The “Menus” Library.

2013-12-05

This library enables you to handle *text based menus* in a rather simple/standardised way. It only handles the logical behaviour of the menu(s), not their physical appearance e.g. on an LCD: the using project should still define the display or drawing procedures.

Content

1 Terminology

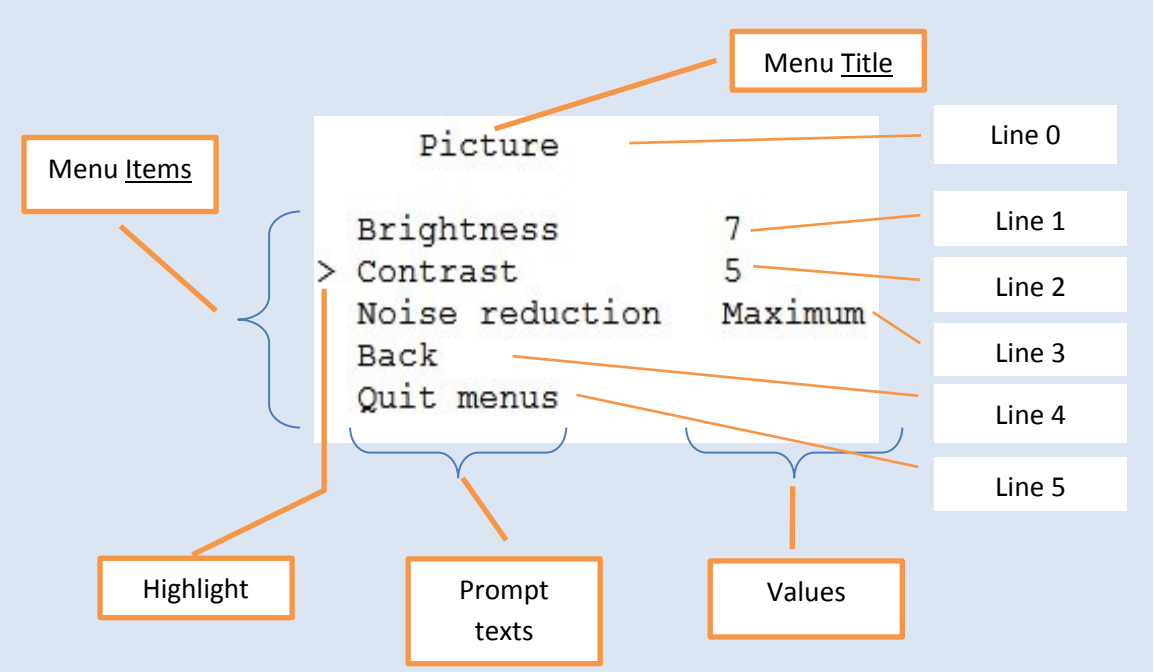

Figure 1

A menu consists of a *Menu Title* and a number of actual *menu items*. The menu items themselves have two parts: the *prompt texts* and optionally the *values* shown next to them. See Figure 1.

Additionally one of the menu items is the *currently "selected"* one, it is marked by some kind of *highlight* (e.g. inverse text or – as above – a ">" sign next to it). This is the menu item that is under control by the user. In Figure 1 line 2 is the selected one.

In the library (and troughout this document) the lines in the menu are numbered as in Figure 1:

- The *menu title* is always *line zero*,
- the *first itemline* is always *line 1*, the *second itemline* is *line 2*, etc, no matter how they are actually displayed on e.g. a screen. The line numbers are "*logical*" line numbers, not physical ones.

Menu lines come in a number of "kinds":

- **Title** with no functionality and no value displayed, (see line 0 in Figure 1)
- Capable of showing and changing a **decimal value** (see line 1 in Figure 1)
- Capable of showing and changing a **hexadecimal value**
- Capable of showing and changing an **enumerated value** (see line 3 in Figure 1)
- Capable of **navigating to another menu** (see line 4 in Figure 1)
- Capable of **leaving menu mode** altogether (see line 5 in Figure 1)

2 The usage of the library

To use the library a number of items/activities have to be provided by the project using it:

1. The ***The Control*** procedures that will be referred to in the menu definition (s)
2. The ***Observers*** that will be referred to in the menu definition (s)
3. The ***EnumTexts*** for enumerated values referred to in the menu definitions(s)
4. The ***Menu Defintions*** : a separate definition for each menu
5. The ***MenuTable*** : one table holding all addresses of above menu definition tables
6. The ***Display routines.*** for displaying the menu's and handling the highlight of the selected line
7. The ***Menu_Exit routine*** : a routine called by the menu machine when menumode is left
8. ***Initialisation*** of the menu machine
9. ***Enter/leave menu mode***
10. ***Sending Events to the menu machine***
11. ***Refresh Menu values***

2.1 The Control procedures

The "controls" defind in the several menu definitions must be defined before the menu definitions can be made (unless thay are defined "forward").

