FontRotator quick guide

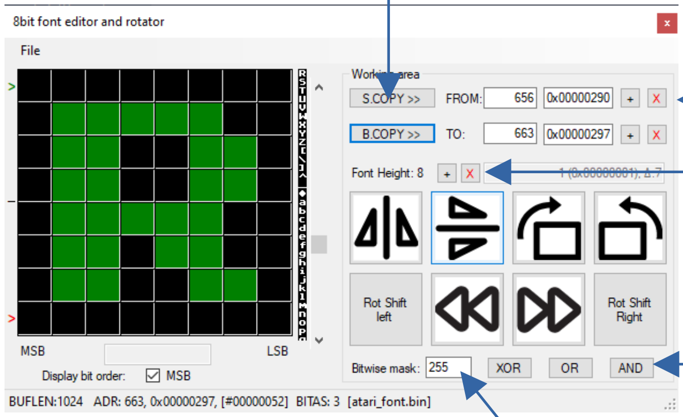
Free bitmap font editor, rotator and extractor

“S.COPY” and “B.COPY” buttons copy
Current screen’s “uper”, “start”, “from” and “bottom”, “end”, “to” address
to working area. All operations work only in these boundaries.

“>” green marker.
Start of work buffer.

“-” black marker.
Font “base”, size.

“>” red marker.
End of work buffer.



BUFLEN:1024 ADR: 663, 0x00000297, [#00000052] BITAS: 3 [atari_font.bin]

Delete or insert byte
at current address.

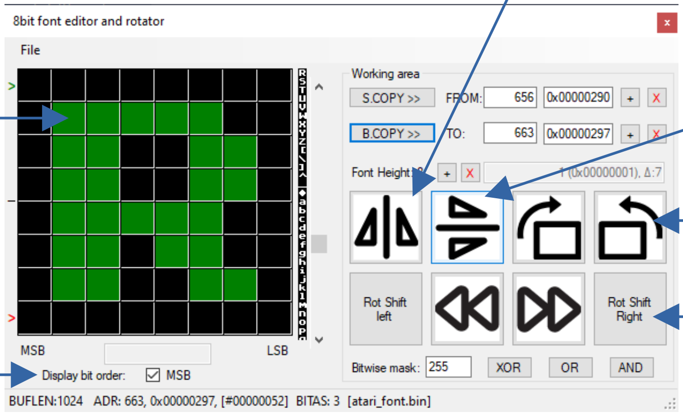
Change font height:
insert or delete byte in
each char.

Bit-wise logical
operation using mask
with work buffer.

Mask for logical operation,
HEX or DEC value.

Horizontal mirror, flip.

Press here to change
pixels (bits).



BUFLEN:1024 ADR: 663, 0x00000297, [#00000052] BITAS: 3 [atari_font.bin]

Vertical mirror, flip in
font heigh chunks.

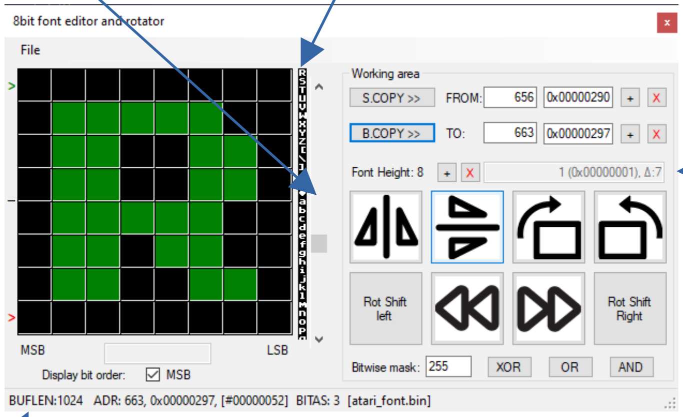
Rotate char left or right,
valid only for 8x8 chars.

Move bit left or right,
rotate bits.

Bit order change.
Only for display,
data not changed.

Navigation scrollbar,
also keyboard scroll
is working: arrow keys for
single line, page key- jump
one char at each press.

Preview bar, helps to find fonts in random binary file.



