



Font Converter

User's Manual **V3.18**

Micrium

For the Way Engineers Work

Micrium
1290 Weston Road, Suite 306
Weston, FL 33326
USA

www.Micrium.com

Designations used by companies to distinguish their products are often claimed as trademarks. In all instances where Micrium Press is aware of a trademark claim, the product name appears in initial capital letters, in all capital letters, or in accordance with the vendor's capitalization preference. Readers should contact the appropriate companies for more complete information on trademarks and trademark registrations. All trademarks and registered trademarks in this book are the property of their respective holders.

Copyright © 2011 by Micrium except where noted otherwise. All rights reserved. Printed in the United States of America. No part of this publication may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the publisher; with the exception that the program listings may be entered, stored, and executed in a computer system, but they may not be reproduced for publication.

The programs and code examples in this book are presented for instructional value. The programs and examples have been carefully tested, but are not guaranteed to any particular purpose. The publisher does not offer any warranties and does not guarantee the accuracy, adequacy, or completeness of any information herein and is not responsible for any errors and omissions. The publisher assumes no liability for damages resulting from the use of the information in this book or for any infringement of the intellectual property rights of third parties that would result from the use of this information.

USER'S MANUAL VERSIONS

This manual describes the latest software version. If any error occurs, please inform us and we will try to assist you as soon as possible.

For further information on topics or routines not yet specified, please contact us.

SW version / manual revision	Date	By	Description
3.18/00	110527	AS	a) New functions to change the font height added.
3.16/00	110106	AS	a) Merging of font files added. b) Micrium Logo replaced with the new one.
3.14/01	100813	AS	a) Table of command line options reworked.
3.14/00	100104	JE	a) Extended antialiased formats added. b) 'Pixel' and 'Point' option added to font options dialog. c) Antialiasing options added. d) New command line option 'edit' added.
3.10/03	090610	AS	a) Redesigned manual.
3.10/02	081118	JE	a) Company name changed.
3.10/01	071112	JE	a) Font size explanation added.
3.10/00	070606	JE	a) Framed fonts added.
3.08/00	060215	JE	a) Logging of commands added.
3.06/00	060201	JE	a) XBF format added.
3.04/00	051221	JE	a) New font type added.
3.02/01	050408	JE	a) Magnification options added.
3.00/01	050117	JE	a) Compatibility options added. b) Opening 'C' files added.
2.18/01	030716	JE	a) System independent fonts added.
2.16/02	030409	JE	a) UConvert removed.

SW version / manual revision	Date	By	Description
2.16/01	020812	JE	a) Version number incremented.
2.14/02	020109	JE	a) Chapter 2.4 changed to 3.
2.14/01	011213	JE	a) 2bpp antialiasing mode added
2.12/01	010402	JE	a) Explanation of antialiasing modes added. b) Description of encoding modes inserted. c) Shift JIS support of FontCvt explained. d) UConvert: Shift JIS and commandline interface.
2.10/01	010104	JE	a) NT & 9x versions merged
2.02/02	010103	JE	a) Renamed: FontCvt -> FontConvert b) Small changes
2.02/01	001213	JE	a) Complete revised b) Version control table added

Table of Contents

0-1	User's Manual Versions	3
Chapter 1	Introduction	1
Chapter 2	Using μ C/FontConverter	2
2-1	Creating a μ C/GUI font file from a Windows font	2
2-2	Font generation options dialog	5
2-2-1	Type of font to generate	5
2-2-2	Encoding	6
2-2-3	Antialiasing	7
2-3	Font Dialog	8
2-3-1	Font, Font Style, and Size	8
2-3-2	Script	8
2-3-3	Unit of Size	9
2-4	User Interface	9
2-4-1	Selecting the current character	9
2-4-2	Toggling character status	10
2-4-3	Selecting pixels	10
2-4-4	Modifying character bits	10
2-4-5	Operations	11
2-4-6	Modifying the view mode	12
2-5	Options	13
2-6	Saving the font	15
2-6-1	Creating a C file	15
2-6-2	Creating a System Independent Font (SIF)	16
2-6-3	Creating an External Binary Font (XBF)	16
2-7	Modifying an existing C font file	16
2-8	Merging fonts with existing C font files	18

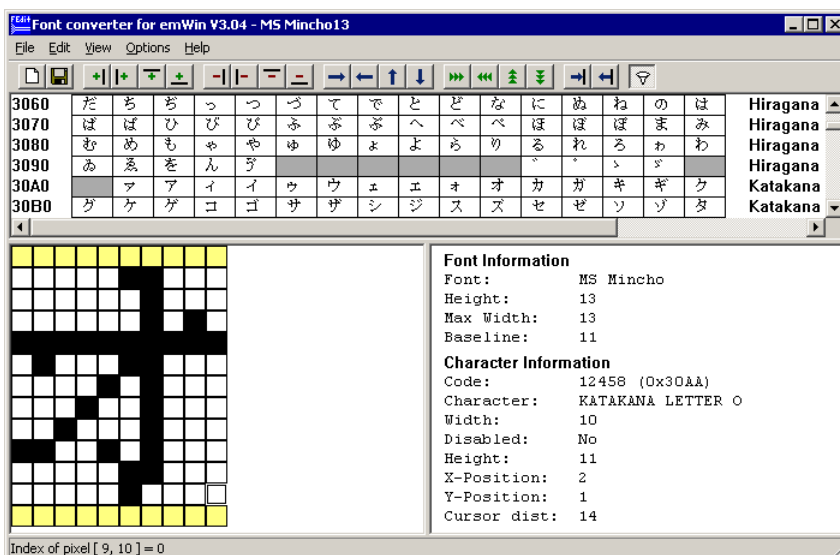
Chapter 3	Pattern files	20
3-1	Creating pattern files using Notepad	20
3-2	Creating pattern files using Font ConvertER	21
3-3	Enabling characters using a pattern file	21
Chapter 4	Supported output modes	22
4-1	Standard mode	23
4-2	Antialiased modes	23
Chapter 5	Command line options	24
5-1	Table of commands	25
5-2	Examples	26
Chapter 6	Examples	27
6-1	Resulting C code, standard mode	27
6-2	Resulting C code, 2 bpp antialiased mode	29
6-3	Resulting C code, 4 bpp antialiased mode	31
6-4	Resulting C code, extended mode	33
Appendix A	µC/FontConverter Licensing Policy	35

Introduction

Fonts which can be used with μ C/GUI should be defined either as GUI_FONT structures in C or should exist as system independent font data. If using C files the structures - or rather the font data which is referenced by these structures - can be rather large. It is very time-consuming and inefficient to generate these fonts manually. We therefore recommend using FontConvert, which automatically generates C files from fonts.

The font converter is a Windows program which is easy to use. Simply load an installed Windows font which is based on TrueType Outlines into the program, edit it if you want or have to, and save it. The C file may then be compiled, allowing the font to be shown on your display with μ C/GUI on demand.

