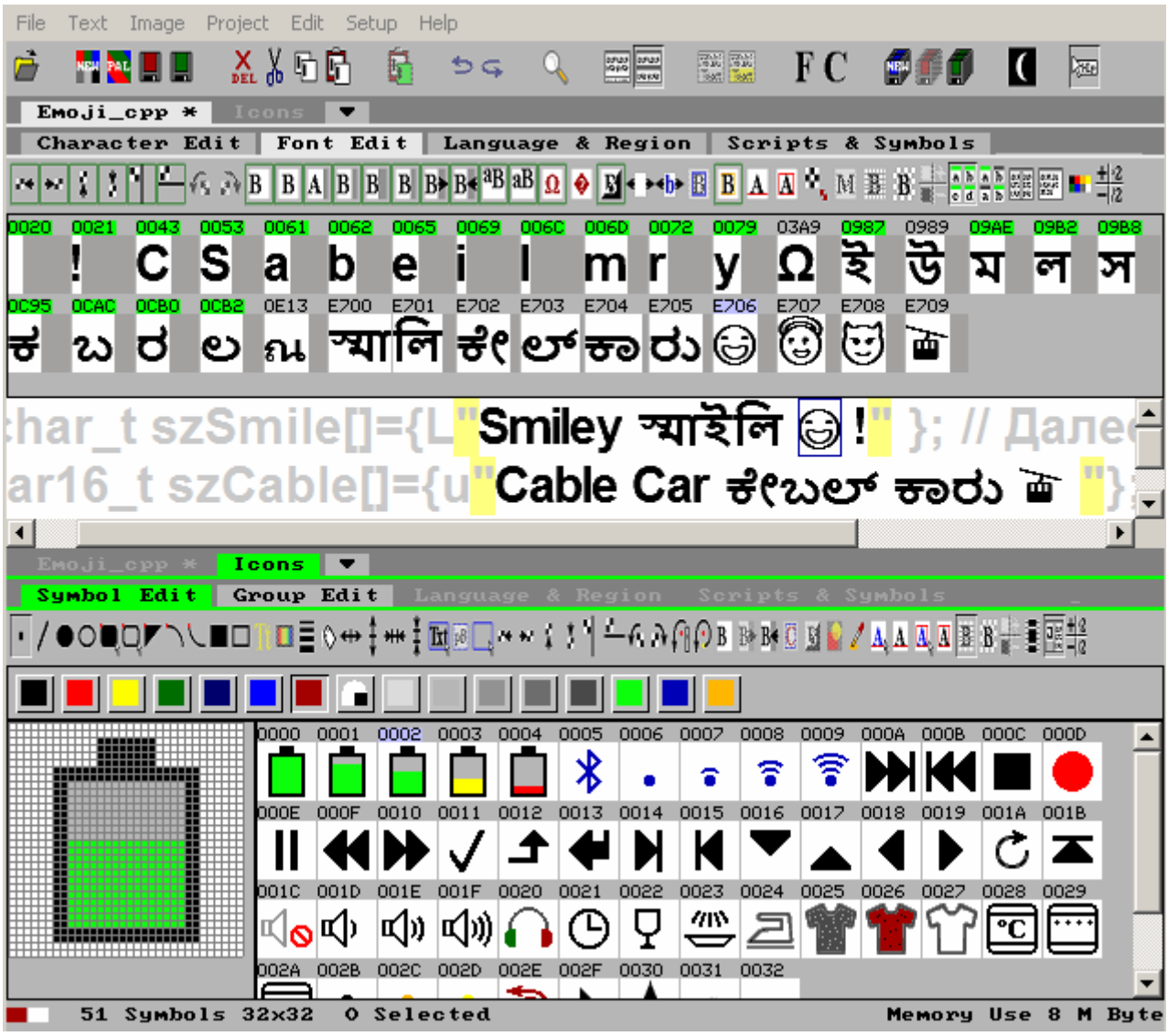
Getting started with

IconEdit

Graphic Font & Symbol Designer for Windows

Quick Guide



10 November 2023

This manual is valid from version 8.3.54 of the IconEdit.

This guide applies to both the Black & White and the Color versions of the IconEdit program. The functionality of the Black&White version is a subset of the Color version, and Grey and Color examples and commands are only valid for the Color version of the program.

DanMagic

Main content

This manual describes how to use the functions of the IconEdit program. A description of the functions and data formats are in the manual *IconEditManual.pdf*.