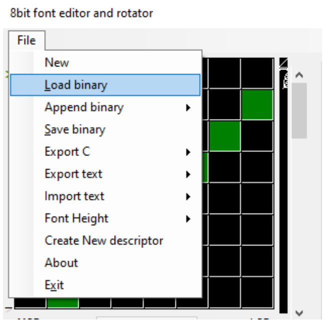
BUFLEN:1024 ADR: 663, 0x00000297, [#00000052] BITAS: 3 [atari_font.bin]

Number of chars in
working buffer, size of
the buffer in bytes.

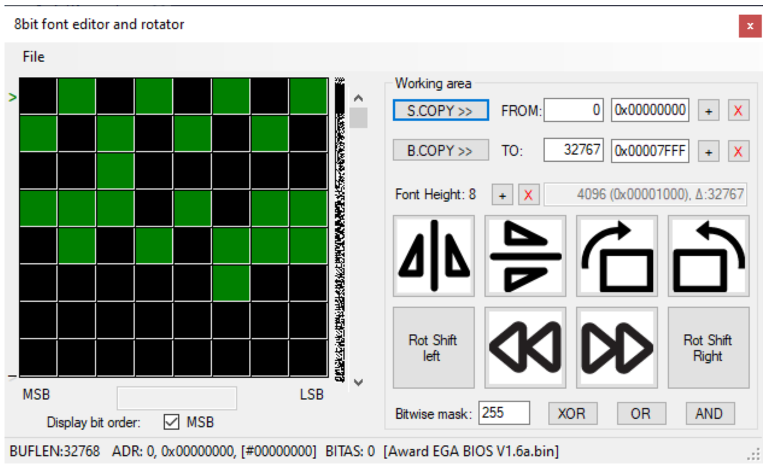
Status and information bar: total buffer size, current address of mouse cursor, character
number, current bit under the mouse, last used file name.

How to find font in some ROM file:

Select “load binary” from file menu and load your unknown file. Here I am using VGA BIOS file from some old ISA EGA video card.

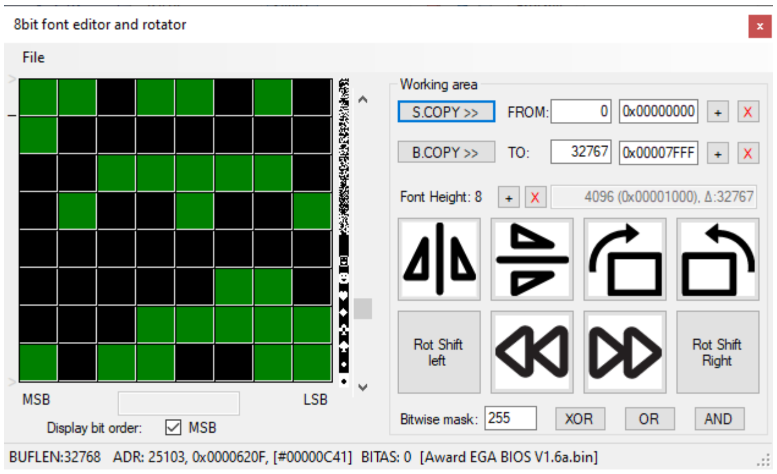


Binary file is loaded and truncated to 1M.
Displays is showing some random pixels-
it is video card software.



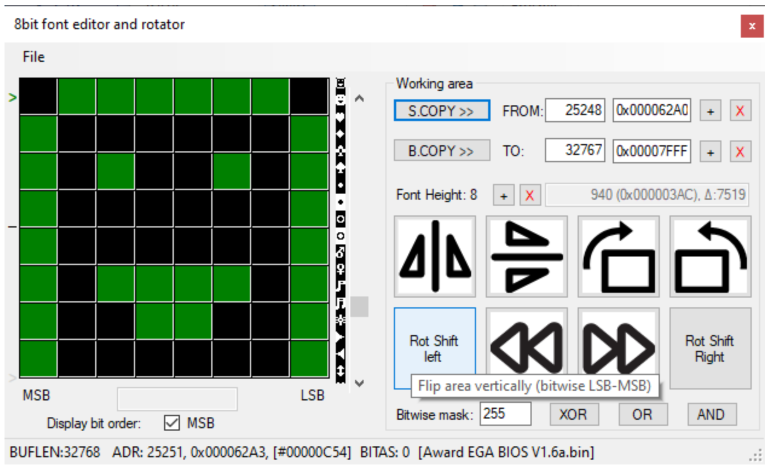
BUFLEN:32768 ADR: 0, 0x00000000, [#00000000] BITAS: 0 [Award EGA BIOS V1.6a.bin]