The following is a sample screen shot of the font converter with a font loaded in normal (standard) mode:

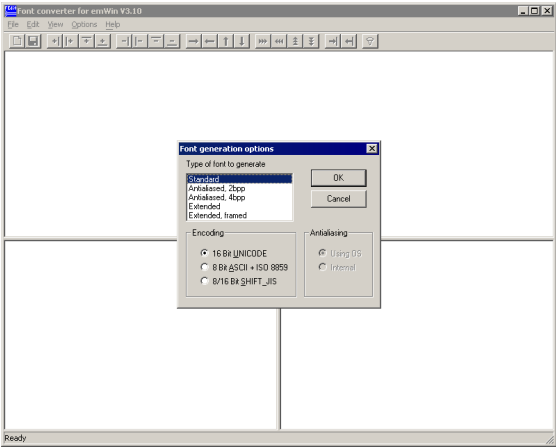


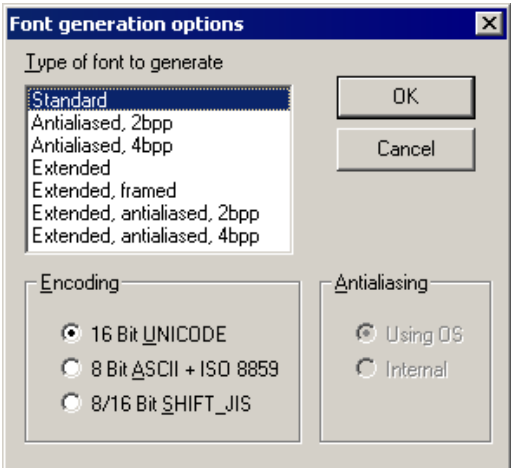
Using μ C/FontConverter

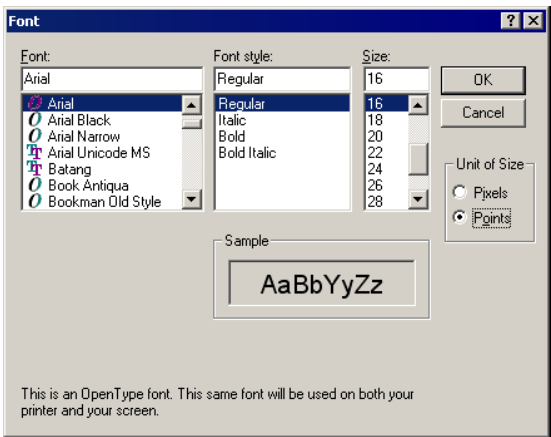
The font converter can create an μ C/GUI font file from an installed Windows font or it can be used to edit the font data of a existing C font file.

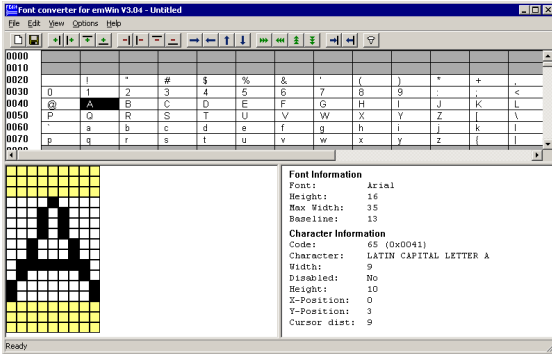
4-1 CREATING A μ C/GUI FONT FILE FROM A WINDOWS FONT

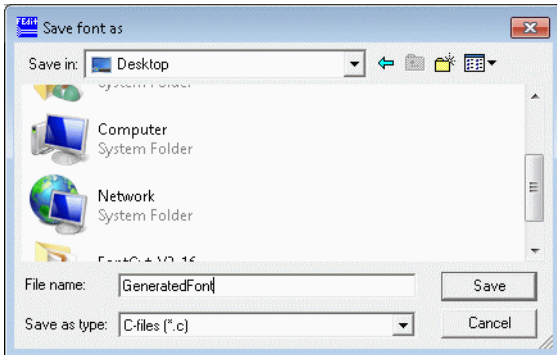
The basic procedure for using μ C/FontConverter for creating a μ C/GUI font file from an installed Windows font is illustrated below. The steps are explained in details in the section that follows.

Step 1	Screenshot
<p>Start the application. The font converter is opened and automatically displays the Font generation options dialog box.</p> <p>The same dialog box appears if File/New is chosen from the font converter menu at any point.</p>	

Step 2	Screenshot
<p>Specify font generation options.</p> <p>In this example, a font is to be generated in extended mode and with 16-bit Unicode encoding. The antialiasing option is irrelevant here since an antialiased mode was not selected.</p> <p>Click OK.</p>	

Step 3	Screenshot
<p>Specify font options.</p> <p>In this example, a regular style, 16 pixel Arial font is chosen.</p> <p>Click OK.</p>	

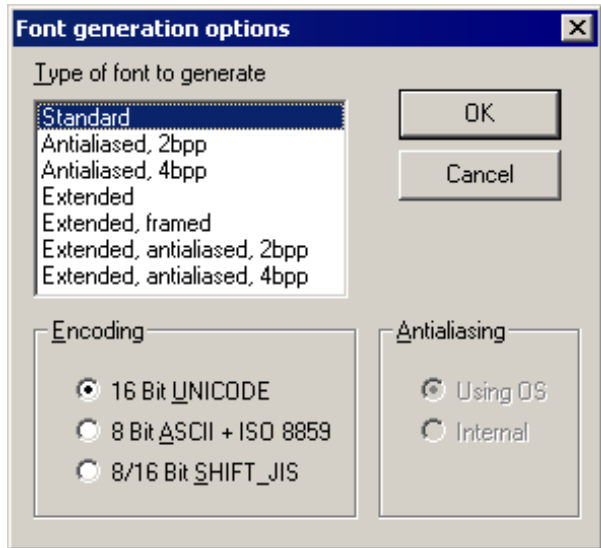
Step 4	Screenshot
<p>Edit the font as necessary.</p> <p>See section "User Interface" for more information on working with the font converter user interface.</p>	

Step 5	Screenshot
<p>Save the μC/GUI font file and choose File/Save As.</p> <p>Select the desired format of the font data file, C file, system independent font or external bitmap font.</p> <p>Select a destination and a name for the font file and click Save.</p> <p>The font converter will create a separate file in the specified destination, containing the currently loaded font data.</p>	

4-2 FONT GENERATION OPTIONS DIALOG

After starting the program or when choosing the menu point File/New, the following dialog automatically occurs:

The selections made here will determine the output mode of the generated font, how it is to be encoded, and how it will be antialiased (if an antialiased output mode is selected).



4-2-1 TYPE OF FONT TO GENERATE

STANDARD

Creates a 1 bit per pixel font without antialiasing.