Disclaimer

The information in documents supplied with the IconEdit tool, and information provided by the IconEdit tool is subject to change without notice. While information contained herein is assumed to be accurate, DanMagic assumes no responsibility for any errors or information.

Copyright Notices, Trademarks, and Immaterial rights

The IconEdit user is obliged to respect all intellectual proprietary rights and copyrights associated with the IconEdit product. See the IconEdit license terms for details.

IconEdit program tool enables the user to use existing graphic designs and fonts installed on the PC as templates during the creation of his/her product designs. The IconEdit user is assumed to oblige and respect all copyrights and trademark rights associated with such material.

With respect to fonts installed on the PC and used as part of the tool users new product creation, this is normally considered "fair use" of a PC font. However if you are in doubt it is recommended to use Open Fonts and Openclipart which may be downloaded from the internet.

"Fair use" is a doctrine in the law of the United States that permits limited use of copyrighted material without having to first acquire permission from the copyright holder.

How to view the document

The PDF file viewer should be set to display the document 1:1 - usually called 100% - to avoid distortion the images.

Contents Overview *Click on item to JUMP*

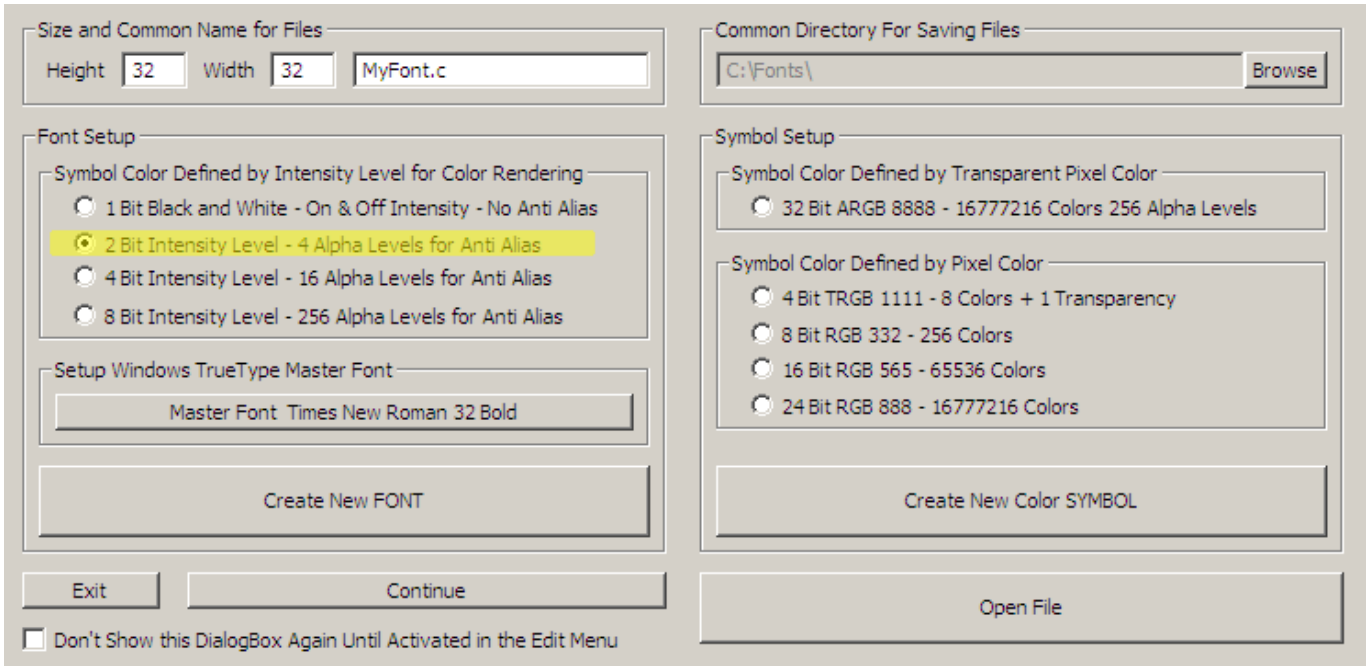
Contents Full *Click on item to JUMP*

Indhold

00: Animated demos for making C-source code fonts in 2 minutes or less

Basic Assumption: *All examples in this manual assume that IconEdit is reset to factory defaults except for continuations.*

The demos show each simple step with yellow highlight for the action in that step:



The demos comes in two groups, bitmap fonts based on a font from the **EuropeanFont** library and bitmap fonts based on a **Windows** vector font.