Scroll data until some recognizable characters appear in preview bar or in edit window.
This BIOS has several fonts inside.



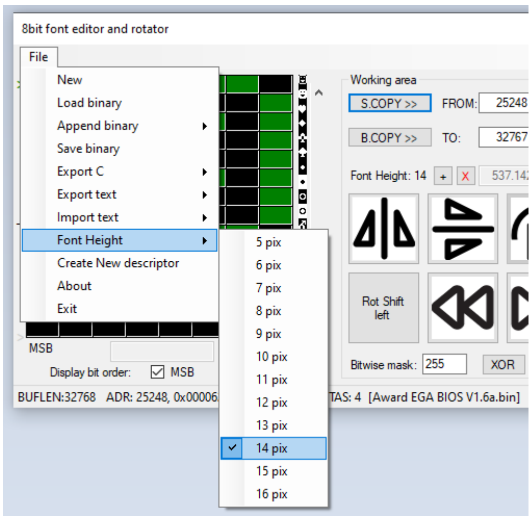
BUFLEN:32768 ADR: 25103, 0x0000620F, [#00000C41] BITAS: 0 [Award EGA BIOS V1.6a.bin]

Move scrollbar or press keys to align top of the first interested char with edit window’s top. Press “S.COPY” to record start address of the font.
Font size is bigger than 8 rows. Calculate font size or just guest it. This font is 14 pixel (rows) height.



BUFLEN:32768 ADR: 25251, 0x000062A3, [#00000C54] BITAS: 0 [Award EGA BIOS V1.6a.bin]

From program menu, select “Font height”, in this case it is 14 pix. This menu do not change anything in the font, only the display. Font height value is used in some editing functions, for font descriptors and C source export.

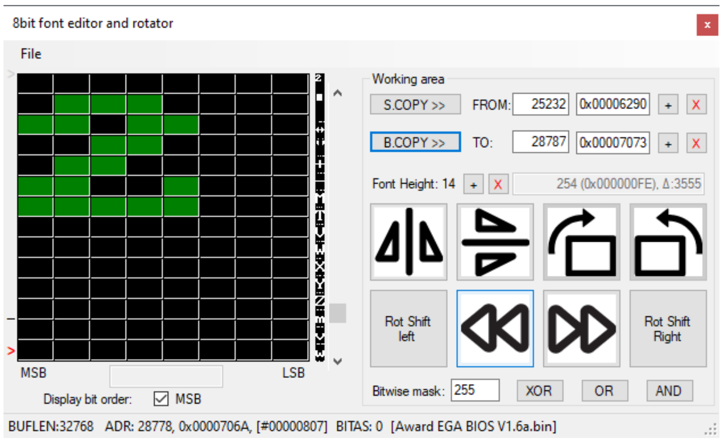


FROM: 25248
TO: 32767
Font Height: 14
537.14
Bitwise mask: 255

BUFLEN:32768 ADR: 25248, 0x00006

TAS: 4 [Award EGA BIOS V1.6a.bin]