The signature of a "**Control**" is the following:

```
TControl  = procedure(Event: byte);
```

In which "Event" is the event causing the call to the control procedure. The Event here can have the values *mEValueUp* and *mEValueDown*.

Example the procedure "BrightnessControl":

```
procedure UpDown(Event: byte; var Value: integer; min, max: integer);
// common procedure to all control procedures
begin
  if (Event = mEValueUp)
  then  begin if Value < Max then inc(Value) end
  else  begin if Value > Min then dec(Value) end;
end;

procedure BrightnessControl(Event: byte);  // this is the Control procedure
begin
  UpDown(Event, Brightness, 0, 50);
  // send new value to the home cinema hardware
end;
```

As you can see the "Control" procedure simply increments or decrements the magnitude of a process variable (here in the example e.g. the Brightness of a home cinema system).

2.2 The Observers

The “observers” defind in the several menu definitions must be defined before the menu definitions can be made (unless thay are defined “forward”).

The signature of an “**Observer**” is the following:

```
TObserver = function: integer;
```

Example the function “BrightnessValue”:

```
function BrightnessValue: integer;
begin
  Result := Brightness;
end;
```

As you can see the “Observer” function simply returns the magnitude of a process variable (here in the example e.g. the Brightness of a home cinema system).

2.3 The EnumTexts

The “EnumTexts” defind in the several menu definitions (for menu items of kind “mkEnum”) must be defined before the menu definitions can be made.

The enumeration texts have to be defined as in following example for the “noise” in the picture menu definition above:

```
const NoiseStrings: array[3] of TMenuEnum =
      (
      'Off',
      'Minimum',
      'Maximum'
      );
```

Note: Make sure the corresponding process variable can have only the values 0..n-1 where n is the number of strings in the enumeration definition. In the above example the “NoiseControl” control must guarantee that the value returned by the “NoiseValue” observer is 0..2.

2.4 The Menu Defintions

For <u>each menu</u> a table has to be defined of the following form:

```
const PictureMenu: array[6] of TMenuLine =
// format:
//      Kind         Text              Control             Observer          EnumTexts
      (
      ( mkTitle,     'Picture',        5,                  nil,              nil),
      ( mkValue,     'Brightness',     @BrightnessControl, @BrightnessValue, nil),
      ( mkValue,     'Contrast',       @ContrastControl,   @ContrastValue,   nil),
      ( mkEnum,      'Noise reduction',@NoiseControl,      @NoiseValue,      @NoiseStrings),
      ( mkMenu,      'Back',           0,                  nil,              nil),
      ( mkQuit,      'Quit menus',     nil,                nil,              nil)
      );
```

In such table all menulines are defined: the title and all menu items. The members of each “line” are:

- **Kind**: the “kind” of the menuline. The “kinds” are:
  - mkTitle (the title if the menu)
  - mkValue (decimal value)
  - mkValueHex (hexadecimal value)
  - mkEnum (enumerated value)
  - mkMenu (navigation to another menu)
  - mkQuit (leave menumode)
- **Text**: this is the “*prompt” text* in each menu item. For the menutitle line this is *the title of the menu*.
- **Control**: A pointer to a procedure that will “control” (increment or decrement) the “value” displayed in the menu line. This procedure has to be provided by the program using the library. Exceptions:
  - the menu *title*: here the *number of menuitems* has to be entered (so all lines except the title line).
  - the *mkMenu* kind item: here the *number of the menu to goto* has to be entered.
- **Observer**: a pointer to a function that returns the numerical value of the entity controlled by the “Control” procedure. This function has to be provided by the program using the library. Only present (not nil) for menuitems that show a value.
- **EnumTexts**: a pointer to an array of strings that contain the enumerationtexts for “mkEnum” kinds of menu items.