ANTIALIASED, 2BPP

Creates an antialiased font using 2 bits per pixel.

ANTIALIASED, 4BPP

Creates an antialiased font using 4 bits per pixel.

EXTENDED

Creates a non antialiased 1 bit per pixel font with extended character information. This type of font is required for applications which need support for compound characters like used in Thai language.

EXTENDED, FRAMED

Creates a non antialiased 1 bit per pixel font with extended character information with a surrounding frame. A framed font is always drawn in transparent mode regardless of the current settings. The character pixels are drawn in the currently selected foreground color and the frame is drawn in background color. For more details please refer to the μ C/GUI user manual.

EXTENDED, ANTIALIASED, 2BPP

Creates an antialiased 2 bit per pixel font with extended character information. Each character has the same height and its own width. The pixel information is saved with 2bpp antialiasing information and covers only the areas of the glyph bitmaps.

EXTENDED, ANTIALIASED, 4BPP

Creates an antialiased 4 bit per pixel font with extended character information. Each character has the same height and its own width. The pixel information is saved with 4bpp antialiasing information and covers only the areas of the glyph bitmaps.

4-2-2 ENCODING

16 BIT UNICODE

With Unicode encoding, you have access to all characters of a font. Windows font files contain a maximum of 65536 characters. All character codes of the C file are the same as those in the Windows font file.

8 BIT ASCII + ISO 8859

This encoding mode includes the ASCII codes (0x20 - 0x7F) and the ISO 8859 characters (0xA0 - 0xFF).

8/16 BIT SHIFT JIS

Shift JIS (Japanese Industry Standard) enables mapping from Unicode to Shift JIS in accordance with the Unicode standard 2. For example, the Katakana letter “KU” is shifted from its Unicode value of 0x30AF to the Shift JIS value of 0x834E, the Kanji character 0x786F is shifted to 0x8CA5 and so on.

4-2-3 ANTIALIASING

You can choose between two ways of antialiasing. This choice only applies when an antialiased font type has been selected.

USING OS

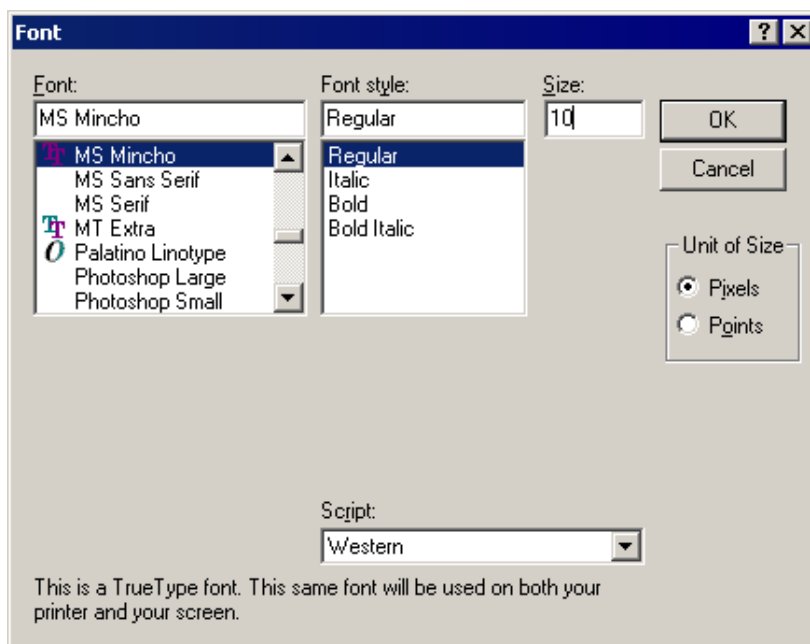
The operating system is used to do the antialiasing. The resulting characters appear exactly the same as in any other windows application where antialiased characters are displayed.

INTERNAL

The internal antialiasing routines of the font converter are used to do the antialiasing. The resulting characters are more exact with regard to proportions.

4-3 FONT DIALOG

After clicking OK in the Font generation options dialog box, a second dialog is displayed as follows:



This is where the font to be converted into a C file is selected. Be sure that you do not violate any copyright laws by converting a font with the font converter.

4-3-1 FONT, FONT STYLE, AND SIZE

These menus are used to select the particular font to be converted. The size of the font is specified in pixels.

4-3-2 SCRIPT

The Script box is used to select the character set which should be mapped down from Unicode into the first 256 characters in accordance with ISO 8859. It only applies when using the 8 Bit ASCII + ISO 8859 encoding mode.

4-3-3 UNIT OF SIZE

This option button can be used to set 'Points' or 'Pixels' as measuring unit. Please note that μ C/GUI does not know something about the unit 'Points' whereas most of other PC applications use the point size for specifying the font size. The font converter uses the operating system for getting the desired font resource. Please note that the font mapper of the operating system is not able to create each font in each desired pixel height. In these cases the font mapper of the operating system creates the nearest possible pixel height. This is not a bug of the font converter.

4-4 USER INTERFACE

After clicking OK in the Font dialog box, the main user interface of the font converter appears, loaded with the previously selected font. You may convert the font into a C file immediately if you wish or edit its appearance first.

The font converter is divided into two areas. In the upper area, all font characters appear scaled 1:1 as they will be displayed on your target device. Disabled characters are shown with a gray background. Per default all character codes which are not included in the chosen font are disabled. For example, many fonts do not include character codes from 0x00 to 0x1F and 0x7F to 0x9F, so these codes are grayed.

The current character is displayed in a magnified scale on the left side of the lower area. Additional information about the font and the current character can be seen on the right side. If you want to modify the character data, you must first activate the lower area, either by pressing the <TAB> key or by simply clicking in the area.

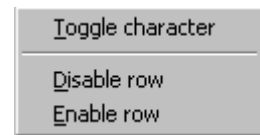
4-4-1 SELECTING THE CURRENT CHARACTER

Characters may be selected:

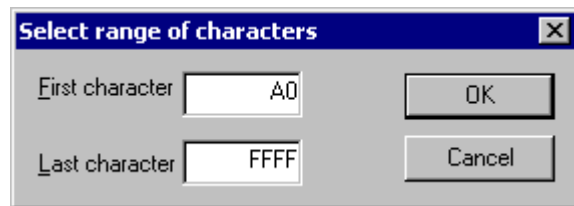
- by using the keys <UP>, <DOWN>, <LEFT>, <RIGHT>, <PGUP>, <PGDOWN>, <POS1>, or <END>;
- by using the scroll bars; or
- by clicking a character with the left mouse button.

4-4-2 TOGGLING CHARACTER STATUS

Use the right mouse button to toggle the status of a specific character or to enable/disable an entire row of characters. The menu point Edit/Toggle activation as well as the <SPACE> key will toggle the status of the current character.