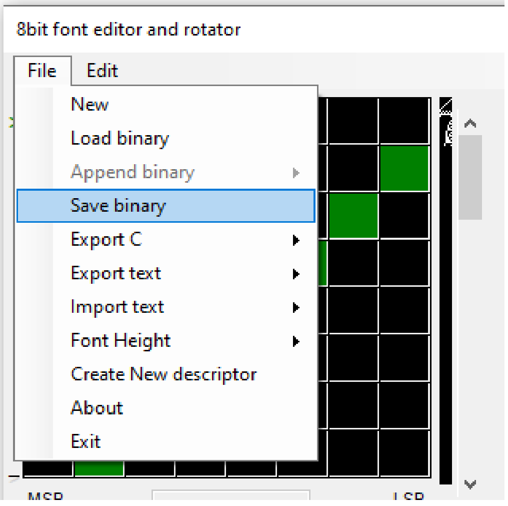
Scroll down to the last character we want to edit or export. Use “page down” key for fast scrolling.
When last char is displayed, press “B.COPY” button. The red marker will display bottom of the char, status window near font height will display size of selected data and calculate number of chars (at selected height) in this buffer.

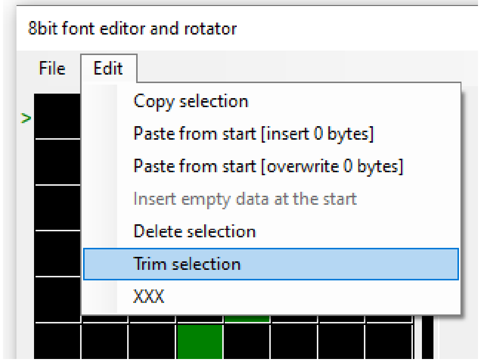


FROM: 25232 0x00006290
TO: 28787 0x00007073
Font Height: 14 254 (0x000000FE), Δ:3555
Bitwise mask: 255

BUFLEN:32768 ADR: 28778, 0x0000706A, [#00000807] BITAS: 0 [Award EGA BIOS V1.6a.bin]

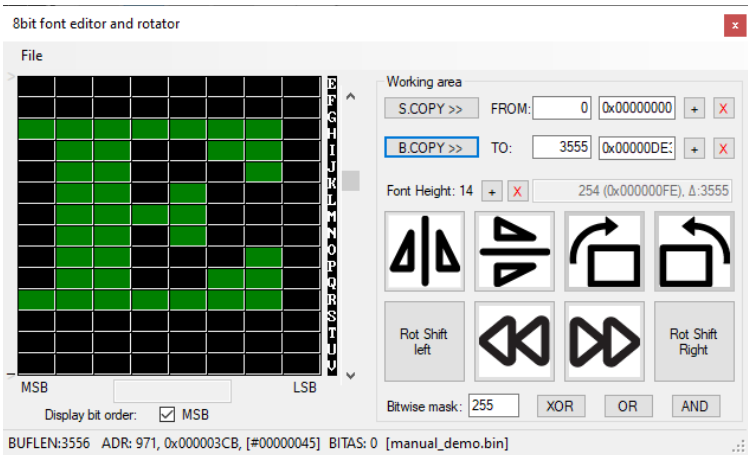
Now it is good time to save new binary. Do not forget to change file name, unless you want to overwrite file.
Reload new file to program. Now the file is smaller and contains only font selected in previous operations.
.




8bit font editor and rotator
File
Edit
Copy selection
Paste from start [insert 0 bytes]
Paste from start [overwrite 0 bytes]
Insert empty data at the start
Delete selection
Trim selection
XXX

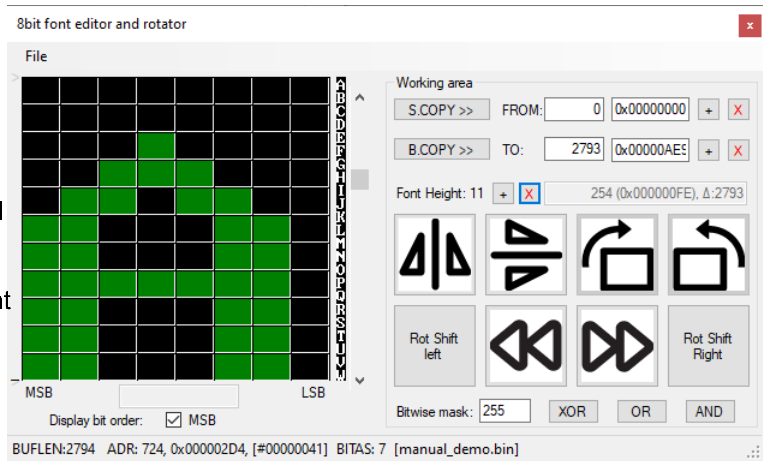
Or just select menu item “Edit/Trim selection”