2.5 The MenuTable

Additionally an overall table containing pointers to all menu definitions, in the following form:

```
const MenuTable: array[4] of  ^const TMenuline =
      (
      @MainMenu,    // this is menu "0"
      @PictureMenu, // this is menu "1"
      @SoundMenu,   // this is menu "2"
      @CodesMenu    // this is menu "3"
      );
```

The order in table “MenuTable” defines the menu “numbers” (starting from zero). This number is to be used in the *mkMenu* kind of menu items.

The name of above table can not be chosen freely, it has to be “MenuTable” .

2.6 The Display routines.

The library expects to see following routines defined by the project using the library:

2.6.1 The “Display_Text” routine

The routine Display_Text should display the string “Txt” on the device that shows the menu.The parameters “Line” and “Value” should be used to decide where to display “Txt”. “Line” is the menuline (starting with 0 for the header, as usual), and “Value” (boolean) indicates that “Txt” should be displayed in the “prompt” area (“Value” = false) or in the “Values” area (“Value” = true) of the menu.

The string “Txt” can of course be manipulated before it is displayed, e.g. a “+” sign can be added in front of it.

Example:

```
procedure Display_Text(Menu, Line: byte; Value: boolean; var Txt: string);
// "Value" indicates the text is a "value" (numerical or enumerated)
var XVal: byte;
begin

  if Line = 0 then       // a "title" text
  begin
    Cls;
    GotoXY(5,0);         // position of the title
  end
  else

  begin                  // a text for a "menu item"
    XVal := 2;           // a "prompt" is to be displayed

    if Value then
    begin                // a "value" is to be displayed
      XVal := 20;

      // "Balance": we want a '+' sign if positive value in menu 2, line 2, so:
      if (Menu = 2) and (Line = 2) and (Txt[0] > '0') then StrAppendPre('+', Txt);
    end;

    GotoXy(XVal, Line + 1); // empty line below the title
    ClrEol;
  end;

  Uart1_write_Text(Txt);
end;
```

The routine should always have the name *Display_Text*, no other name is allowed.

2.6.2 The "Display_Highlight" routine

This routine should highlight in some way (like inverse display, highligh character next to the line, etc…) the selected menuline, represented by the parameter "Line". The parameter "On_" indicates if the line should be highlighted or not.

Example:

```
procedure Display_Highlight(Menu, Line: byte; On_: boolean);
var Ch: char;
begin
  if Line > 0 then Line := Line + 1; // we want an empty line below title
  GotoXY(0, Line);
  Ch := ' ';
  if On_ then Ch := '>';
  Uart1_Write(Ch);
end;
```

The routine should always have the name *Display_Highlight*, no other name is allowed.

2.7 The Menu_Exit routine

This routine will always be called when the menumode is left (by the menu machine itself or by calling the "Menu_Off" procedure). Do not call this routine yourself (it is a "callback" routine).

Example:

```
procedure Menu_Exit; // routine will be called when the menumode is left from inside the
menu
begin
  Cls;
  Uart1_write_text('Menu mode left, type "m" to enter menu mode again');
end;
```

The name of the routine can NOT be chosen freely.

2.8 Initialisation

All items described above are "definitions" that have to be made before the library can actually be used. From this section onwards the actual usage of the library itself is described.

Before anything else the library has to be initialised. This is done as follows:

```
Menu_Init;
```

The Menu_Init routine is not blocking.

2.9 Enter/leave menu mode

The menu mode can be entered or left with a call to routines `Menu_On;` and `Menu_Off;`
Both routines are not blocking.

Example:

```
// Initialise the menu machine
  Menu_Init;
```

One can test if the menu mode is active with:

```
If Menu_Mode then … // returns boolean (true when the menumode is on)
```

One can obtain the *MenuNumber* and the *LineNumber* with:

```
Function Menu_Status: word;
```

This function returns the *Menu Number* in the high byte and the *selected MenuLine* in the low byte of its result. If menu mode is not active the result is $ffff (65535).

2.10 Sending Events to the menu machine

The using program has to send events to the menu machine e.g. in the main program loop. The menu machine does not intercept any events by itself.

The menu machine knows only 4 events:

- meValueUp
- meValueDown
- mePrevLine
- meNextLine

The events do what their name suggests: *mePrevLine* and *meNextLine* will select the previous and the next menu item line with respect to the currently selected one, no wrap around; *meValueUp* and *meValueDown* will call the corresponding “Control” procedure (if any) to increment or decrement the intended variable (or to do some action if no variable is to be controlled).

