_pointer ptr

This is the pointer to various things, such as strings to display in text areas, or list index, or integer value for an integer entry. If this is for number entries, such as floating point numbers, the returned value is stored in the address it points to.

If you intend to use the struct to call input_float, assign the address of the floating point number to ptr.f_buffer. Eg.

float x=1.5;

```
phi_prompt_struct floatInput;
floatInput.ptr.f_buffer=&x;
```

Note the “&” is necessary to extract the memory address of x.

If you want to display an interactive list, you should do this:

```
PROGMEM prog_char pasta_00[]="Spaghetti";
PROGMEM prog_char pasta_01[]="Rotelle";
PROGMEM prog_char pasta_02[]="Rotini";
PROGMEM prog_char pasta_03[]="Fettuccine";
PROGMEM prog_char pasta_04[]="Lasagne";
PROGMEM prog_char pasta_05[]="Penne";
PROGMEM prog_char pasta_06[]="Rigatoni";
PROGMEM const char *pasta_items[]=
{pasta_00,pasta_01,pasta_02,pasta_03,pasta_04,pasta_05,pasta_06};

 phi_prompt_struct myListInput;

 myListInput.ptr.list=(char**)&pasta_items;
```

Note the “&” is necessary to extract the memory address of pasta_items.

2) four_bytes low

This normally stores the lower limit of an input entry of the default list/menu item when the list/menu is first displayed. It is also used to store the user’s final list/menu choice. In the following examples, assume a struct called myStruct is already defined.

You will store lower limit for text panel’s allowed character.
Eg. myStruct.low.c=’A’;
It sets the lower limit of allowed character to ‘A’.

You will store lower limit for integer entry.
Eg. myStruct.low.i=5;
It sets lower limit of an integer entry to be 5.

You will store the index of the initial highlighted item for your list.
Eg. myStruct.low.i=0;
It sets the first or zeroth list/menu item on highlight. Every time up/down is pressed, this number changes and the list/menu is repainted. When the user has made a decision, this value contains the item’s index the user finally decides.

Eg.

```
 if (select_list(&myStruct)!=-1) user_choice= myStruct.low.i;
```

If the user didn't press escape (return -1) then the user choice is stored in myStruct.low.i so we can assign its value to user_choice, which we will use to make decisions.

You will store the index of the first character to be displayed in your text area.
Eg.
myStruct.low.i=80;
text_area(&myStruct);

It sets the first character to be displayed in the text area to be the 80th characters so the first 80 characters don't get displayed. If you have a long string, you can display parts of the string by manipulating the value low.i.

3) four_bytes high;

This value stores upper limit for text panel (.c), integers (.i) and floats (.f) and last item for list (.i). Note that if you have 5 items in a list, the last is the 4th, counting from zero.

You will store upper limit for text panel's allowed character.
Eg. myStruct.high.c='Z';
It sets the upper limit of allowed character to 'Z'. With the previous command myStruct.low.c='A'; you will have an input panel that has valid inputs of capitalized letters.

4) four_bytes step;

This determines the increment of various inputs. Say if you want an integer input that increments 5 every time the user presses up, you can do : myStruct.step.i=5;
Same goes with input panel but it is most appropriate to set them to 1.
This field is also used by the list renderer render_list(). Since a list should always have increment of 1, this field is split in four bytes, each storing things other than increment.

```
myStruct.step.c_arr[0]=1;//rows
myStruct.step.c_arr[1]=2;//cols
myStruct.step.c_arr[2]=0;//y2
myStruct.step.c_arr[3]=12;//x2
```

The rows and columns store the size in row and column of the list so if you want a list with two columns and two rows like the following, you want them to be both 2.

Figure 5-1. A two row by two column list.

The x2 and y2 are coordinates for additional features used in list rendering. The list index currently uses this coordinates so if you specify 12 column and 0 row, you will get an index at that location. See render_list() for more information.

5) byte col

This is the location (column) of the item to be displayed, be it integer entry or list.

6) byte row

This is the location (row) of the item to be displayed, be it integer entry or list.

7) byte width

This indicates how many characters an integer or float or text entry needs. If you want your integers between 0 and 99, you need width to be 2 as there will be as many as 2 digits. If you want 0 to 100, then 3 for width. This also determines the width of each column of a list. If you have a 16X2 display and want to display a list that is 1 column and 2 rows with an arrow indicator in the front, then you are left with 15 columns (16-1 for arrow). You will set this to 15. If you want 2 columns and 2 row of list, then you should use 7 as width and enable the arrow indicator.

8) int option

This is the option for each library functions. You will find detailed explanation in each library function that uses this field.

*9) void (*update_function)(phi_prompt_struct *)*

This is not being used in this version but reserved for future releases. Essentially if you have an entry, you can use this function to make some updates while the user changes the value. Say if the user is changing the orientation of a motor with an integer entry, then every time the user changes the value with up/down keys, a function can be called to rotate the motor real time, instead of rotating it after the user finishes making a decision. If you have used an old version of MS Word you might know this. Pulling on the scroll bar doesn't change the page until you let go of your mouse. While in newer versions, if you pull the scroll bar, the page changes as you pull. The update function is going to do this in future releases.