For this demo, font height is too big, there are several lines below character.
Only some special characters are using this space. We can remove some lines at the ”char base” marked with black “-” symbol at edit window.
Just press red “X” near font height size.
New size of the font is “11”



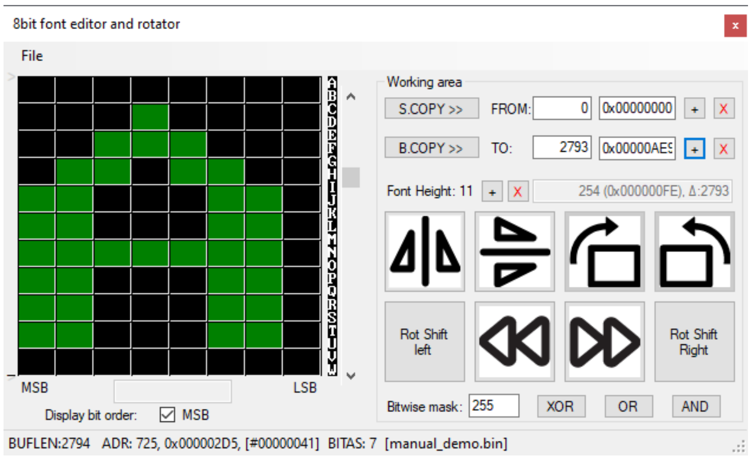
FROM: 0 0x00000000
TO: 3555 0x00000DE
Font Height: 14
254 (0x000000FE), Δ:3555
Bitwise mask: 255
Display bit order: ☑ MSB
BUFLEN:3556 ADR: 971, 0x000003CB, [#00000045] BITAS: 0 [manual_demo.bin]

Font height is 11 rows, whole buffer is smaller too: from 3556 bytes it shrunk to 2794.
Some symbols maybe are damaged, but basic font is OK. The only thing is, that symbols are on the bottom of the char matrix. We can chage it.
Press “X” button near “FROM” address. This will delete one byte at the beginning of the buffer. And press “+” button near “TO” address. This will add one byte at the end of buffer and the font will align to the new boundaries.
Status windows shows that we have 254 chars, difference between start and stop address is 2793, and bottom line shows that total size of the buffer is 2794 bytes (zero address is also one byte). The char “A” and mouse are at adress 724 decimal (or 02D4 hex), it is [41 hex]’st char in the file. The mouse pointer was last detected on 7th bit of current address in the edit window.



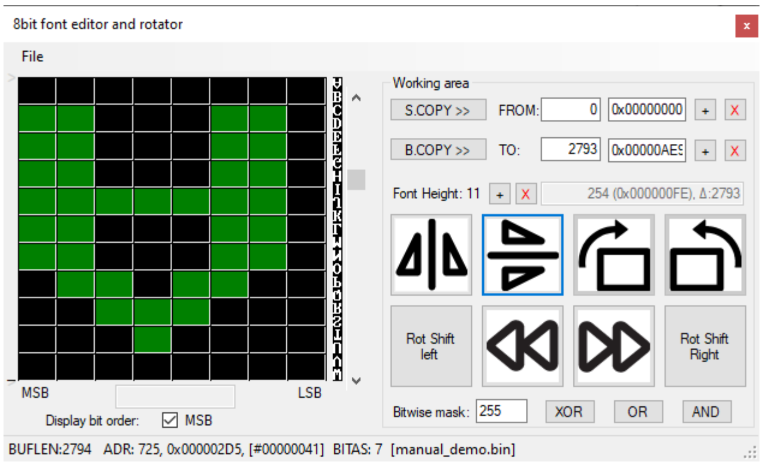
FROM: 0 0x00000000
TO: 2793 0x00000AE
Font Height: 11
254 (0x000000FE), Δ:2793
Bitwise mask: 255
Display bit order: ☑ MSB
BUFLEN:2794 ADR: 724, 0x000002D4, [#00000041] BITAS: 7 [manual_demo.bin]

Do not forget to save your new extracted font to file.
Now we can edit pixels in the font, delete unused characters, move or mirror whole font or even single chars or groups. To work with single char or group of chars, use “COPY” buttons to select working area.
After all edits, do not forget to select whole working area to save or export. All file operations are dealing only withing the selected area.
Take note, that “rotate” functions are only valid with the 8x8 pixel fonts.