If you need to change the status of a particular range of characters, choose Edit/Enable range of characters or Edit/Disable range of characters from the menu. The range to be enabled or disabled is then specified in a dialog box using hexadecimal character values. To disable all characters, select Edit/Disable all characters from the menu.



4-4-3 SELECTING PIXELS

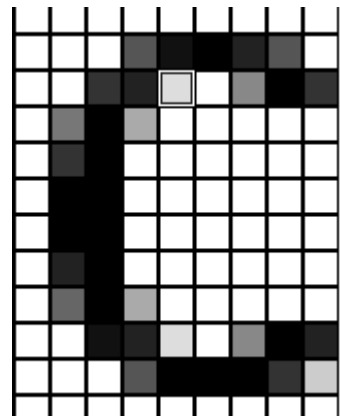
When the lower area of the user interface is activated, you can move through the pixels with the cursor, either by using the <UP>, <DOWN>, <LEFT> and <RIGHT> keys or by clicking on the pixels with the left mouse button.

4-4-4 MODIFYING CHARACTER BITS

In the lower area you can use the <SPACE> key to invert the currently selected bit. In antialiased mode, you can increase and decrease the intensity of a pixel with the keys <+> and <->.

The status bar displays the intensity of the current pixel as follows









Index of pixel [4, 4] = 2



4-4-5 OPERATIONS





SIZE OPERATIONS

The size of a character (the font) may be modified by selecting Edit/Insert/Right, Left, Top, Bottom or Edit/Delete/Right, Left, Top, Bottom from the menu, or by using the toolbar:

-  Add one pixel to the right.
-  Add one pixel to the left.
-  Add one pixel at the top
-  Add one pixel at the bottom
-  Delete one pixel from the right.
-  Delete one pixel from the left
-  Delete one pixel at the top
-  Delete one pixel at the bottom

SHIFT OPERATIONS

Choose Edit/Shift/Right, Left, Up, Down from the menu to shift the bits of the current character in the respective direction, or use the toolbar:

-  Shift all pixels right.
-  Shift all pixels left.
-  Shift all pixels up.
-  Shift all pixels down.

MOVE OPERATIONS (EXTENDED FONT FORMAT ONLY)

Choose Edit/Move/Right, Left, Up, Down from the menu to move the character position in the respective direction, or use the toolbar:



Move image to the right.



Move image to the left.



Move image up.



Move image down.

CHANGE CURSOR DISTANCE (EXTENDED FONT FORMAT ONLY)

Choose Edit/Cursor distance/Increase, Decrease from the menu to move the character position in the respective direction, or use the toolbar:



Increase cursor distance.



Decrease cursor distance.

4-4-6 MODIFYING THE VIEW MODE

The view mode may be changed by selecting the following options from the menu:

VIEW/ALL CHARACTERS

If enabled (standard), all characters are shown. If disabled, only the rows with at least one enabled character are shown.



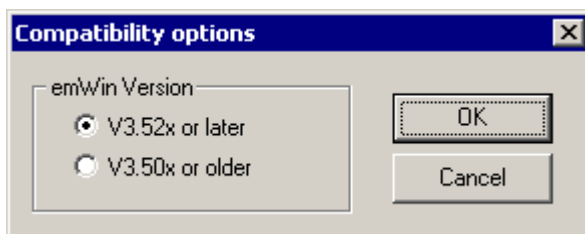
Toggles viewing mode.

4-5 OPTIONS

COMPATIBILITY OPTIONS

The font converter is able to create font files for all versions of μ C/GUI. Because there have been a few small changes of the font format from the μ C/GUI version 3.50 to the version 3.52, the C font files for these versions should be slightly different to avoid compiler warnings or compiler errors.

Use the command Options/Compatibility to get into the following dialog:

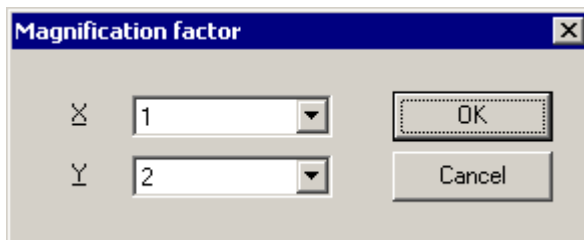


After choosing the desired μ C/GUI version the OK button should be pressed.

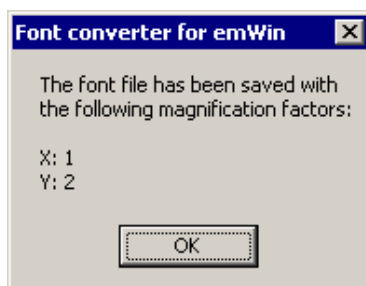
MAGNIFICATION OPTIONS

The font converter is able to save the font data in a magnified format.

Use the command Options/Magnification to get into the following dialog:

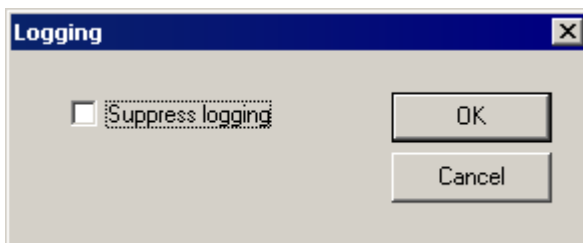


A magnification factor for the X and the Y axis can be specified here. If for example the magnification factor for the Y axis is 2 and the height of the current font data is 18, the font height in the font file will be 36. The magnification in X works similar. After saving the font in a magnified format a short message is shown to inform the user, that the saved font is magnified:



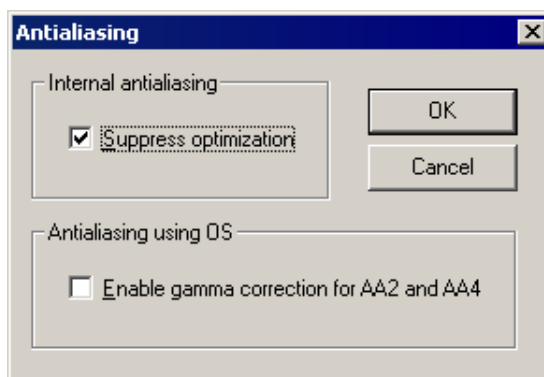
LOGGING

Logging of commands can be enabled or disabled using the command Options/Logging:



When logging is enabled the C files contain a history of the commands which has been used to modify the font file.

ANTIALIASING



When using 'Internal antialiasing' it is recommended to enable Suppress optimization. This makes sure, that the horizontal and vertical alignment of the characters fits to each other.

The option Enable gamma correction for AA2 and AA4 should be disabled. When the option is enabled the antialiased pixels of the characters will appear a little more darker.

4-6 SAVING THE FONT

The font converter can create C font files or system independent font data files. For details about the SIF format please refer to the μ C/GUI documentation.

4-6-1 CREATING A C FILE

When you are ready to generate a C file, simply select File/Save As from the font converter menu, specify a destination and name for the file, choose the C file format and click Save. A C file will automatically be created.