7. List of simple library functions

The main attractions of the phi_prompt have been compatibility with a whole range of input devices, and having a collection of simple library functions. These simple functions get things done usually in one line of code for things that take 10 lines of code in typical situations.

Here is an example, see for yourself how easy and readable the code is. The code requires no explanation.

```
int choice=simple_select_list("Side dish?\nGreen beans\nCarrots\nAsparagus\n");
lcd.clear();
if (choice==0) simple_text_area("Sorry, but we are out of green beans.");
```

Without further delay, let's see what simple functions will do:

- One line of code renders menu/list, scrollable text or get user text input
- Return values are easy to use in response to user input
- Automatically scales message width and height to use the whole screen
- Simple function call to toggle scroll bar, auto scroll etc. for simple menu/list
- Possible multi-selection lists, dynamically append or modify the list or items
- Only limited number of menu/list items are allowed to save memory

Here is a list of the simple functions:

1) byte simple_select_list(char msg[],byte b=0,long l=-1)

Displays a menu/list from a formatted string. With no b option the list item 0 is highlighted. Returns the item selected.

msg This is the message to be displayed as menu/list. This message must be formatted as "Menu_title\nItem1\nItem2\nLast_item\n". The '\n' is new line.
The first line Menu_title will be displayed on line one of the display. The rest of the lines will be treated as menu/list items.

The user can scroll through the items while the title is always displayed on line one.

Warning: There is a maximal of 16 menu/items to this function. To use more items, you need to use the more feature-rich, flexible, but complex select_list().

It traps execution. It monitors the key pad input and returns if any key is pressed.
The function auto scales to fit on any LCD with size determined by phi_prompt initialization.

Example:
simple_select_list(msg) will display a select list with the 12 months on your display.

Please define char msg[]="Select month\nJAN\nFEB\nMAR\nAPR\nMAY\nJUN\nJUL\nAUG\nSEP\nOCT\nNOV\nDec\n".

It returns the item the user selected, the first item being number zero. If escape is pressed, it returns 255.

2) void simple_select_list_center_choice(byte b)

Enable (b=true) or disable (b=false) center_select list choice.

3) void simple_select_list_scroll_bar(byte b)

Enable (b=true) or disable (b=false) scroll bar.

4) void simple_select_list_auto_scroll(byte b)

Enable (b=true) or disable (b=false) auto scroll long item.

5) byte get_simple_list_title(char msg[], char buf[])

Extracts the title from a simple list msg. The list has to observe the same format as the list for simple_select_list: "Menu_title\nItem1\nItem2\nLast_item\n".

6) byte get_simple_list_item(char msg[], char buf[], byte item_num)

Extracts the "item_num" item from a simple list msg. The list has to observe the same format as the list for simple_select_list: "Menu_title\nItem1\nItem2\nLast_item\n".

msg This is the message of a menu/list. This message must be formatted as "Menu_title\nItem1\nItem2\nLast_item\n". The '\n' is new line.

buf This is the buffer to store extracted menu/list item. It must be long enough to hold the item to be extracted.

item_num This is the menu/list item to be extracted. It is zero-based.
If you display a simple list, chances are you will want to later display the selected item in text form. This function extracts that item.
Since the first item in a simple list is the list title, it skips it. To get the simple list title, call get_simple_list_title.
It returns whether the item is extracted, 0 for success, other values for failure. The extracted item is in buf.

7) byte simple_input_panel(char msg[],byte lo, byte h, byte opt)

Collect user input. Returns the user input in a string.
msg This is the message to be displayed as the initial content of the input panel, which you can modify. Warning: this string is used to store user input and is thus modified.

lo This is the lower limit of each character, when inputted with up/down. With lo='A', you can only enter characters with ASCII values equal or above 'A'. This only applies to inputs made with up/down keys.

h This is the upper limit of each character, when inputted with up/down. With h='Z', you can only enter characters with ASCII values equal or below 'Z'. This only applies to inputs made with up/down keys.

opt This is the option in addition to the upper and lower limits. Option 0, default, option 1 include 0-9 as valid inputs. This only applies to inputs made with up/down keys.

It traps execution. It monitors the key pad input and returns if any key is pressed.

The function auto scales to fit on any LCD with size determined by phi_prompt initialization.

Example:

simple_input_panel(msg, 'A', 'Z', 1) will have 6 digits for input and restricted within numbers 0-9 and letters A-Z on each character. Please define char msg[]="000000".

"000000" will set up initial content of a numeric input with up to 6 digits. To enter 123, just make the change with your keypad so the string reads "000123".

It returns the length of the string. The actual user input is already in the string msg.

8) byte simple_text_area(char msg[],byte b=0)

Displays a scrollable text area to fill the size of the screen.

msg This is the message to be displayed with formatting characters such as '\n' and '\t' in the message.

It traps execution until the user made a choice of '0'-'9' or enter/escape.

The function auto scales to fit on any LCD with size determined by phi_prompt initialization.

Example:

simple_text_area("Designed by Dr. Liu\nVersion:1.0\nFree software") will display a long message on LCD and allow you to scroll to see the entire message. The new line symbol '\n' is allowed and correctly interpreted. The message also automatically wraps at the end of each line.