The demos also show how to make each font or process in batch.

See also other and more detailed descriptions and instructions in the **"How to"** chapters.

A: Fonts based on the EuropeanFonts library

1: Multi Language Fonts from Scratch

Set up an EuropeanFonts library master font and create fonts for many languages just by a few clicks.

https://www.ramtex.dk/iconedit/how-to-make-a-multi-language-font-from-library.htm

2: Text Optimized Fonts from Master Font

IconEdit can find texts in C-strings and make fonts with only the characters found.

https://www.ramtex.dk/iconedit/how-to-make-a-text-optimized-font-from-library.htm

3: Middle Eastern Fonts from Master Font

Middle Eastern texts are written from right to left (RtL) but stored in C-strings left to right (RtL). IconEdit can find texts in C-strings, make fonts with presentation characters, and reverse the C-string texts for use on normal displays.

https://www.ramtex.dk/iconedit/how-to-make-an-arabic-text-optimized-font-from-library.htm

4: Reduce a Master Font to a Multi Language Font

Drag and Drop an EuropeanFonts library master font on IconEdit and reduce the font to a few languages by just a few clicks.

https://www.ramtex.dk/iconedit/how-to-reduce-to-a-multi-language-font-from-library.htm

5: Reduce a Master Font to a Text Optimized Font

IconEdit can find texts in C-strings and mark characters in a font with the characters found.

https://www.ramtex.dk/iconedit/how-to-reduce-to-a-text-optimized-font-from-library.htm

B: Fonts based on Windows fonts

1: Multi Language Fonts from Scratch

Set up a Windows master font and create fonts for many languages just by a few clicks.

https://www.ramtex.dk/iconedit/how-to-make-a-multi-language-font.htm

2: Text Optimized Fonts from Windows font

IconEdit can find texts in C-strings and make fonts with only the characters found.

https://www.ramtex.dk/iconedit/how-to-make-a-text-optimized-font.htm

3: Middle Eastern Fonts from Windows font

Middle Eastern texts are written from right to left (RtL) but stored in C-strings left to right (RtL). IconEdit can find texts in C-strings, make fonts with presentation characters, and reverse the C-string texts for use on normal displays.

https://www.ramtex.dk/iconedit/how-to-make-an-arabic-text-optimized-font.htm

4: South East Asian Fonts from Windows font

South East Asian texts are written as a combination of basic characters and diacritics that has to be combined before they can be displayed. IconEdit can find texts in C-strings, make fonts with combined characters and modify the C-string texts for use on normal displays.

https://www.ramtex.dk/iconedit/how-to-make-an-indian-text-optimized-font.htm

5: Emoji Fonts from Windows font

Emoji has 20-bit Unicode code points, but emojis in texts are written as a combination of two 16-bit surrogate characters. IconEdit can find 20-bit pseudo code for emojis in C-strings, make fonts with the emojis, and convert the 20-bit pseudo code to two 16-bit surrogate characters.

https://www.ramtex.dk/iconedit/how-to-make-an-emoji-text-optimized-font.htm

C: Data conversions

1: Convert C-string text to UTF-8 text

UTF-8 is a way to encode Unicode texts as 8-bit bytes instead of 16-bit words. UTF-8 has advantages in some cases:

- Your texts are primarily latin characters (ASCII).
- Your old compiler does not understand 16-bit Unicode.

https://www.ramtex.dk/iconedit/how-to-make-an-utf-8-text.htm

2: How to Convert Images to C-Source code in Batch

The demo will show you how to convert all images of a certain type in a directory to c-souce code in batch.

The conversion can be made from .BMP, .JPG, and .PNG images to many different color modes .

https://www.ramtex.dk/iconedit/how-to-batch-convert-images.htm

01: Quick Ways of Creating a Prototype Font in IconEdit

Basic Assumption: *All examples in this manual assume that IconEdit is reset to factory defaults except for continuations.*

This chapter will show 3 fast ways to cerate a prototype font:

A: Create a Font for a Language in the Language & Region Window
B: Open a Text or Text Catalogue File to Make a Font Directly
C: Create a Code Point Character List for the Font in the New Dialog Box

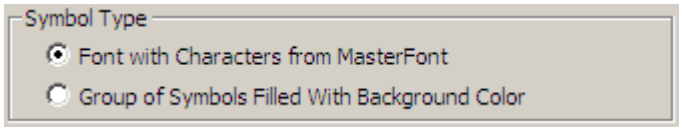
Other and more detailed descriptions and instructions follows in the **"How to"** chapters.