The default setting for the filename is built by the name of the source font and the current height in pixels. For example, if the name of the source font is "Example" and the pixel height is 10, the default filename would be Example10.c. If you keep this default name when generating a C file, the resulting name of the font will be GUI_FontExample10.c. Please see Chapter 6 for examples of C files generated from fonts.

4-6-2 CREATING A SYSTEM INDEPENDENT FONT (SIF)

When you are ready to generate the file, simply select File/Save As from the font converter menu, specify a destination and name for the file, choose the `System independent font` format and click Save. A system independent font file will automatically be created.

This file does not contain C structures which can be compiled with μ C/GUI but binary font data, which can be used as described in the current μ C/GUI documentation.

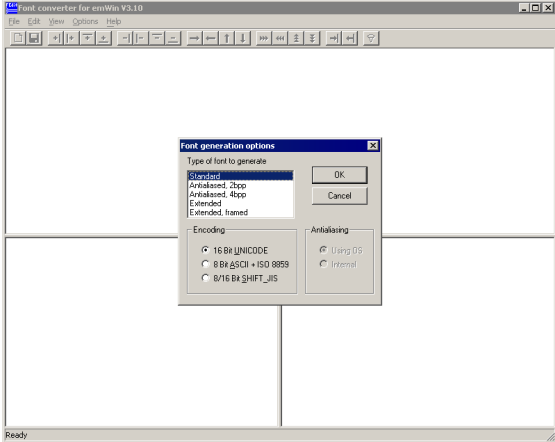
4-6-3 CREATING AN EXTERNAL BINARY FONT (XBF)

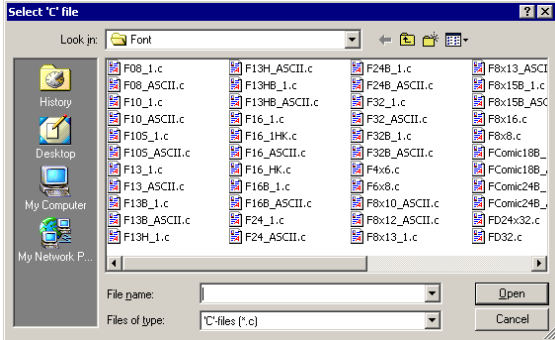
When you are ready to generate the file, simply select File/Save As from the font converter menu, specify a destination and name for the file, choose the `External binary font` format and click Save. An external binary font file will automatically be created.

This file does not contain C structures which can be compiled with μ C/GUI but binary font data, which can be used as described in the current μ C/GUI documentation.

4-7 MODIFYING AN EXISTING C FONT FILE

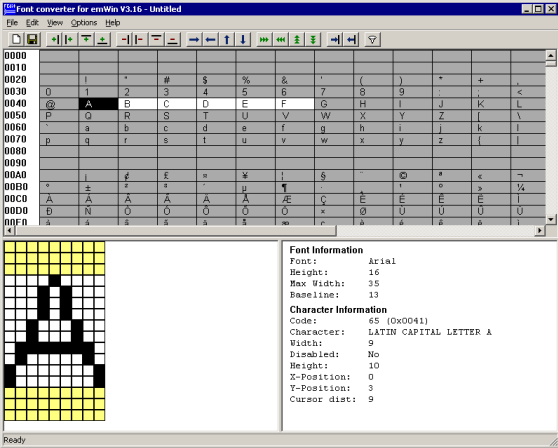
The font converter is able to open existing font files and to modify their font data. The tool can only open C font files generated by the font converter. If the C font files have been modified manually, it can not be guaranteed, that they can be opened by the font converter.

Step 1	Screenshot
<p>Start the application.</p> <p>The font converter is opened and automatically displays the Font generation options dialog box.</p> <p>Press Cancel.</p>	 <p>The screenshot shows the 'Font converter for emWin V3.10' application window. A dialog box titled 'Font generation options' is open in the center. The dialog has a 'Type of font to generate' section with a list: 'Standard' (selected), 'Antialiased', 'Bpp', 'Antialiased_Bpp', 'Extended', and 'Extended_framed'. Below this is an 'Encoding' section with radio buttons for '1684 UNICODE', '8194 ASCII + ISO 8859', and '8176 BA_GHFT_IG5'. To the right of the encoding section is an 'Antialiasing' section with radio buttons for 'Using OS' and 'Internal'. 'OK' and 'Cancel' buttons are at the bottom right of the dialog. The background application window is mostly empty with a menu bar (File, Edit, View, Options, Help) and a toolbar.</p>

Step 2	Screenshot
<p>Use the command File\Load C file.</p> <p>Select the desired C font file to be opened and click OK.</p>	 <p>The screenshot shows a 'Select 'C' file' dialog box. The 'Look in:' field is set to 'Font'. The main area displays a grid of files with icons, including 'F08_1.c', 'F08_ASCII.c', 'F10_1.c', 'F10_ASCII.c', 'F105_1.c', 'F105_ASCII.c', 'F13_1.c', 'F13_ASCII.c', 'F138_1.c', 'F138_ASCII.c', 'F19H_1.c', 'F19H_ASCII.c', 'F13HB_1.c', 'F13HB_ASCII.c', 'F16_1.c', 'F16_1HK.c', 'F16_HK.c', 'F16B_1.c', 'F16B_ASCII.c', 'F24_1.c', 'F24_ASCII.c', 'F24B_1.c', 'F24B_ASCII.c', 'F32_1.c', 'F32_ASCII.c', 'F32B_1.c', 'F32B_ASCII.c', 'F4x6.c', 'F6x8.c', 'F8x10_ASCII.c', 'F8x12_ASCII.c', 'F8x13_1.c', 'F8x13_ASCII.c', 'F8x15B_1.c', 'F8x15B_ASC', 'F8x16.c', 'F8x8.c', 'FComic18B_', 'FComic18B_', 'FComic24B_', 'FComic24B_', 'FD24x32.c', and 'FD32.c'. At the bottom, there is a 'File name:' field, a 'Files of type:' dropdown set to 'C'-files (*.c)', and 'Open' and 'Cancel' buttons.</p>

4-8 MERGING FONTS WITH EXISTING C FONT FILES

The Font Converter is able to add the content of an existing C font file to the current font data. Once a font is loaded via “File” -> “Load 'C' file...” or created by “File” -> “New” a C font file can be merged to it using “File” -> “Merge 'C' file...”. The Font Converter requires the fonts to be of the same size, so the merging can be processed properly.