It returns 1 for enter 255 for escape, or '0'-'9' for number keys.

9) byte simple_formatted_msg(char msg[])

Displays a long message to fill the size of the screen. It returns 0. It doesn't clear the screen.

msg This is the message to be displayed with formatting characters such as '\n' and '\t' in the message.

It doesn't trap execution or sense user inputs. The function returns as soon as the message is rendered. You may not see the entire message if it requires scrolling. It is intended for filling the screen with a message, simple like that, not anything more.

The function auto scales to fit on any LCD with size determined by phi_prompt initialization.

Example:

simple_formatted_msg("Designed by Dr. Liu\nVersion:1.0\nFree software") will display a long message on LCD.

It returns 0.

10) Additional functions

There are a few functions that I didn't introduce. They are used by the main functions and only become useful if you are constructing dynamic contents, which requires a fair amount of programming skills. Refer to *section 4 Using simple functions, subsection 6 Even more simple functions*.

8. List of standard library functions

There are two types of user interface functions included in this release. The simple functions, which we introduced in the previous section, are geared towards quickly whipping up a decent menu system with next to no effort. The classic functions are geared towards total control of every aspect of the interactivity. You can have many display options and display complex lists with smart use of the formal functions. They are harder to setup (simple functions need no setup). One important point to be made here is that simple functions are not only meant for the newbies. I developed them partly for my own projects and use them as testing grounds for new functionalities. Dynamic features such as dynamically updated contents in the menu, multi-select lists, and in-menu-adjust items, or menus/lists with variable amount of items can only be accomplished with simple functions.

The following are classic functions featured in the library. Some functions are only meant to be called by other library functions so they are not explained here.

1) void init_phi_prompt(LiquidCrystal *l, phi_buttons ** b, int w, int h, char i)

Parameters:

This is required before you can use any functions from the library. The l and b are just liquid crystals and phi_buttons array. The w and h are width and height of the display. The character i is the indicator character. I normally use the right arrow.
Eg.:

```
LiquidCrystal lcd(LCD_RS,LCD_EN,LCD_D4,LCD_D5,LCD_D6,LCD_D7);

phi_buttons btn_1(btn_u, LOW);
phi_buttons btn_2(btn_d, LOW);
phi_buttons btn_3(btn_l, LOW);
phi_buttons btn_4(btn_r, LOW);
phi_buttons btn_5(btn_b, LOW);
phi_buttons btn_6(btn_a, LOW);
phi_buttons *btns[]={&btn_1,&btn_2,&btn_3,&btn_4,&btn_5,&btn_6};

void setup()
{
  lcd.begin(lcd_columns, lcd_rows);
  init_phi_prompt(&lcd,btns,lcd_columns, lcd_rows, '~');
}
```

The above code is minimal to start the library. Notice lots of names such as LCD_RS, btn_u etc. are not given. You can use my code if you have a phi-2 shield or replace them with pins corresponding to your hardware setup. Also notice there is “&” in front of the lcd but not in front of the btns. The ‘~’ actually looks like a right arrow on the HD44780 display.

2) void set_indicator(char i);

Parameters:

This function sets the indicator character to i. Any subsequent list/menu rendering will use that character to indicate highlighted item. You can use things like ‘>’, ‘#’ or ‘*’ if you like them better.

3) void scroll_text(char * src, char * dst, char dst_len, short pos);

You can use this function to animate a line of text. If you have a long message and you want to animate it by scrolling it, you can call this function.
The src points to the source char array. The dst is the buffer that will be filled by the string cut from the source starting at position pos with length dst_len. So your dst char array should be at least dst_len+1 long.
Eg.

```
void horizontal_scroll_demo()
{
  char thankyou[]="Thank you for using phi_prompt!";
  char buffer[15];
  lcd.clear();
  for (byte i=0;i<47;i++)
  {
    scroll_text(thankyou,buffer,14,i-14);
    lcd.setCursor(1,1);
    lcd.print(buffer);
    wait_on_escape(250);
  }
}
```

The above function scrolls the char array thankyou and displays it on the LCD as an animation. Notice that when i<14, the position is negative, corresponding to amount of blank spaces before the first character. This helps the animation.

4) void scroll_text_P(PGM_P src, char * dst, char dst_len, short pos);

This is the same as the last function, only differs on the src points to a PROGMEM space.

Eg.

```
PROGMEM prog_char thankyou[]="Thank you for using phi_prompt!";

void horizontal_scroll_demo()
{
  char buffer[15];
  lcd.clear();
  for (byte i=0;i<47;i++)
  {
    scroll_text_P(thankyou,buffer,14,i-14);
    lcd.setCursor(1,1);
```

```
    lcd.print(buffer);
    wait_on_escape(250);
  }
}
```

The above animates a message stored in PROGMEM. This saves the SRAM.

5) void msg_lcd(char* msg_lined);

This displays a PROGMEM char array msg_lined at the current lcd cursor location. It is a basic function and provides no scrolling or clipping.

6) void center_text(char * src);

Parameters:

Char * src is pointing to a char array.

This displays char array src at the center of the top LCD line, automatically considering the size of the display, a good way to present a title. See fig. 6-1.