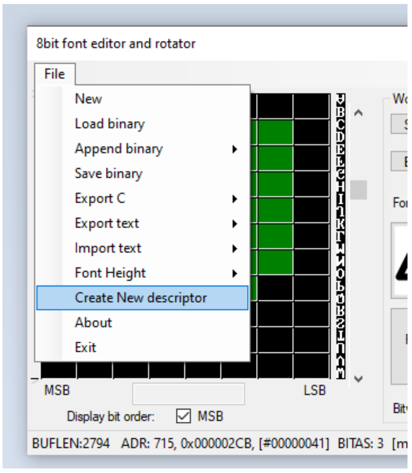
FROM: 0 0x00000000
TO: 2793 0x00000AE
Font Height: 11
254 (0x000000FE), Δ:2793
Bitwise mask: 255
Display bit order: ☑ MSB
BUFLEN:2794 ADR: 725, 0x000002D5, [#00000041] BITAS: 7 [manual_demo.bin]

No problems to mirror whole or only part of the font. Take note, that vertical mirror is working with chunks of data in the size of font and aligned with font base (that black “-” symbol in the left of the edit window).



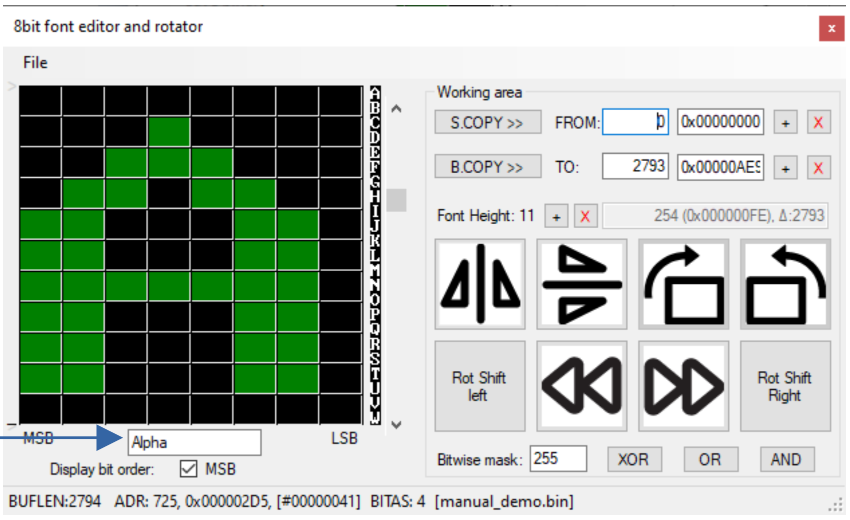
FROM: 0 0x00000000
TO: 2793 0x00000AE
Font Height: 11
254 (0x000000FE), Δ:2793
Bitwise mask: 255
Display bit order: ☑ MSB
BUFLEN:2794 ADR: 725, 0x000002D5, [#00000041] BITAS: 7 [manual_demo.bin]

Now we can save or export the font. But there is another interesting feature.
It is called “font descriptor”. It is an additional data about the font.
To create “font descriptor” select menu item “Create New descriptor”.
The descriptor is dependent on the font size and start/stop values. If font size is changed (or size of the buffer or work area) all the descriptors are deleted. Maybe in future I will create method to recalculate all descriptors. Current software version has this limitation. So, do not change sizes of the font in either way. Create descriptor when the font size and character set are defined.



