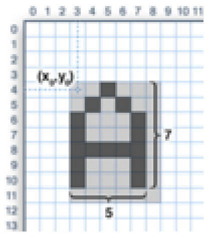
Arduino++

Arduino, CNC, software and other ramblings

Parola A to Z – Defining Fonts

ON NOVEMBER 8, 2016MARCH 15, 2017 / BY MARCO_C / IN ARDUINO, PAROLA, SOFTWARE



The Parola library (https://github.com/MajicDesigns/MD_Parola) allows you to display text on MAX72xx controlled LED matrices using a wide range of text effects. One of the base components for this flexibility are the replaceable fonts and the utilities that allow new fonts to be designed and implemented in the library code.

The design of the MD_Parola/MD_MAX72xx libraries results in the font management code being located in the MD_MAX72xx library. This allows code written for MD_MAX72xx to easily use text and frees the MD_Parola library from (mostly) needing to know where and how characters are stored.

As the pixel resolution resembles a scaled up version of the Hitachi LCD displays and the old 'green screen' monitors, the basis of the default Parola font comes from a combination of the pixel patterns from LCD displays and the system font from old MS-DOS systems.

The default font defined as part of the library is located in PROGMEM memory. Alternative fonts can be specified using library methods, and the font builder utilities (described below) provide a convenient way to modify existing, or develop alternative, fonts.

Font Data Format

At a basic level the font is just a byte data table in memory. The table is stored as a series of contiguous bytes, in a repeating pattern for each of the 256 ASCII characters. Each character bitmap is stored in the following format:

- **byte 0** is the number of column bytes (*n*) that form this character (could be zero if the character is not defined).
- **byte 1..*n*** – one byte for each column of the character. Byte 1 is the leftmost column of the character. The least significant bit of each data byte is the bottom pixel position of the character matrix (row 7 of the matrix).

If a particular ASCII character value is not defined, the size byte is 0. An empty data table is therefore represented by 256 zero bytes.

This is a flexible format that efficiently encodes each column of the character bitmap in one byte without restricting the number of columns that make up a character. Fonts can therefore be variable or fixed width, entirely determined by the first size byte, on a character by character basis.

An example of the standard font in the library shows how the code formatting can be used to logically separate the defined characters even though the data is physically a sequential byte stream:

```
  5, 0x30, 0x38, 0x3e, 0x38, 0x30, // 30 - 'Up Pointer'
  5, 0x06, 0x0e, 0x3e, 0x0e, 0x06, // 31 - 'Down
 Pointer'
  2, 0x00, 0x00, // 32 - 'Space'
  1, 0x5f, // 33 - '!'
  3, 0x07, 0x00, 0x07, // 34 - '"'
  5, 0x14, 0x7f, 0x14, 0x7f, 0x14, // 35 - '#'
```

To find a character in the font table, the library does a sequential search by looking at the first byte (the size *n*), skips *n*+1 bytes to the next character's size byte, repeating until the last or target character is reached.

The compile-time switch USE_INDEX_FONT enables indexing of the font table for faster access, at the expense of increased RAM usage (500 bytes total, or 1 word per ASCII character). The index is built for the default font during the library startup phase and rebuilt every time the font is changed. When indexing is enabled, a single lookup is required to access the character data, rather than the sequential search described above. By default indexing is disabled to conserve RAM.

If fonts are not required, the built in support for fonts (both methods, data and user defined fonts) may be completely disabled using the compile-time switch USE_LOCAL_FONT. By default fonts are enabled.

Double height fonts are stored in one font definition table by restricting the range of displayable ASCII character to 0..127. The lower half of each character is located at ASCII codes 0 through 127 and the upper part is located in ASCII codes 128 through 255, offset 128 positions from the lower half. For example, character 'A' is stored in positions 65, the normal location for a single height character, and 193 (65 + 128).