Eg. center_text("Top Menu");

(a)

(b)

Figure 6-1. center_text auto scales on different displays. (a) On a 16X2 display. (b) On a 20X4 display. The '>' and '<' are automatically filled to emphasize the title.

7) void scroll_bar_v(byte p, byte c, byte r, byte h);

Parameters:

This renders a vertical scroll bar at column and row (c,r), with height h rows and percentage of p.
This is used to render text area and list/menu when the scroll bar option is enabled. To render a scroll bar representing 30% of the position at column, row of (15,0) and a height of 4 rows on a 20X4 display, do the following:

scroll_bar_v(30, 15, 0, 4);

It makes very little sense to draw scroll bar on a 2-row display such as 16X2, 20X2 or 40X2.

8) void long_msg_lcd(phi_prompt_struct *para);

This displays a long message, stored in SRAM, depending on the setting of the para. See parameters explained in text_area().

9) void long_msg_lcd_P(phi_prompt_struct *para);

This displays a long message, stored in PROGMEM, depending on the setting of the para. See text_area_P().

10) byte render_list(phi_prompt_struct *para);

This function actually renders a static list instead of an interactive list so unless you just want to render a static list you want to use select_list() to render an interactive list for your user to interact with.

Parameters:

*para->ptr.list=(char**)&list_item_array;*

The list_item_array is a PROGMEM prog_char * array that stores the address of all list items. The following is an example of such an array:

```
PROGMEM prog_char pasta_00[]="Spaghetti";
PROGMEM prog_char pasta_01[]="Rotelle";
PROGMEM prog_char pasta_02[]="Rotini";
PROGMEM prog_char pasta_03[]="Fettuccine";
PROGMEM const char *pasta_items[]= {pasta_00,pasta_01,pasta_02,pasta_03};
```

para->low.i=high_lighted_item;
This is the highlighted item. If you enable the arrow/dot or flash cursor options, the arrow and flashing cursor will be on this item. The select_list() manipulates this value when the user presses up and down so different items will be highlighted.

para->high.i=last_list_item;

This indicates the last item on the list so the rendering program knows which item is the last item to render.

para->width=column_width;

This is the width of each column in characters. This width doesn't include arrow/dot or scroll bar spaces if these options are selected. It doesn't include the blank space between columns of items either. If you want a two column list with arrow/dot to occupy all space on a 16X2 display, then set this to 7. So each column is 7+1(arrow/dot) and two columns take all 16 spaces. If you disable arrow/dot, then you use 2*7 for both columns and 1 for space between the columns.

para->col=col;
para->row=row;

This is where the list starts, or its top left corner.

para->step.c_arr[0]=number_rows;
para->step.c_arr[1]=number_columns;

These are number of columns and rows a list has. This is different from width, which is the width of one column.

para->step.c_arr[2]=y2;
para->step.c_arr[3]=x2;

These are the positions of a list index if the index is enabled by the option.

para->option=option;

Below is the option byte:

List in SRAM	Reserved 0	Scrolling bar	Center choice	Flashing cursor	Auto scroll	Current /total	Index list 1 thru 0	Arrow or dot
256	128	64	32	16	8	4	2	1

Bit 0 – Arrow or dot. The first column of the list is a dot if it is not highlighted or an arrow (customizable) if it is highlighted. This is very useful if you display more than one item on your list and show which item is being highlighted. This is displayed to the left of the list so the list will be one column wider than what you would expect so plan accordingly
Bit 1 – index list 1 thru 0. The index looks like 123>567890 if 4 is highlighted. This is drawn at column and row, encoded in step.c_arr[3] as column and step.c_arr[2] as row. Eg. If you want to display this index at (col,row)=(12,0), you do step.c_arr[3]=12 as column and step.c_arr[2]=0. The highlighted item is indicated on the index as a special character, customizable. Read set_indicator(). Setting this bit disables the next index type.

Bit 2 – an index current/total, such as 5/12. This is drawn at column and row, encoded in step.c_arr[3] as column and step.c_arr[2] as row. Eg. If you want to display this index at (col,row)=(12,0), you do step.c_arr[3]=12 as column and step.c_arr[2]=0. The highlighted item is the number before the slash while the total list item is after the slash. This index is only displayed if the previous index is disabled since they use the same coordinates for locations.

Bit 3 – auto scroll. This enables auto scrolling of list items that are too long to fit within given display width by width.i. Not enabling this option results in truncated items.

Bit 4 – flashing cursor. This turns on a flashing black box enabled by the HD44780-comptaible display. The cursor is at the beginning of the list item highlighted. You can save a column of your display by enabling this option and disabling the arrow/dot option.

Bit 5 – Center choice. The highlighted item will always be centered in the display when possible (odd number of rows to display and plenty of items before and after the highlighted item). The list will scroll automatically to center the choice you might make. If this is disabled, the highlighted item goes from top to bottom of the list and you will flip to the next page with highlighted item on top of the next page. It works nicely with a long list and 3 rows of room to display it. Some people like this way better than flipping pages and pages of items instead.

Bit 6 – Scrolling bar. This displays a scrolling bar on the right side of the list. It looks like a DOS editor and is only useful for multi-line of list items. This will increase the width of your list by one column. Make sure that you don't have anything immediately to the right of the list or run out of screen space.