A: Create a Font for a Language in the Language & Region Window:

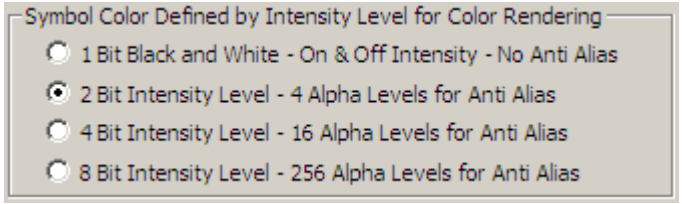
To make a new font press the ***New Font or Symbol Group*** button in the Main Toolbar:

This opens the create dialog box:

Choose how characters or symbols should be organized and shown initially:



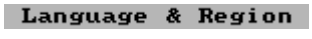
Choose a color format, either Black & White On & Off intensity with fully opaque colors or Intensity Level semi transparent colors for smoothing of edges. This option is only available in the color version of IconEdit.



Press **OK**.

This creates a font with only one character.

Switch to the **Language & Region** window



Scroll down to a language for instance Greek

Choose Greek in the tick box to create the characters necessary for Greek

The font now looks like this:

The small number above each character is the Unicode Code Point in hexadecimal notation. This value will follow the image of the character during all editing, and eventually end up in the Code Point Character List file that is associated with the symbol file when the font is saved.

Your Font is ready for saving....

Press the ***Save All As…*** button in the Main tool bar to save the data as *Greek.c* in the proper directory. The Sym and Code Point Character List files are saved automatically.

B: Open a Text or Text Catalogue File to Make a Font Directly:

To make a new text optimized font to fit a text or text catalogue press the ***Open File*** button in the Main Toolbar:

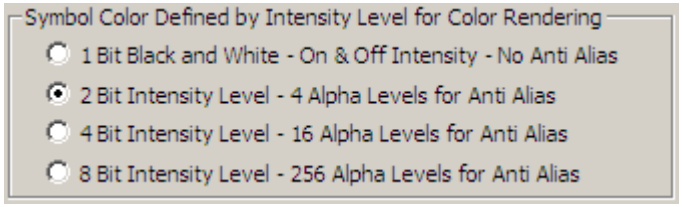
Open the file *ButtonText.cpp*:

```
#define N_TEXT 4
#ifdef LATIN_ALPHABET
wchar_t szText[N_TEXT]={L"Start",
                        L"Stop",
```

```
                          L"Pause",
                          L"Continue"
                         };
#elif CYRILLIC_ALPHABET
wchar_t szText[N_TEXT]={L"Старт",
                          L"Стоп",
                          L"Пауза",
                          L"Далее"
                         };
#endif
```

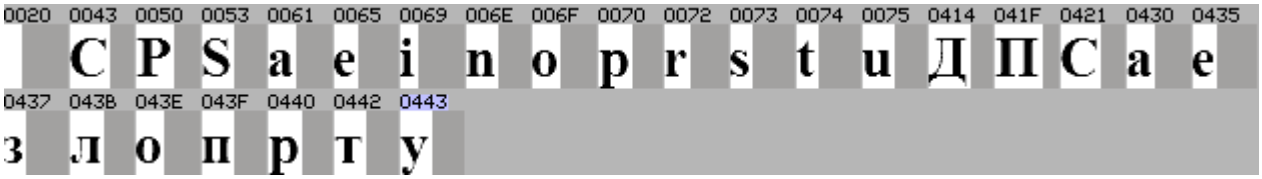
This in turn opens the create font dialog box:

Choose a color format, either Black & White On & Off intensity with fully opaque colors or Intensity Level semi transparent colors for smoothing of edges. This last option is only available in the color version of IconEdit.