Creating Fonts

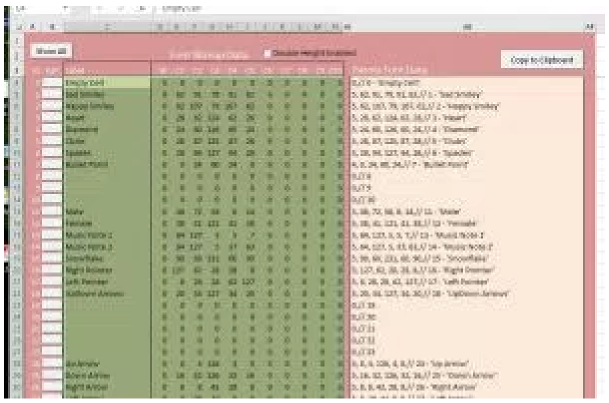
Creating font files manually is a tedious process, so the MD_MAX72xx library is supplied with two utilities to ease the pain.

- **txt2font** is a command line utility that processes a text definition of the font data file. This is implemented in C with source code supplied, allowing users on any operating system to compile a version to work with their OS.
- **FontBuilder** is a GUI utility implemented as VBA in Microsoft Excel. As FontBuilder requires Microsoft Office products, it does not work environments where these are not available.

For both utilities, several font definition files are supplied as examples or starting points for your own fonts.

The FontBuilder Excel/VBA application

FontBuilder is a Microsoft Excel spreadsheet with VBA macros to manage a GUI interface for defining and managing font characters. It supports both single and double height fonts and allows characters up to 10 columns wide.



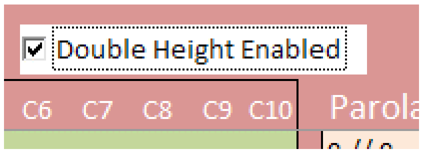
(https://arduinoplusplus.files.wordpress.com/2016/11/azparola_worksheets.jpg)

Data is stored and managed using different worksheets in the workbook. The main work sheet is the "Work Sheet". All macros are designed work in this sheet. Additional sheets are used to store font data that is not being worked on.

The red areas of the worksheet are protected from being overwritten, the green area is the main data manipulation space and the orange area is the output code that will be copied to the clipboard. Put simply – Red is no-go, green is for input and Orange is for output.

At any time, the orange output area will contain the latest font table and the “Copy to Clipboard” button will place the font file text in clipboard.

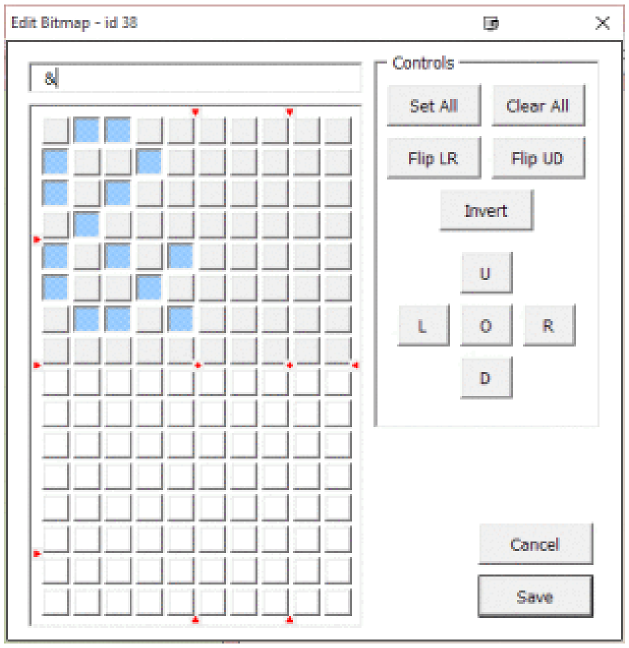
The work area is normally initialized to zero before starting a new font, or an existing font may be manually copied into the workspace. Data for a single character can be manually typed into the cells (useful for copying similar characters) or created using a special form that is reached from the corresponding small button in column 2 of “Work Sheet”.