Bit 7 – Reserved. This is reserved for future options and should not be used. A future porting of this library to GLCD may use this bit for inverting text on highlighted item.

Bit 8 – List in SRAM. If your list is in the SRAM, set this bit. A lot of times you will want to construct a dynamic list in SRAM depending on your current situation. This option allows you to supply a pointer to SRAM for rendering.

You can use almost any combinations of the above options.

A few typical choices:

0 – Classic list with no features. This is used for simple static display of list or display one item (maybe after the user chooses it) from a list instead of asking the user to choose from the list.

1 – Classic list with arrow/dot indicator. This is used for simple display of list or display of chosen list item. It renders the list with arrow/dot indicator for user to see which item is highlighted. This is good enough as an interactive list. You may consider adding auto scroll to handle long items.

9 – Classic list with arrow/dot indicator and auto scroll the highlighted item if it is too long. This is used for simple display of list or display of chosen list item. It renders the list with arrow/dot indicator for user to see which item is highlighted. This is good enough as an interactive list. Everything else is cosmetic.

16 – Classic list with blinking box indicating the item highlighted.
24 – Above list with auto scroll of longer items.

33 – List that centers around highlighted item with arrow/dot indicator. This is used for typical list of more than 3 items. Centering highlighted item is only appropriate for single column list display. Using it on multiple-column list is not recommended.

41 – List that centers on the highlighted item with arrow/dot indicator, and auto scrolls the highlighted item if it is too long. This is used for typical list of more than 3 items and occasionally has long list items that need scrolling. Centering highlighted item is only appropriate for single column list display. Using it on multiple-column list is not recommended.

56 – Classic list with blinking box indicating the item highlighted and center choice and auto scroll long items.

73 – Classic list with auto scroll, scroll bar to the right, and arrow/dot option

80 – Classic list with scroll bar to the right and flashing cursor.

88 – Classic list with scroll bar to the right, auto scroll and flashing cursor.

105 – Classic list with center choice, auto scroll, scroll bar to the right, and arrow/dot option

112 – Classic list with center choice, scroll bar to the right and flashing cursor.

120 – Classic list with center choice, auto scroll, scroll bar to the right and flashing cursor.

121 – Classic list with center choice, auto scroll, scroll bar to the right, with arrow/dot option and flashing cursor.

OK combinations:
Auto scroll with flashing cursor makes the cursor flash

Bad combinations:
8 – Classic list with auto scroll the highlighted item if it is too long. This is used for simple display of list or display of chosen list item. Since only long items scroll, it’s not a good way to indicate which item is highlighted by simply looking at which item is scrolling, unless all items are long.

32 – List that centers around highlighted item. This is used for typical list of more than 3 items. Centering highlighted item is only appropriate for single column list display. Using it on multiple-column list is not recommended. There is no indicator of which item is highlighted so it is not a good way.

40 – List that centers on the highlighted item and auto scrolls the highlighted item if it is too long. This is used for typical list of more than 3 items and occasionally has long list items that need scrolling. Centering highlighted item is only appropriate for single column list display. Using it on multiple-column list is not recommended. There is no indicator of which item is highlighted so it is not a good way.

Multiple-column with center choice is bad and confusing.

Return values:

If auto scrolling is enabled and the highlighted item needs auto scroll, the function returns 1 so the caller knows it needs to keep calling this function so that the auto scroll occurs. Otherwise it returns 0.

11) int wait_on_escape(int ref_time);

Parameters:

Ref_time is the time in milliseconds the function traps while checking for key press. This is a replacement of delay(int ms). It does the same thing but it also keeps checking all keys and returns 0 if no key press is detected and the wait is over. If a key is pressed while waiting, it immediately returns the key pressed and won't wait until the time is over.

Return values:

0 No key was pressed and wait is over.
1-6 keys up/down/left/right/confirm/escape was pressed and the wait was terminated.

You can use this to loop certain functions until a certain key is pressed:

Eg.

```
while(true)
{
  temp1=wait_on_escape(50);
  switch (temp1)
  {
    case 0: // Nothing happened in 50ms of delay.
    break;
```

```
    case 1: // Up is pressed. Do some up stuff
    break;

    case 2: // Down is pressed. Do some down stuff
    break;

    case 3: // Left is pressed. Do some left stuff
    break;

    case 4: // Right is pressed. Do some right stuff
    break;

    case 5: // Enter is pressed. Do some enter stuff
    break;

    case 6: // Escape is pressed. Let's just quit.
    return;
    break;

    default:
    break;
  }
}
```

The above loop monitors the key press and responses to different key press with different codes. It also escapes when escape is pressed.

12) int ok_dialog(char msg[]);

Parameters:
This renders an OK dialog. Primarily you want to show the user a message and the user needs to press a key to continue. Otherwise the function keeps waiting. It doesn't actually return a value. The return value type is there to be consistent with the YES/NO dialog and any future dialog functions. The OK dialog just needs the char array that has the message to show to the user. The message will be truncated if there is not enough space. The dialog auto scales to occupy the entire display so you don't have to specify you have a 20X4 display or 16*2 display. See fig. 6-2 below.