The sending of an event to the menu machine is done with e.g.

```
Menu_Event(meValueUp);
```

The Menu_Event routine is not blocking.

2.11 Refresh Menu values

When a value (possibly) displayed in a menu is updated without the menu doing it (something else caused the change), then one can update all values in the currently displayed menu with:

```
procedure Menu_Refresh;
```

The Menu_Refresh routine is not blocking.

3 Interface of the Menus Library

```
unit Menus;

{ Declarations section }

const // menuline "kind"
      mkTitle    = 0;  // menu title line
      mkValue    = 1;  // numeric value changing type of line
      mkValueHex = 2;  // same as above, but display is in hex
      mkEnum     = 3;  // enumerated value changing type of line
      mkMenu     = 4;  // goto other menu type of line
      mkQuit     = 5;  // quit menumode type of line

type TObserver = function: integer;
     TControl  = procedure(Event: byte);
     TMenuText = String[25];         // menu item texts
     TMenuEnum = string[10];         // menu Value enumeration texts

     TMenuLine =
     record
       Kind    : byte;
       Text    : TMenuText;          // menuline prompt text
       Control : ^TControl;          // "action" or "control" procedure, called when up/down events
                                     // are handled
       Observer: ^TObserver;         // fetches the current "value" to be displayed
       EnumVals: ^const TMenuEnum;   // the strings belonging to an enum value fetched
     end;

const // Menu "events"
      meNoEvent   = 0;
      meValueUp   = 1;
      meValueDown = 2;
      mePrevLine  = 3;
      meNextLine  = 4;

// external display procedures
procedure Display_Highlight(Menu, Line: byte; On_: boolean); external;
procedure Display_Text(Menu, Line: byte; Value: boolean; var Txt: string); external;

// external Menu Exit callback procedure
procedure Menu_Exit; external;

// external declared constants
const MenuTable: array[1] of  ^const TMenuline; external; // dummy size

// published procedures
procedure Menu_Init;
procedure Menu_On; // will show "menu 0", the main menu
Procedure Menu_Off;
procedure Menu_Refresh;
procedure Menu_Event(E_: byte);
function  Menu_Mode: boolean;
function  Menu_Status: word;


implementation
```

5 Example of Usage

In the example below the input (key events) come in via the Uart, and output (= display of the menustrings) is sent via Uart to a VT100 (emulated e.g. with HyperTerm) terminal. The uart speed is 115200 bd.

The keystrokes accepted are:

- ‘m’: enter menu mode
- ‘q’: quit menu mode
- ‘u’: value up
- ‘d’: value down
- ‘n’: next menu item
- ‘p’: previous menu item
- ‘r’: refresh currently displayed menu

```
program Test_menus;

{$DEFINE UART_ON}

uses Menus, VT100;

// System variables
// ----------------
var Brightness,
    Contrast,
    Noise,
    Volume,
    Balance,
    Stereo,
    Code1,
    Code2,
    Code3: integer;

// Observers
// ---------
function BrightnessValue: integer;
begin
  Result := Brightness;
end;

function ContrastValue: integer;
begin
  Result := Contrast;
end;

function NoiseValue: integer;
begin
  Result := Noise;
end;

function VolumeValue: integer;
begin
  Result := Volume;
end;

function BalanceValue: integer;
begin
  Result := Balance;
end;

function StereoValue: integer;
begin
  Result := Stereo;
end;
```

```
function Code1Value: integer;
begin
  Result := Code1;
end;

function Code2Value: integer;
begin
  Result := Code2;
end;

function Code3Value: integer;
begin
  Result := Code3;
end;


// Control Procedures
// ------------------

procedure UpDown(Event: byte; var Value: integer; min, max: integer); // common procedure
begin
  if (Event = mEValueUp)
  then
  begin
    if Value < Max then inc(Value)
  end
  else
  begin
    if Value > Min then dec(Value)
  end;
end;

procedure BrightnessControl(Event: byte);
begin
  UpDown(Event, Brightness, 0, 50);
  // send new value to the home cinema hardware
end;

procedure ContrastControl(Event: byte);
begin
  UpDown(Event, Contrast, 0, 30);
  // send new value to the home cinema hardware
end;

procedure NoiseControl(Event: byte);
begin
  UpDown(Event, Noise, 0, 2);
  // send new value to the home cinema hardware
end;

procedure VolumeControl(Event: byte);
begin
  UpDown(Event, Volume, 0, 15);
  // send new value to the home cinema hardware
end;

procedure BalanceControl(Event: byte);
begin
  UpDown(Event, Balance, -10, +10);
  // send new value to the home cinema hardware
end;

procedure StereoControl(Event: byte);
begin
  UpDown(Event, Stereo, 0, 1);
  // send new value to the home cinema hardware
end;

procedure Code1Control(Event: byte);
begin
```

```
  UpDown(Event, Code1, 0, 17);
  // send new value to the home cinema hardware
end;

procedure Code2Control(Event: byte);
begin
  UpDown(Event, Code2, 0, $ff);
  // send new value to the home cinema hardware
end;

procedure Code3Control(Event: byte);
begin
  UpDown(Event, Code3, 0, 4);
  // send new value to the home cinema hardware
end;


// Menu value texts (for "enumerated" values)
// ------------------------------------------
const NoiseStrings: array[3] of TMenuEnum =
      (
      'Off',
      'Minimum',
      'Maximum'
      );

const StereoStrings: array[2] of TMenuEnum =
      (
      'Mono',
      'Stereo'
      );

const Code3Strings: array[5] of TMenuEnum =
      (
      'One',
      'Two',
      'Three',
      'Four',
      'Five'
      );

// Action procedures
// -----------------
procedure SoundtestControl(Event: byte);
begin
  GotoXy(0,15);
  Uart1_write_Text('Sound Test');
end;


// -------------------------- Menu Definitions --------------------------------

const MainMenu: array[5] of TMenuLine =
// format:
//      Kind         Text              Control    Observer   EnumTexts
      (
      ( mkTitle,     'Main Menu',      3,         nil,       nil),     // "Control" here is actually
the "number of itemlines, excluding header"
      ( mkMenu,      'Picture',        1,         nil,       nil),     // "Control" here is actually
the menunumber to show
      ( mkMenu,      'Sound',          2,         nil,       nil),
      ( mkMenu,      'Codes',          3,         nil,       nil),
      ( mkQuit,      'Quit menus',     nil,       nil,       nil)
      );

const PictureMenu: array[6] of TMenuLine =
// format:
//      Kind         Text              Control             Observer          EnumTexts
      (
```

```
      ( mkTitle,     'Picture',        5,                  nil,              nil),     // "Control"
here is actually the "number of itemlines, excluding header"
      ( mkValue,     'Brightness',     @BrightnessControl, @BrightnessValue, nil),
      ( mkValue,     'Contrast',       @ContrastControl,   @ContrastValue,   nil),
      ( mkEnum,      'Noise reduction',@NoiseControl,      @NoiseValue,      @NoiseStrings),
      ( mkMenu,      'Back',           0,                  nil,              nil),     // "Control"
here is actually the menunumber to show
      ( mkQuit,      'Quit menus',     nil,                nil,              nil)
      );

const SoundMenu: array[7] of TMenuLine =
// format:
//      Kind         Text              Control             Observer        EnumTexts
      (
      ( mkTitle,     'Sound',          6,                  nil,            nil),     // "Control"
here is actually the "number of itemlines, excluding header"
      ( mkValue,     'Volume',         @VolumeControl,     @VolumeValue,   nil),
      ( mkValue,     'Balance',        @BalanceControl,    @BalanceValue,  nil),
      ( mkEnum,      'Mode',           @StereoControl,     @StereoValue,   @StereoStrings),