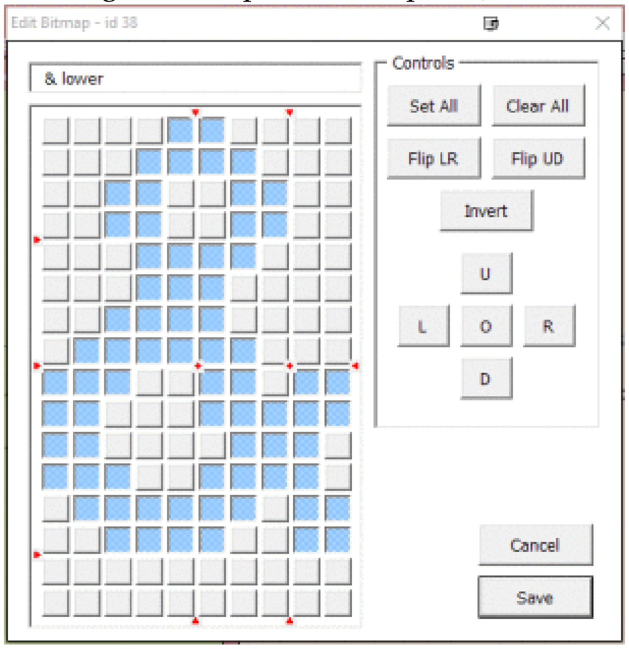
At the top of “Work Sheet” is a checkbox that allows the tool to be used in either single or double height mode. In single height mode each character is only allowed to be 8 LEDs tall, 16 for double height. This setting is used to enable different characters editing modes, simplifying management of the two halves of a double height character.

By default this tool will generate proportional fonts, but fixed width fonts can also be created. The width of proportional fonts is taken from the character definition. Fixed width characters are defined by changing the data sheet ‘W’ column to the exact width needed, overriding the automatic setting. If every character is fixed, then each W column entry will need to be set to the same value. It is worth noting that even with proportional fonts, the definition for the SPACE character needs to have a fixed width (suggested 2 for single height fonts) or it will result in a zero column character.

The character input form is used to define a character in the font file. The double height checkbox will change how tall the character definition can be, as shown by the two figures below, but the functionality remains the same.



(https://arduinoplusplus.wordpress.com/2016/11/08/parola-fonts-a-to-z-defining-fonts/azparola_fbinput1/)



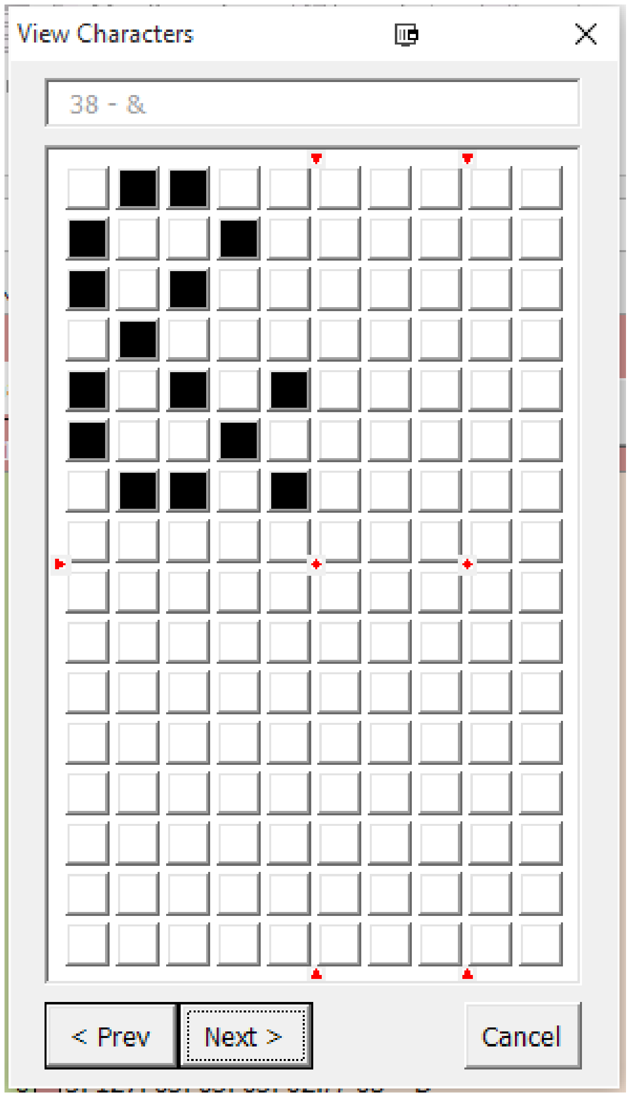
(https://arduinoplusplus.wordpress.com/2016/11/08/parola-fonts-a-to-z-defining-fonts/azparola_fbinput2/)
The title of the form includes the ASCII value for the edited character and the text field above the matrix grid is for editing the character label. To set or reset a pixel in the font, click on the pixel.