Eg. *ok_dialog("Death ray was engaged. Annihilation in progress...");*

(a)

(b)

Figure 6-2. OK dialog auto scales on 16X2 and 20X4 displays. (a) 16X2 display. (b) 20X4 display. Intended message was “Death ray was engaged. Annihilation in progress...”. The text is automatically wrapped to occupy multiple lines. The message on a 16X2 display is clipped the so that it could fit.

13) int yn_dialog(char msg[]);

Parameters:

This renders a YES/NO dialog so the user can choose YES or NO.

Return values:

1 for YES and 0 for NO so the program can decide what to do in case of YES and NO. The dialog auto scales to occupy the entire display so you don’t have to specify you have a 20X4 display or 16*2 display. See fig. 6-3 below.

Eg. *int choice=yn_dialog(“Engage death ray now?”);*

(a)

(b)

Figure 6-3. YES/NO dialog auto scales on 16X2 and 20X4 displays. (a) 16X2 display. (b) 20X4 display. Intended message was “Engage death ray now?”. The text is automatically wrapped to occupy multiple lines. The message will be clipped if it doesn’t fit on a smaller display.

*14) int input_integer(phi_prompt_struct *para);*

Parameters:
para->ptr.i_buffer=&input_buffer;
This stores the address of the integer variable intpu_buffer. The initial value of the input panel is taken from this input_buffer. The returned value is also stored to this variable. Notice the "&" is needed.

para->low.i=lower_limt;
para->high.i=upper_limit;
para->step.i=step_size;

These set up the allowed numbers and how much is increased if up is pressed. Right now negative numbers are not supported. Use input_float() if you need negative numbers.

para->col=col;
para->row=row;
para->width=width;

These set up where to display the number and how many characters in width it occupies. With such pin-point accuracy, you can compact something like "adjust date" on one 16X2 screen with multiple integer inputs and select lists in series:

Figure 6-4. Sample screen for "adjust date".

para->option=option;

Option 0: space pad right; 1: zero pad left; 2: space pad left.

Actions:
It traps execution. It monitors the key pad input and changes the number as the user interacts with the keypad. When the user makes the choice, it stores the number in the input_buffer set up in the parameters and returns the key pressed so that the program can decide what to do next.

Return values:
Left key was pressed (-3)
Right key was pressed (-4)
Enter was pressed (1)
Escape was pressed (-1)

You may use left and right key to programmatically navigate among different entries or fields or treat them all as confirm except for -1.

*15) int input_float(phi_prompt_struct *para);*

Parameters:
para->ptr.i_buffer=&input_buffer;
This stores the address of the floating point variable intpu_buffer. The initial value of the input panel is taken from this input_buffer. The returned value is also stored to this variable. Notice the "&" is needed.

para->step.c_arr[0]=dig_after_decimal; // digits after decimal
para->step.c_arr[1]=dig_before_decimal; // digits before decimal

These set up the allowed number of digits after and before the decimal point. I have not implemented steps, upper and lower limits on float input. If you need a generic number entry that takes both positive and negative numbers with or without digits after the decimal point, use this one instead of input_integer().

para->col=col;
para->row=row;

These set up where to display the number. How many characters in width it occupies is derived from digits before and after decimal.

para->option=option;

Option 0: only positive numbers allowed, 1: only negative numbers allowed, 2: both positive and negative numbers are allowed.

Actions:
It traps execution. It monitors the key pad input and changes the number as the user interacts with the keypad. The user can move the cursor to any digit and change it and move to any other digits. When the user makes the choice, it stores the number in the input_buffer set up in the parameters and returns the key pressed so that the program can decide what to do next.

Return values:
Left key was pressed (-3)
Right key was pressed (-4)
Enter was pressed (1)
Escape was pressed (-1)

You may use left and right key to programmatically navigate among different entries or fields or treat them all as confirm except for -1.

16) int select_list(phi_prompt_struct *para);

If you want to present a list to the user to select (say day of the week or type of pasta), or you want a menu, you can use *select_list()*. The options of this function is exactly the same as the static list rendering program render_list(), except that it contains interactive code that changes which list item is highlighted. All parameters are kept the same and passed to render_list().

17) int input_panel(phi_prompt_struct *para);

Parameters:
para->ptr. ptr.msg =input_buffer;
This char array input_buffer is the buffer of the input panel. Its content will be printed out as the default content. Make the initial value close enough to what you expect from the user will save the user's time. The user input will be stored back to this array when done.

para->low.i=char_lower_limt;
para->high.i=char_upper_limit;

The above determines the range of valid input characters. You can assign 'A' and 'Z' to the limits to restrict to A to Z or use ' ' and '~' to include every character as valid inputs. There are cases where you only want say alphanumerical inputs for a password panel. Then you use 'A' and 'Z' and enable 0-9 input in the option. If you just want numbers then you can use '0' and '9'. Notice that if the buffer initially has characters not allowed by the rule, then those characters won't be allowed to change. This is nice to have. Say if you want the user to enter a file name then you can have "AAAAAAAA.TXT" in the buffer and only allow A to Z with 09 option. Then the dot in the file name won't be able to change. You can also construct something like "800-244-1111" and only allow 0-9 input. Then the '-' won't be able to change and the user can change other digits to complete their phone number entry.

para->width=panel_width
para->col=col;
para->row=row;

These set up the total length in characters of the input panel and where to display the panel. You should not make this longer than the size of the screen. I have not implemented any scroll input panel yet.

para->option=option;

Option 0, default, option 1 includes 0-9 as valid characters besides those indicated in upper and lower limits.