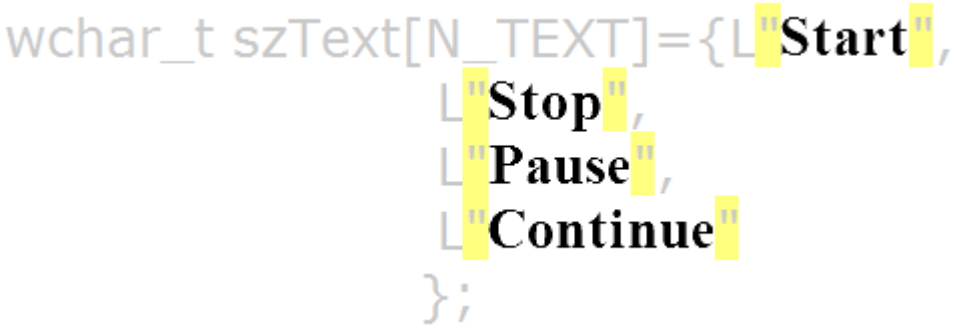
Press **OK**.

The text or text catalogue is analyzed for all occurring characters either in the whole text or, in the case of a C source code text catalogue, only the text inside the text strings to build a font. The resulting font then gets an auto-generated name *ButtonText_cpp.c*:



The small number above each character is the Unicode Code Point in hexadecimal notation. This value will follow the image of the character during all editing, and eventually end up in the Code Point Character List file that is associated with the symbol file when the font is saved.

When the font is created the text is shown with the new font in a separate window:



```
wchar_t szText[N_TEXT]={L"Старт",
                        L"Стоп",
                        L"Пауза",
                        L"Далее"
                       };
```

The blue marking makes it easier to trace the characters in the different windows.

```
Char 0x0443  Cyrillic
CYRILLIC SMALL LETTER U
Text Length 38
```

Mouse help identifies the character and shows how long the text string will be on the target display.

Your Font is ready for saving....

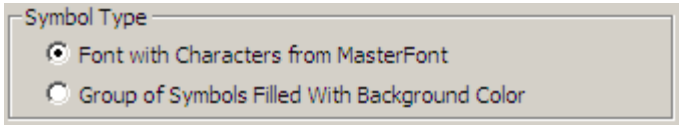
Press the ***Save All As…*** button in the Main tool bar to save the data as *ButtonText_cpp.c* in the proper directory. The Sym and Code Point Character List files are saved automatically.

C: Create a Code Point Character List for a Font in the New Dialog Box:

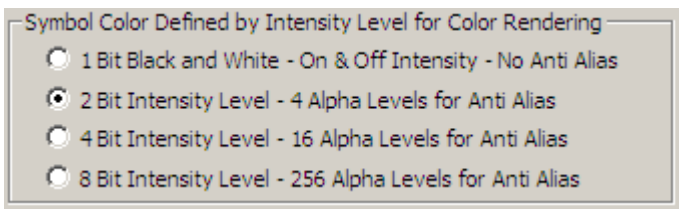
To make a new font press the ***New Font or Symbol Group*** button in the Main Toolbar:

This opens the create dialog box:

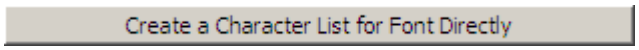
Choose how characters or symbols should be organized and shown initially:



Choose a color format, either Black & White On & Off intensity with fully opaque colors or Intensity Level semi transparent colors for smoothing of edges. This last option is only available in the color version of IconEdit.



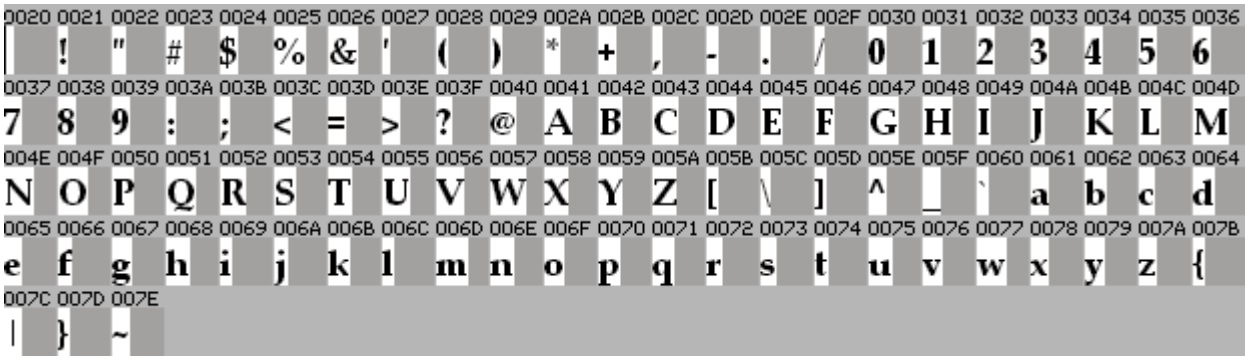
Most applications need the Numbers and the basic Latin characters in the range from 0x20 to 0x7E, this is known as the ASCII range. To make them open the Character List dialog box:



Mark SPACE with a mouse click and mark the character range from SPACE to TILDE with a shift mouse click on TILDE. The characters are highlighted in orange to indicate insert:

Press **OK**.