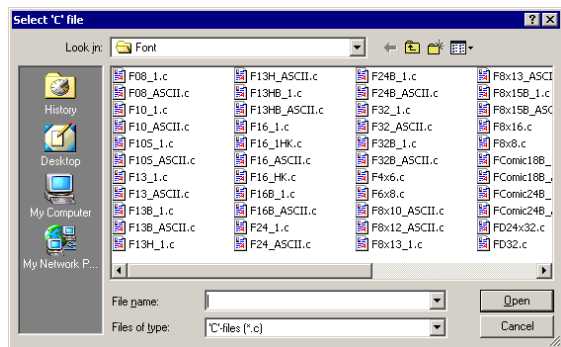
Step 1	Screenshot
<p>Load an existing font or create a new one as described above.</p> <p>In this example the existing font contains the characters A-F (0x41 - 0x46).</p>	

Step 2

Screenshot

Use the command **File\Merge C file....**

Select the desired C font file to be merged and click **OK**.

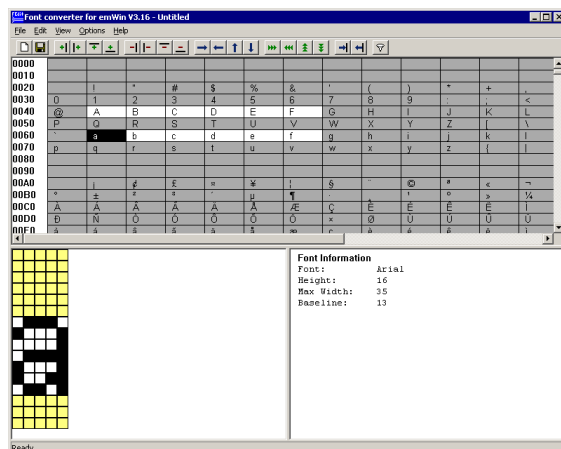


Step 3

Screenshot

The merged font file contains the characters a-f (0x61 - 0x66).

Now the font can be edited and saved as a new font file.



Pattern files

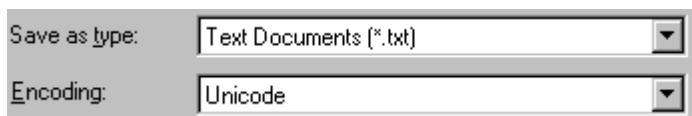
If you need to create fonts with a special set of characters (often for displaying a specific text), it can be very time consuming to enable every character by hand. In these cases, pattern files can be used to enable your character codes.

A pattern file is nothing but a simple text file which contains the characters to be included in the font file. It can be used by FontConvert to enable only the characters you need.

3-1 CREATING PATTERN FILES USING NOTEPAD

One option for creating a pattern file is to use Notepad, part of the WindowsNT accessories:

- Copy the text you want to display into the clipboard.
- Open Notepad.exe.
- Insert the contents of the clipboard into the Notepad document.
- Use Format/Font to choose a font which contains all characters of the text. You can skip this step if you do not want to see the characters.
- Use File/Save As to save the pattern file. It is very important that you save the file in text format:



3-2 CREATING PATTERN FILES USING FONT CONVERTER

A pattern file may also be created directly in FontConvert. Select Edit/Save pattern file from the menu to create a text file which includes all currently enabled characters.

3-3 ENABLING CHARACTERS USING A PATTERN FILE

It is usually helpful to begin by disabling all characters. Select Edit/Disable all characters from the menu if you need to do so.







Now choose Edit/Read pattern file. After opening the appropriate pattern file, all characters included in the file are enabled. If the pattern file contains characters which are not included in the currently loaded font, a message box will appear.

Supported output modes

There are three modes supported by FontConvert: standard, 2-bit antialiased and 4-bit antialiased. If you are using a black and white LCD display, only the standard mode makes sense. If using a grayscale or color display, it is possible to improve the appearance of a font through antialiasing.

Antialiasing smoothes curves and diagonal lines by blending the background color with that of the foreground. The higher the number of shades used between background and foreground colors, the better the antialiasing result. The general purpose of using antialiased fonts is to improve the appearance of text. While the effect of using high-quality antialiasing will be more visually pleasing than low-quality, computation time and memory consumption will increase proportionally. Low-quality (2bpp) fonts require twice the memory of nonantialiased (1bpp) fonts; high-quality (4bpp) fonts require four times the memory.

The following table shows the difference between the modes by displaying the magnified character C in each:

Font Type	Black On White	White On Black
Standard (no antialiasing) 1 bpp 2 shades		
Low-quality (antialiased) 2 bpp 4 shades		
High-quality (antialiased) 4 bpp 16 shades		

5-1 STANDARD MODE

When using this mode, a pixel can either be set or not. The memory requirement for one pixel is one bit. If a pixel is set, it is displayed in the current foreground color.

5-2 ANTIALIASED MODES

These modes are recommended if you want to display characters with smoothed edges. Every pixel is stored as a 2- or 4-bit value which describes the foreground intensity. For example, when using 4-bit antialiasing, a value of 15 displays the pixel in the current foreground color. An intensity of 10 means that the pixel color is a mixture of 10 shares of foreground color and 5 shares of background color.

Before using one of these modes, the feature must be activated in your operating system. Choose the effects sheet of the display properties dialog and activate Smooth edges of screen fonts.

Chapter

5

Command line options

5-1 TABLE OF COMMANDS

The following table shows the available command line options:

Command	Description
create<FONT-NAME>,<STYLE>,<HEIGHT>,<TYPE>,<ENCODING>[,<METHOD>]	<p>Create font:</p> <p><FONTNAME> Name of the font to be used</p> <p><STYLE></p> <p>REGULAR - Creates a normal font</p> <p>BOLD - Creates a bold font</p> <p><HEIGHT> Height in pixels of the font to be created</p> <p><TYPE></p> <p>STD - Standard 1 bpp font</p> <p>AA2 - Antialiased font (2bpp)</p> <p>AA4 - Antialiased font (4bpp)</p> <p>EXT - Extended font</p> <p><ENCODING></p> <p>UC16 - 16 bit Unicode encoding</p> <p>ISO8859 - 8 bit ASCII + ISO8859</p> <p>JIS - Shift JIS</p> <p><METHOD></p> <p>OS - Antialiasing of operating system (default)</p> <p>INTERNAL - Internal antialiasing method</p>
edit<ACTION>,<DETAIL>[,<CNT>]	<p>Equivalent to the 'Edit' menu:</p> <p><ACTION></p> <p>DEL - Deletes pixels</p> <p>INS - Inserts pixels</p> <p><DETAIL></p> <p>TOP - Delete/insert from top</p> <p>BOTTOM - Delete/insert from bottom</p> <p><CNT></p> <p>Number of operations, default is 1</p>
enable[FIRST-LAST>,<STATE>]	<p>Enables or disables the given range of characters:</p> <p><FIRST-LAST> Hexadecimal values separated by a '-' defining the range of characters</p> <p><STATE></p> <p>1 - Enables the given range</p> <p>0 - Disables the given range</p>

Table 5.1:

Command	Description
exit	Exits the application after the job is done
merge<FILENAME>	Merges the given 'C' file to the current content.
readpattern<FILENAME>	Reads a pattern file: <FILENAME> Name of the pattern file to be read
saveas<FILE-NAME>,<TYPE>	Saves the font data as 'C' file or 'SIF' font file: <FILENAME> File name including extension <TYPE> C - Saves as 'C' file SIF - Saves as System independent font file
?	Shows all available commands

Table 5.1:

- All commands are processed from left to right.
- If using -exit Font Converter will stop execution if any error occurs. The return code in this case is != 0.