Actions:

It traps execution. It monitors the key pad input and changes cursor position and each character from keypad input. The user can move the cursor to any character and change it and move to any other character. The content of the buffer is modified every time the user changes any characters so if you want a backup, you should do so before passing the buffer to the function. When the user makes the choice, it returns the key pressed so that the program can decide what to do next.

Return values:
Left key was pressed (-3)
Right key was pressed (-4)
Enter was pressed (1)
Escape was pressed (-1)

You may use left and right key to programmatically navigate among different entries or fields or treat them all as confirm except for -1.

*18) int input_number(phi_prompt_struct *para);*

Parameters:
para->ptr. ptr.msg =input_buffer;
This char array input_buffer is the buffer of the input number. Its content will be printed out as the default content. Make the initial value the same length as the input length and contain all spaces. The user input will be stored back to this array when done.

para->width=panel_width
para->col=col;
para->row=row;

These set up the total length in characters of the input number and where to display the panel. You should not make this longer than the size of the screen. I have not implemented any scroll input panel yet.

para->option=option;

Option 0, default, option 1 conceal input as password input.

Actions:
This function is for keypads with numerical keys only. If you only have up/down keys, please use input_integer(). It traps execution. It monitors the key pad input and changes cursor position and each character from keypad input. The user can move the cursor to any character with left and right. The content of the buffer is modified every time the user changes any characters so if you want a backup, you should do so before passing the buffer to the function. When the user makes the choice, it returns the key pressed so that the program can decide what to do next. Up key outputs a negative sign and moves cursor to the right. Down key outputs a decimal sign and moves cursor to the right.

Return values:
Left key was pressed (-3)
Right key was pressed (-4)
Enter was pressed (1)
Escape was pressed (-1)

After the function returns, you will need to use sscanf to convert the string into number. See example code.

*19) int text_area(phi_prompt_struct *para);*

Parameters:
para->ptr. ptr.msg =input_buffer;
This char array input_buffer has the content to be displayed in the text area, which is an area that displays text with wrapping and scrolling on up/down keys and optional scroll bar. Changing the content of the buffer is doable but not suggested as the alignment of the text can be messed up. The purpose of this function is to show long texts such as credits and instructions. A later revision has enabled "new line" with the '\n' character so the text area will start a new line every time a '\n' is found. This greatly improves readability and portability of your message across different-sized displays.

para->low.i=char_lower_limt;
para->high.i=char_upper_limit;

The above determines the range of the message to be displayed. Currently the upper limit of the message is automatically set to the end of the char array.

para->col=col;
para->row=row;

These set up the total width in characters of the text area and the top left corner of the area.

para->step.c_arr[0]=row;
para->step.c_arr[1]=col;

The above gives the size in columns and rows of the text area. If scroll bar is enabled in the option, expect the text area to be one character wider than this width in columns.

para->option=option;

Option 0: display classic message; option 1: display message with scroll bar on right.

Actions:
It traps execution. It monitors the key pad input and changes message position with up and down keys. When the user presses left and right keys, the message scrolls up or down one

page, which is 2 lines if you display a 3-line message or 3 lines if you display a 4-line message. If the user presses 1-9 number keys, the function returns these numbers in ASCII for simple list select functions.

Return values:
'1'-'9' if the user presses one of these keys
Enter was pressed (1)
Escape was pressed (-1)

You may use left and right key to programmatically navigate among different entries or fields or treat them all as confirm except for -1.

*20) int text_area_P(phi_prompt_struct *para);*

This has the same functionality except that the char array is stored in PROGMEM so that the following parameter is different from the original version:

para->ptr. ptr.msg =input_buffer;

The input buffer is the array name of a PROGMEM prog_char array, where the message is stored. This method is preferred over text_area() to preserve SRAM.

21) Customizing your list or menu

Since I included example codes for menus, you can load it to your Arduino and go through the "set menu style" exercise to find your favorite menu style and copy it down. A 20X4 display will take full advantage of the library's various rendering functions.

9. Updates

09/18/2012:
The 1.0 version is ready for release. I have finally finished documenting the simple functions. They are extremely useful for quickly getting a menu or user interface together as well as dynamic features. Sample code using these simple functions are also included in the release. What is not included in the release is a phi_prompt professional development kit manual. It includes detailed explanation of how menu rendering works, accompanied by various annotated examples on dynamic menu content, multi-select menu, in-menu-adjust etc. This is meant for much more serious user interface construction. You can see previews of these features here:

http://www.youtube.com/watch?v=XYJ9nkugFEM

This level of programming requires some solid background in C/C++ so if you have not finished a few project of your own, you are better off with the simple functions. You may

contact me for a copy of the phi_prompt professional development kit manual at a fee, which includes timely technical support.