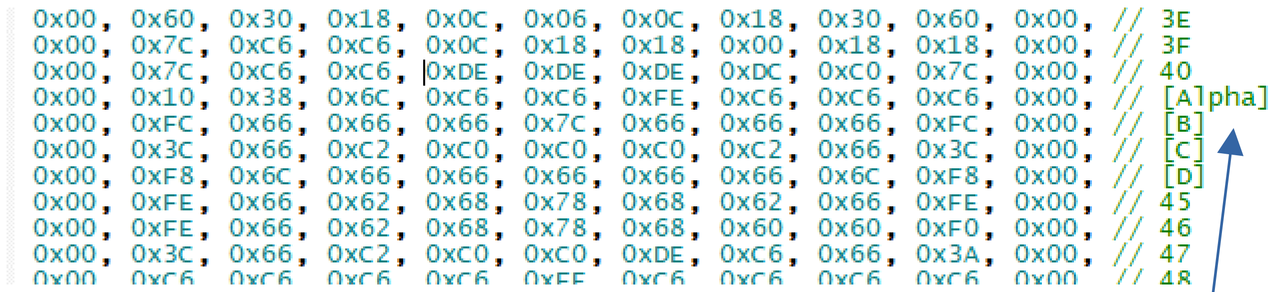
BUFLEN:2794 ADR: 715, 0x000002CB, [#00000041] BITAS: 3 [m

Font descriptor allows to describe each char in the font.
This information is stored in XML file, in the same folder as binary file.
Also, the information about chars are exported to C source.
When loading binary file, program loads XML file and restores font height and descriptor data. There may be some warning if checksum in XML is not equal to new checksum of binary data.



BUFLEN:2794 ADR: 725, 0x000002D5, [#00000041] BITAS: 4 [manual_demo.bin]

This is descriptor
of char “A”. I called this char as “Alpha”



```
0x00, 0x60, 0x30, 0x18, 0x0C, 0x06, 0x0C, 0x18, 0x30, 0x60, 0x00, // 3E
0x00, 0x7C, 0xC6, 0xC6, 0x0C, 0x18, 0x18, 0x00, 0x18, 0x18, 0x00, // 3F
0x00, 0x7C, 0xC6, 0xC6, 0xDE, 0xDE, 0xDE, 0xDC, 0xC0, 0x7C, 0x00, // 40
0x00, 0x10, 0x38, 0x6C, 0xC6, 0xC6, 0xFE, 0xC6, 0xC6, 0xC6, 0x00, // [Alpha]
0x00, 0xFC, 0x66, 0x66, 0x66, 0x7C, 0x66, 0x66, 0x66, 0xFC, 0x00, // [B]
0x00, 0x3C, 0x66, 0xC2, 0xC0, 0xC0, 0xC0, 0xC2, 0x66, 0x3C, 0x00, // [C]
0x00, 0xF8, 0x6C, 0x66, 0x66, 0x66, 0x66, 0x66, 0x6C, 0xF8, 0x00, // [D]
0x00, 0xFE, 0x66, 0x62, 0x68, 0x78, 0x68, 0x62, 0x66, 0xFE, 0x00, // 45
0x00, 0xFE, 0x66, 0x62, 0x68, 0x78, 0x68, 0x60, 0x60, 0xF0, 0x00, // 46
0x00, 0x3C, 0x66, 0xC2, 0xC0, 0xC0, 0xDE, 0xC6, 0x66, 0x3A, 0x00, // 47
0x00, 0xC6, 0xC6, 0xC6, 0xC6, 0xFE, 0xC6, 0xC6, 0xC6, 0xC6, 0x00, // 48
```

Example of Generic GCC source file export. When exporting C source, there is no need to save file. All source code is copied to windows clipboard too. Just cancel file save dialog and paste new source code directly to your program.

File descriptor in C language source code file. Chars without descriptor have only serial number.

Currently software can load and save binary files (descriptor XML files are loaded automatically if they are present).
Program can import IntelHEX file and can export to C source code (single or dual dim array), IntelHEX (.hex) file.
In the future versions of the the software, maybe other import/export options will be implemented. The software is under construction, so, bugs are still present and this manual will be slightly different compared with the last version of available software.