5-2 EXAMPLES

Creates an extended bold font of 32 pixels height with Unicode encoding using the font "Cordia New":

```
FontCvt -create"Cordia New",BOLD,32,EXT,UC16
```

Reads the C font file "FontFile.c", disables all characters and reads a pattern file:

```
FontCvt FontFile.c -enable0-ffff,0 -readpattern"data.txt"
```


These sections provide examples of C files generated by the font converter in standard, 2bpp antialiased and 4bpp antialiased modes, respectively.

6-1 RESULTING C CODE, STANDARD MODE

The following is an example of a C file in standard mode:

```
/*
  C-file generated by Font converter for emWin version 3.04
  Compiled:          Dec 13 2005 at 12:51:50
  C-file created:   Dec 21 2005 at 12:42:57

  Copyright (C) 1998-2005
  Segger Microcontroller Systeme GmbH
  www.segger.com

  Solutions for real time microcontroller applications

  Source file: Sample10.c
  Font:        Arial
  Height:      10
*/

#include "GUI.H"
#ifndef GUI_CONST_STORAGE
#define GUI_CONST_STORAGE const
#endif

/* The following line needs to be included in any file selecting the
   font. A good place would be GUIConf.H
*/

extern GUI_CONST_STORAGE GUI_FONT GUI_FontSample10;

/* Start of unicode area <Basic Latin> */

GUI_CONST_STORAGE unsigned char acFontSample10_0041[10] = { /* code
0041 */
```

```

_____
  X
  X X
  X X
  X X
  X X
  X X
  XXXXX
  X X
  X X
_____};

```

```
GUI_CONST_STORAGE unsigned char acFontSample10_0061[10] = { /* code
0061 */
```

```

_____
_____
_____
  XXX
  X X
  XXXX
  X X
  X XX
  XX X
_____};

```

```
GUI_CONST_STORAGE GUI_CHARINFO GUI_FontSample10_CharInfo[2] = {
  { 8, 8, 1, acFontSample10_0041 } /* code 0041 */
, { 6, 6, 1, acFontSample10_0061 } /* code 0061 */
};
```

```
GUI_CONST_STORAGE GUI_FONT_PROP GUI_FontSample10_Prop2 = {
  97 /* first character */
, 97 /* last character */
, &GUI_FontSample10_CharInfo[1] /* address of first character */
, (GUI_CONST_STORAGE GUI_FONT_PROP*)0 /* pointer to next
GUI_FONT_PROP */
};
```

```
GUI_CONST_STORAGE GUI_FONT_PROP GUI_FontSample10_Prop1 = {
  65 /* first character */
, 65 /* last character */
, &GUI_FontSample10_CharInfo[0] /* address of first character */
, &GUI_FontSample10_Prop2 /* pointer to next GUI_FONT_PROP */
};
```

```
GUI_CONST_STORAGE GUI_FONT GUI_FontSample10 = {
    GUI_FONTTYPE_PROP /* type of font */
    ,10                /* height of font */
    ,10                /* space of font y */
    ,1                 /* magnification x */
    ,1                 /* magnification y */
    ,&GUI_FontSample10_Prop1
};
```

6-2 RESULTING C CODE, 2 BPP ANTIALIASED MODE

The following is an example of a C file in 2 bpp antialiased mode:

```
/*
C-file generated by Font converter for emWin version 3.04
Compiled:      Dec 13 2005 at 12:51:50
C-file created: Dec 21 2005 at 12:42:57

Copyright (C) 1998-2005
Segger Microcontroller Systeme GmbH
www.segger.com

Solutions for real time microcontroller applications

Source file: Sample10.c
Font:      Arial
Height:    14
*/

#include "GUI.H"

#ifndef GUI_CONST_STORAGE
#define GUI_CONST_STORAGE const
#endif

/* The following line needs to be included in any file selecting the
font. A good place would be GUIConf.H
*/

extern GUI_CONST_STORAGE GUI_FONT GUI_FontSample10;

/* Start of unicode area <Basic Latin> */
```

```
GUI_CONST_STORAGE unsigned char acFontSample10_0041[ 28] = { /* code
0041 */
    0x00, 0x00,
    0x00, 0x00,
    0x00, 0x00,
    0x0B, 0xC0,
    0x1F, 0xD0,
    0x2E, 0xE0,
    0x3C, 0xF0,
    0x78, 0xB4,
    0xBF, 0xF8,
    0xE0, 0x78,
    0xE0, 0x3C,
    0x00, 0x00,
    0x00, 0x00,
    0x00, 0x00
};

GUI_CONST_STORAGE unsigned char acFontSample10_0061[ 28] = { /* code
0061 */
    0x00, 0x00,
    0x00, 0x00,
    0x00, 0x00,
    0x00, 0x00,
    0x00, 0x00,
    0x6F, 0x40,
    0x93, 0xC0,
    0x2B, 0xC0,
    0xB7, 0xC0,
    0xF7, 0xC0,
    0x7B, 0xC0,
    0x00, 0x00,
    0x00, 0x00,
    0x00, 0x00
};

GUI_CONST_STORAGE GUI_CHARINFO GUI_FontSample10_CharInfo[2] = {
    { 8, 8, 2, acFontSample10_0041 } /* code 0041 */
    , { 6, 6, 2, acFontSample10_0061 } /* code 0061 */
};

GUI_CONST_STORAGE GUI_FONT_PROP GUI_FontSample10_Prop2 = {
    0x0061 /* first character */
    , 0x0061 /* last character */
    , &GUI_FontSample10_CharInfo[ 1] /* address of first character */
    , (GUI_CONST_STORAGE GUI_FONT_PROP*)0 /* pointer to next
GUI_FONT_PROP */
};

GUI_CONST_STORAGE GUI_FONT_PROP GUI_FontSample10_Prop1 = {
    0x0041 /* first character */
    , 0x0041 /* last character */
    , &GUI_FontSample10_CharInfo[ 0] /* address of first character */
    , &GUI_FontSample10_Prop2 /* pointer to next GUI_FONT_PROP */
};
```

```
GUI_CONST_STORAGE GUI_FONT GUI_FontSample10 = {
    GUI_FONTTYPE_PROP_AA2 /* type of font */
    ,14 /* height of font */
    ,14 /* space of font y */
    ,1 /* magnification x */
    ,1 /* magnification y */
    ,&GUI_FontSample10_Prop1
};
```