Small red markers are used to simplify alignment. These are located vertically at the 5 column and the 8 column positions. 5 columns is a common boundary for the many 5×7 fonts and 8 columns marks the edge of the 8×8 matrix square. Vertically the markers are located after rows 4 (midpoint of the top matrix), 8 (bottom of the top matrix) and 14 (location for double height descender).

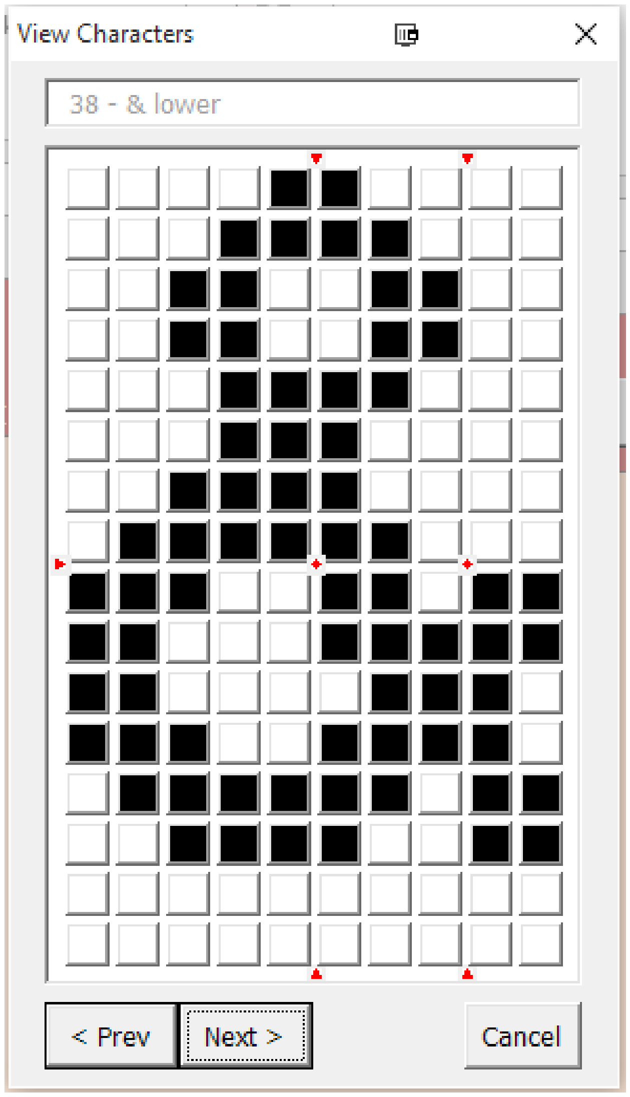
The edit buttons have the following function:

- **Set All** sets all the pixels ON, **Clear All** sets all the pixels OFF
- **Flip buttons** are to flip the definition Left to RIGHT (around the vertical axis) and Up to Down (around the horizontal axis)
- **L**, **R**, **U** and **D** slide the image by one pixel to the Left, Right, Up and Down, respectively
- **O** rotates the image by 90 degrees clockwise. The rotation only occurs for the upper left square of pixels – the additional pixels to the right or below (depending on double height mode) remain untouched.

The **Show All** button at the top left corner of the Work Sheet is used to scan through all the characters in a read-only mode, useful for seeing what characters are defined. How the character is displayed is controller by the double height checkbox setting.