03/01/2012:
Since the success of this library's first official release on 05/23/2011, I have been hearing back from users commenting and requesting new features such as using rotary encoder as input, or analogbutton as inputs. I have decided to completely revamp the input device class I used in the 523 release, which was phi_buttons. Now the phi_prompt comes with the latest and greatest ever input device class library the phi_interfaces library as the physical layer for input devices. You can use any one or combination of any way the following devices as your input: buttons, matrix keypad, analogbuttons, rotary encoder, 2-axis analog joystick, serial port or serial keypad, or my popular phi-panel serial LCD keypad panel. You can use your cell phone with blue tooth serial port as a keypad if you want too!

I have also thought about how complex the 523 version seems to newbie's so I decided to add a few simple functions for starters. If you are not good with programming or are concerned with time spent constructing a user interface, these functions are for you. You can create a menu or list or display a multiple-page long message, all with one line of code. I am actually using these simple functions in some of my projects and products. The function automatically utilizes the entire screen and chooses the best way to display your menu and list, all without your intervention.

Despite all the above changes, all you need to do if you have previously programmed with phi_prompt, is the initialization part of the library. All previous function calls stay the way they are! If you are updating your project from phi_prompt 523 to phi_prompt 1.0, read my notes in a later section.

Since now that my input device class has drastically enabled various input devices to be seamlessly integrated together, I am considering a possible output device class to incorporate various output devices such as character LCD, graphic LCD, serial port etc. as alternate output devices so that Arduino will eventually feel like a computer.

05/23/2011:
There have been major updates to this library since the 20110427 version was released.
Most of the updates don't affect how the library is invoked but you do want to know what have I done within the month, right?

- ☆ The pointers to lcd (.lcd) and buttons (.btns) are removed from the struct (phi_prompt_struct) and placed in the library as static variables to be shared by all structs. This simplifies programming so:

  msg_lcd(msg_00,&lcd);
  wait_on_escape(4000,btns_1);

  becomes

```
msg_lcd(msg_00);
wait_on_escape(4000);
```

- ☆ Several static variables are added to the library to store the configuration of the library. The indicator is the character used in various places to highlight a list item.

```
static LiquidCrystal * lcd;
static phi_buttons ** btns;
static int lcd_w;
static int lcd_h;
static char indicator;
```

- ☆ The list renderers were removed and rewritten into one list renderer, *render_list()* , which has all previous features and a whole zoo of new features.
- ☆ Added auto scroll feature for lists so if you have tight space and long list items, enable this feature to see your list item scroll automatically like on your MP3 players.
- ☆ Added vertical scroll bar. You have the option to display it to the right of your list or text area. You can also call *scroll_bar_v()* directly to draw a vertical bar anywhere you want, although I don't know if that is useful.
- ☆ Added YES/NO and OK dialogs to simplify getting responses from the user. These dialogs automatically scale to maximal screen size.

10. Future improvement

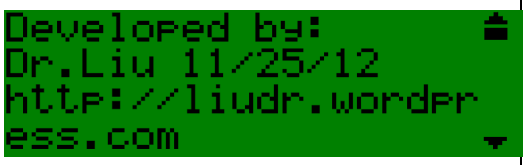
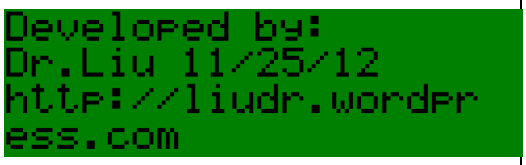
I would pounder on the following in the future:

- Feedback from Arduino fans like you can help me a lot. Visit http://liudr.wordpress.com and leave your feedback under Phi_prompt
- I will consider a maximize function to easily set up a text area to occupy all LCD space.
- I will also consider implementing the update_function() to run every time the user value is updated to make number entries into a real-time adjustment.

11. The legal stuff

The software is free to personal and educational uses only. If you intend to incorporate it in your commercial product, Contact me at http://liudr.wordpress.com. The developer assumes no responsibility for personal injuries or property damages if you use the library.

12. Phi_prompt simple functions cheat sheet

Command	Rendering
Simple menu or select list *response=simple_select_list("Day of week:\nSUN\nMON\nTUE\nWED\nTHU\nFRI\nSAT\n");* **Hint:** terminate every entry with '\n' **Result**: respond=0 to 6 for items 1 to 7	
Simple text area (scrollable) *response=simple_text_area("Developed by:\nDr.Liu 11/25/12\nhttp://liudr.wordpress.com\nThis is an example.\nPress Confirm to continue");* **Hint**: Use '\n' to create new line **Result**: response= button you pressed to dismiss the message. Don't care if just displaying message.	
Simple long message (not scrollable) *response=simple_formatted_msg("Developed by:\nDr.Liu 11/25/12\nhttp://liudr.wordpress.com\nThis is an example.\nPress Confirm to continue");* **Hint**: Use '\n' to create new line **Result**: response= 0. The function exits immediately after rendering the message.	
Simple input panel *char msg[]="000000";* *lcd.clear();* *lcd.print("Enter number:");* *simple_input_panel(msg, '0', '9', 1)* **Hint**: msg is modified after the user changes it. **Result**: msg is modified to user input. Use sscanf to convert numeric text into number.