      ( mkValue,     'Sound Test',     @SoundTestControl, nil,             nil),
      ( mkMenu,      'Back',           0,                  nil,            nil),     // "Control"
here is actually the menunumber to show
      ( mkQuit,      'Quit menus',     nil,                nil,            nil)
      );

const Codesmenu: array[6] of TMenuLine =
// format:
//      Kind         Text              Control             Observer        EnumTexts
      (
      ( mkTitle,     'Codes',          5,                  nil,            nil),     // "Control"
here is actually the "number of itemlines, excluding header"
      ( mkValue,     'Code1',          @Code1Control,      @Code1Value,    nil),
      ( mkValueHex,  'Code2',          @Code2Control,      @Code2Value,    nil),
      ( mkEnum,      'Code3',          @Code3Control,      @Code3Value,    @Code3Strings),
      ( mkMenu,      'Back',           0,                  nil,            nil),     // "Control"
here is actually the menunumber to show
      ( mkQuit,      'Quit menus',     nil,                nil,            nil)
      );

const MenuTable: array[4] of  ^const TMenuline =  // this table is necessary for "Menus"
      (
      @MainMenu,    // this is menu "0"
      @PictureMenu, // this is menu "1"
      @SoundMenu,   // this is menu "2"
      @CodesMenu    // this is menu "3"
      );

// ----------------Display procedures required by "Menus" ------------------------

procedure Display_Text(Menu, Line: byte; Value: boolean; var Txt: string); // display text; line 0 =
title, line 1 is 1st item,  etc...
// "Value" indicates the text is a "value" (numerical or enumerated)
var XVal: byte;
begin

  if Line = 0 then       // a "title" text
  begin
    Cls;
    GotoXY(5,0);         // position of the title
  end
  else

  begin                  // a text for a "menu item"
    XVal := 2;           // a "prompt" is to be displayed

    if Value then
    begin                // a "value" is to be displayed
      XVal := 20;

      // "Balance": we want a '+' sign if positive value in menu 2, line 2, so:
```

```
      if (Menu = 2) and (Line = 2) and (Txt[0] > '0') then StrAppendPre('+', Txt);
    end;

    GotoXy(XVal, Line + 1); // empty line below the title
    ClrEol;
  end;

  Uart1_write_Text(Txt);
end;

procedure Display_Highlight(Menu, Line: byte; On_: boolean); // line = 0 is header; line 1 is 1st
menu line etc
var Ch: char;
begin
  if Line > 0 then Line := Line + 1; // we want an empty line below title

  GotoXY(0, Line);

  Ch := ' ';
  if On_ then Ch := '>';
  Uart1_Write(Ch);
end;

// -------------- Menu Exit callback procedure  --------------------------------
procedure Menu_Exit; // routine will be called always when the menumode is left
begin
  Cls;
  Uart1_write_text('Menu mode left, type "m" to enter menu mode again');
end;
// -----------------------------------------------------------------------------

var Ch: char;

begin
  { Main program }

  Uart1_Init(115200);
  delay_ms(200);
  Cls;
  UART1_Write_Text('Started, type "m" to enter menu mode, type "q" to leave menu mode');

  // set the "process" variables to their initial value (here there are only 6)
  Brightness := 7;
  Contrast   := 5;
  Noise      := 2;
  Volume     := 10;
  Balance    := 0;
  Stereo     := 1;
  Code1      := 12;
  Code2      := $1a;
  Code3      := 3;

  // Initialise the menu machine
  Menu_Init;

  while true do // simple "process" loop, example of how to give events to the menu machine
  begin

    if Uart1_data_ready then
    begin
      Ch := Uart1_read;

      if Menu_Mode then // menu mode, send some events to the menu
      begin
        case ch of  // these are the actual events to be executed by the menu machine itself (there
are only 4)
          'u', 'U': Menu_Event(meValueUp);
          'd', 'D': Menu_event(meValueDown);
          'p', 'P': Menu_Event(mePrevLine);
          'n', 'N': Menu_Event(meNextLine)
```

```
          else
          begin
            // handle non menu events
            case Ch of
              'r', 'R': Menu_Refresh;  // refresh all value fields in the menu visible
              'q', 'Q': Menu_Off;      // leave menu mode
            end;
          end;
        end;
      end

      else  // not in menu mode
      begin
        case Ch of
          'm', 'M': Menu_On; // enter menu mode, display main menu
        end;
          // do non menu processing
      end;
    end;

  end;

end.
```

```
unit VT100;

// ---------------- VT 100 display procedures ----------------------------------

procedure GotoXY(X_, Y_: byte); // display gotoxy, coordinates start from zero
procedure ClrEol;
procedure Cls;

implementation

procedure GotoXY(X_, Y_: byte);
var StrX, StrY: string[3];
begin
  inc(X_);
  inc(Y_);
  ByteToStr(X_, StrX);
  ltrim(StrX);
  ByteToStr(Y_, StrY);
  Ltrim(StrY);
  Uart1_write_Text(#27 + '[' + StrY + ';' + StrX + 'H');
end;

procedure ClrEol;
begin
  Uart1_Write_text(#27 + '[K');
end;

procedure Cls;
begin
  Uart1_write_Text(#27 + '[2J' + #27 + '[1;1H');
end;

end.
```

[end of document]