(https://arduinoplusplus.wordpress.com/2016/11/08/parola-fonts-a-to-z-defining-fonts/azparola_show1/)



(https://arduinoplusplus.wordpress.com/2016/11/08/parola-fonts-a-to-z-defining-fonts/azparola_show2/)

The txt2font Utility

The txt2font utility is a command line application that converts a structured text definition of the font into a data file in the right format for MD_MAX72xx to use. The utility is supplied as an Win32 executable and source code for other OS.

The application is invoked from the command line and only the root name of the file is given as a command line parameter (eg “txt2font fred”). The application will look for an input file with a “.txt” extension (“fred.txt”) and produce an output file with a “.h” extension (“fred.h”).

The txt2font file format is line based. Lines starting with a ‘.’ are directives for the application. All other lines direct the current character definition. An example of the beginning of a font definition file is shown and explained below.

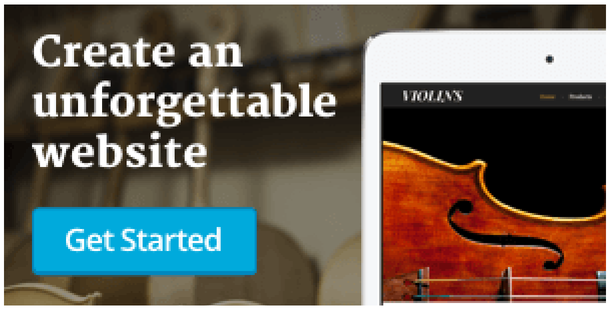
```
.NAME sys_var_single
.HEIGHT 1
.WIDTH 0
.CHAR 0
.NOTE 'Empty Cell'
.CHAR 1
.NOTE 'Sad Smiley'
 @@@
@@@@@
@ @ @
@@@@@
@@ @@
@   @
 @@@
.CHAR 2
.NOTE 'Happy Smiley'
```

The directives have the following meaning:

- **.NAME** defines the name for the font and is used in naming the font table variable. The name can appear anywhere in the file. If omitted, a default name is used. This directive should appear once at the start of a file.
- **.HEIGHT** defines the height for the font. Single height fonts are '1' and double height fonts are '2'. If omitted, the application assumes single height font. This directive should appear once at the start of a file.
- **.WIDTH** specifies the width of the font for all the characters defined between this WIDTH and the next WIDTH definition. Zero denotes variable (or unspecified) width; any other number defines the fixed width. WIDTH may be changed within the file – for example to define a fixed size space (where no pixels can be drawn!) character in a variable width font.
- **.CHAR** ends the definition of the current character and starts the definition for the specified ASCII value. Valid parameters are 0..255 for single height, and 0..127 for double height. If a character code is omitted in the font definition file it is assumed to be empty.
- **.NOTE** is an option note that will be added as a comment for the entry in the font data table.

Any lines not starting with a '.' are data lines for the current character. The font characters are drawn using a non-space character (eg, '*' or '@') wherever a LED needs to be 'on'. The application scans from the top down, so any lines missing at the bottom of the character definition are assumed to be blank. However, blank lines at the top need to be shown. Any extra rows are ignored will cause program errors.

The next part of this series (https://arduinoplusplus.wordpress.com/2016/11/13/parola-fonts-a-to-z-managing-fonts/) will explore the code to manage and use the font table.



ARDUINO FONT LED LIBRARY MD_MAX72XX PAROLA PAROLAAZ SOFTWARE

2 thoughts on "Parola A to Z – Defining Fonts"

1. **Don Pera**
   Very good explanation thanks.
   My question is using the application The FontBuilder Excel application as or so I created the Font.h file because I could not do it, and what I copy to the portapapels I change the extension to, and I have overwritten the font.hy library. No way the change I made from the source.
   Then uds. I could do the favor of exoplicar how and how this is done there is no report on this detail.
   My idea is to make the font.h library for vertical notice since googleando I could not find anything on it.
   I appreciate the attention lent and delighted with this resource greetings and excuse my poor English I do not write translate thank you

   JUNE 21, 2017 AT 17:47 REPLY
   marco_c

Hard to say what could be going wrong. I note you have raised a query in the Parola thread on the Arduino forum. The discussion will continue there.

JUNE 22, 2017 AT 07:37 **REPLY**