6-3 RESULTING C CODE, 4 BPP ANTIALIASED MODE

The following is an example of a C file in 4 bpp antialiased mode:

```
/*
C-file generated by Font converter for emWin version 3.04
Compiled:      Dec 13 2005 at 12:51:50
C-file created: Dec 21 2005 at 12:42:57

Copyright (C) 1998-2005
Segger Microcontroller Systeme GmbH
www.segger.com

Solutions for real time microcontroller applications

Source file: Sample10.c
Font:      Arial
Height:    10
*/

#include "GUI.H"

#ifndef GUI_CONST_STORAGE
#define GUI_CONST_STORAGE const
#endif

/* The following line needs to be included in any file selecting the
font. A good place would be GUIConf.H
*/

extern GUI_CONST_STORAGE GUI_FONT GUI_FontSample10;

/* Start of unicode area <Basic Latin> */
```

```

GUI_CONST_STORAGE unsigned char acFontSample10_0041[ 40] = { /* code
0041 */
    0x00, 0x00, 0x00, 0x00,
    0x00, 0xCF, 0xF2, 0x00,
    0x03, 0xFF, 0xF6, 0x00,
    0x09, 0xFB, 0xFB, 0x00,
    0x0E, 0xE2, 0xFE, 0x00,
    0x5F, 0x90, 0xCF, 0x40,
    0xBF, 0xFF, 0xFF, 0x90,
    0xFC, 0x00, 0x6F, 0xC0,
    0xF8, 0x00, 0x2F, 0xF2,
    0x00, 0x00, 0x00, 0x00
};

GUI_CONST_STORAGE unsigned char acFontSample10_0061[ 30] = { /* code
0061 */
    0x00, 0x00, 0x00,
    0x00, 0x00, 0x00,
    0x00, 0x00, 0x00,
    0x3D, 0xFE, 0x60,
    0xD3, 0x0F, 0xE0,
    0x29, 0xCF, 0xF0,
    0xDF, 0x4F, 0xF0,
    0xFF, 0x3F, 0xF0,
    0x6F, 0xAF, 0xF0,
    0x00, 0x00, 0x00
};

GUI_CONST_STORAGE GUI_CHARINFO GUI_FontSample10_CharInfo[2] = {
    { 8, 8, 4, acFontSample10_0041 } /* code 0041 */
, { 6, 6, 3, acFontSample10_0061 } /* code 0061 */
};

GUI_CONST_STORAGE GUI_FONT_PROP GUI_FontSample10_Prop2 = {
    0x0061 /* first character */
, 0x0061 /* last character */
, &GUI_FontSample10_CharInfo[ 1] /* address of first character */
, (GUI_CONST_STORAGE GUI_FONT_PROP*)0 /* pointer to next
GUI_FONT_PROP */
};

GUI_CONST_STORAGE GUI_FONT_PROP GUI_FontSample10_Prop1 = {
    0x0041 /* first character */
, 0x0041 /* last character */
, &GUI_FontSample10_CharInfo[ 0] /* address of first character */
, &GUI_FontSample10_Prop2 /* pointer to next GUI_FONT_PROP */
};

GUI_CONST_STORAGE GUI_FONT GUI_FontSample10 = {
    GUI_FONTTYPE_PROP_AA4 /* type of font */
, 10 /* height of font */
, 10 /* space of font y */
, 1 /* magnification x */
, 1 /* magnification y */
, &GUI_FontSample10_Prop1
};

```

6-4 RESULTING C CODE, EXTENDED MODE

```

/*
C-file generated by Font converter for emWin version 3.04
Compiled:      Dec 13 2005 at 12:51:50
C-file created: Dec 21 2005 at 12:45:52

Copyright (C) 1998-2005
Segger Microcontroller Systeme GmbH
www.segger.com

Solutions for real time microcontroller applications

Source file: Arial16.c
Font:      Arial
Height:    16
*/

#include "GUI.H"

#ifndef GUI_CONST_STORAGE
#define GUI_CONST_STORAGE const
#endif

/* The following line needs to be included in any file selecting the
font. A good place would be GUIConf.H
*/

extern GUI_CONST_STORAGE GUI_FONT GUI_Font16;

/* Start of unicode area <Basic Latin> */

GUI_CONST_STORAGE unsigned char acGUI_Font16_0041[ 20] = { /* code
0041 */
    _X_,
    _X_X_,
    _X_X_,
    _X_X_,
    _X_X_,
    _X_X_,
    _X_X_,
    _XXXXXXX_,
    _X_X_,
    _X_X_,
    _X_X_,
    _X_X_};

GUI_CONST_STORAGE unsigned char acGUI_Font16_0061[ 7] = { /* code
0061 */
    _XXX_,
    _X_X_,
    _X_,
    _XXXX_,
    _X_X_,
    _X_XX_,
    _XX_X_};

```

```
GUI_CONST_STORAGE GUI_CHARINFO_EXT GUI_Font16_CharInfo[2] = {
    { 9, 10, 0, 3, 9, acGUI_Font16_0041 } /* code 0041 */
, { 5, 7, 1, 6, 7, acGUI_Font16_0061 } /* code 0061 */
};

GUI_CONST_STORAGE GUI_FONT_PROP_EXT GUI_Font16_Prop2 = {
    0x0061 /* first character */
, 0x0061 /* last character */
, &GUI_Font16_CharInfo[ 1] /* address of first character */
, (GUI_CONST_STORAGE GUI_FONT_PROP_EXT *)0
};

GUI_CONST_STORAGE GUI_FONT_PROP_EXT GUI_Font16_Prop1 = {
    0x0041 /* first character */
, 0x0041 /* last character */
, &GUI_Font16_CharInfo[ 0] /* address of first character */
, &GUI_Font16_Prop2 /* pointer to next GUI_FONT_PROP_EXT */
};

GUI_CONST_STORAGE GUI_FONT GUI_Font16 = {
    GUI_FONTTYPE_PROP_EXT /* type of font */
, 16 /* height of font */
, 16 /* space of font y */
, 1 /* magnification x */
, 1 /* magnification y */
, {&GUI_Font16_Prop1}
, 13 /* Baseline */
, 7 /* Height of lowercase characters */
, 10 /* Height of capital characters */
};
```


Appendix

A

μC/FontConverter Licensing Policy

You need to obtain an “Object Code Distribution License” to embed μC/FontConverter in a product that is sold with the intent to make a profit. Each individual product (*i.e.*, your product) requires its own license, but the license allows you to distribute an unlimited number of units for the life of your product. Please indicate the processor type(s) (*i.e.*, ARM7, ARM9, MCF5272, MicroBlaze, Nios II, PPC, *etc.*) that you intend to use.

For licensing details, contact us at:

Micrium
1290 Weston Road, Suite 306
Weston, FL 33326
USA

Phone +1 954 217 2036
FAX +1 954 217 2037

www.micrium.com
licensing@micrium.com