UnoLib documentation

liquidcrystal.pas

version 11/10/2025

The liquidcrystal.pas module contains the implementation of TLiquidCrystal class, designed to simplify controlling Liquid Crystal Displays (LCDs) that use the Hitachi HD44780 controller chipset (the standard for most text-based LCDs).

Since FPC for AVR does not allow dynamic object creation, an object of type TLiquidCrystal declared in the var section is ready to use and does not require the use of a constructor.

TLiquidCrystal methods

```
procedure Init(rs, rw, enable, d0, d1, d2, d3, d4, d5, d6, d7:
UInt8);
procedure Init(rs, enable, d0, d1, d2, d3, d4, d5, d6, d7:
UInt8);
procedure Init(rs, rw, enable, d0, d1, d2, d3: UInt8);
procedure Init(rs, enable, d0, d1, d2, d3: UInt8);
```

Initializes the object, defining the pin connections. Multiple variations exist for 4-bit and 8-bit modes, with or without the Read/Write (*rw*) pin.

Parameters
rs - Register Select pin.
rw - Read/Write pin.
enable - enable pin.
d0..d7 – data pins. In 4-bit mode (most common) used parameters are: (rs, en, d4, d5, d6, d7), in 8-bit mode: (rs, en, d0, d1, d2, d3, d4, d5, d6, d7).

Note: In Arduino sources these methods are constructors.

```
procedure _init(fourbitmode, rs, rw, enable, d0, d1, d2, d3,
d4, d5, d6, d7: UInt8);
```

Generic procedure initializing the object, used by all variants of *Init* procedure.

```
procedure _begin(cols, rows: UInt8; charsize: UInt8 =
LCD_5x8DOTS);
```

Initializes the LCD interface and specifies the dimensions (columns and rows) of the display.

Parameters
cols - the number of columns.
rows - the number of rows.
charsize – character size constant (LCD_5x8DOTS, or LCD_5x10DOTS)

```
procedure Clear;
```

Clears the display and sets the cursor back to the home position (0, 0).

```
procedure Home;
```

Positions the cursor at the upper-left corner (0, 0). Does not clear the display.

```
procedure NoDisplay;
```

Turns the display off (disabling the visible text) without losing the content stored in its memory.

```
procedure Display;
```

Turns the display on, restoring the text and cursor state.

```
procedure NoBlink;
```

Turns off the blinking block cursor.

```
procedure Blink;
```

Turns on the blinking block cursor.

```
procedure NoCursor;
```

Hides the underline cursor.

```
procedure Cursor;
```

Displays the underline cursor.

```
procedure ScrollDisplayLeft;
```

Shifts the entire display content (text and cursor) one position to the left.

```
procedure ScrollDisplayRight;
```

Shifts the entire display content one position to the right.

```
procedure LeftToRight;
```

Sets the direction of subsequent text to the default, left-to-right.

```
procedure RightToLeft;
```

Sets the direction of subsequent text to right-to-left.

```
procedure Autoscroll;
```

Turns on automatic scrolling. Each new character added pushes existing characters one position away from the cursor.

```
procedure NoAutoscroll;
```

Turns off automatic scrolling.

```
procedure CreateChar(location: UInt8; charmapptr: UInt8P0);
```

Creates a custom character (a 5x8 pixel glyph) for use on the display. Up to 8 custom characters (numbered 0 to 7) can be defined.

Parameters
location - the memory location (0-7) where the custom character will be stored.
charmapptr – pointer to an array of 8 bytes, where each byte defines a row of the 5x8 pixel character.

```
procedure SetCursor(col, row: UInt8);
```

Positions the cursor (the location where the next text will be written).

Parameters
col - the column number (starting at 0).
row - the row number (starting at 0).

```
procedure WriteChar(aChar: char);
```

Writes a single character to the display.

Parameters
aChar – character to write.

Note: In Arduino sources the original method is called *write*.

```
procedure Write(aStr: shortstring);
```

Writes text to the LCD at the current cursor position.

Parameters
aStr – string to write.

Note: In Arduino sources the original method is called *print*. The *aStr* shortstring parameter consumes 256 bytes.

Example

See documentation of dht.pas module.