The font now looks like this:



The small number above each character is the Unicode Code Point in hexadecimal notation. This value will follow the image of the character during all editing, and eventually end up in the Code Point Character List file that is associated with the symbol file when the font is saved.

Your Font is ready for saving....

Press the ***Save All As…*** button in the Main tool bar to save the data as *ASCII.c* in the proper directory. The Sym and Code Point Character List files are saved automatically.

D: Quality Improvement:

Such auto-generated fonts are usually sufficient in the development and debug phase of a project, but before the first release of the project all characters should be checked for readability and harmony.

For options for quality improvement, adding languages or character sets and minimizing the ROM footprint of the font see the chapters in the section **How to Build Fonts from Scratch**.

02: Different Ways of Storing Data in IconEdit

A: How to Save your Work without a Valid Licence

1: Formats that Retain all Relevant Data

IconEdit can store data in any B&W or Color format in an IEF file *Name.ief*. IEF is a fast input/output disk file format for temporary storage of symbol data. It contains the same information as the normal c-sym-cp-pal file quad of the symbol data format, but in a much smaller proprietary format.

Several DataSets can be stored together in an IEP file *ProjectName.iep*. IEP is a container for all the DataSets in a project, it has a general header for the project followed by a number of DataSets in IEF format, all in one format for easy exchange of data with other members of a project group or for working on several projects almost simultaneously.

Data stored in the IEF or IEP formats can later be converted to C-source code and other formats when IconEdit is running under a valid licence.

B&W fonts and symbol groups can be stored as the classic font Glyph Bitmap Distribution Format *Name.bdf*.

2: Save Formats that May Reduce or Change Information

Any B&W or Color format can be stored as a Windows bitmap *Name.bmp*, a JPG image *Name.jpg*, a PNG image *Name.png*, or if it is small enough a Windows Icon *Name.ico* for use in illustrations or presentations.

B: How to Save your Work with a Valid Licence

1: Formats that Retain all Relevant Data

IconEdit normally store data in any B&W or Color format in a set of 2 to 4 separate disk files: A header file *Name.c* that defines the necessary data structures and includes the other files. There is always a pixel data file *Name.sym*. Fonts usually have a Code Point Character List file *Name.cp* with Code Points for the symbols defined in the *sym* file. Color files can have a palette file *Name.pal* with definitions of pixel colors or tool colors. The files are all in ‘C’ source format and can be read by a text editor or included directly in a ‘C’ source code for a program.

IconEdit can store data in any B&W or Color format in an IEF file *Name.ief* or an IEP project file *ProjectName.iep*. The project files are meant for easy exchange of data with other members of a project group or for working on several projects simultaneously. These formats are much smaller than all the other input / output formats in IconEdit, but the data can only be read by the IconEdit program.

B&W font data can be saved as Binary Distribution File *Name.bdf*, this format can be read by most font editors.

B&W data can be saved as binary or ‘C’ files *Name.bin* or *Name.h* these formats can be configured to any of 8 possible byte and bit orientations to fit most B&W displays.

2: Export Formats that May Reduce or Change Information

Any B&W or Color format can be stored as a Windows bitmap *Name.bmp*, a JPG image *Name.jpg*, a PNG image *Name.png*, or if it is small enough a Windows Icon *Name.ico* for use in illustrations or presentations.

The Code Points in a font can be stored as a Unicode text *Name.txt* for use in text editors or as Comma Separated Values *Name.csv* for use in spreadsheets.

C: How to Work with Projects

IconEdit can store a number of DataSets in any mixture of B&W or Color formats in an IEP file *Name.iep*. IEP is a fast input/output disk format, and all the DataSets are stored in one file. This is for use in the beginning of a design project to keep the DataSets together and for exchanging data with other members of a work group.

At a later stage of the design project when the symbols and fonts are included in source code files via the *Name.c* header files the DataSets have to be stored in the *Name.c Name.sym Name.cp Name.pal* file quad.

IconEdit can automatically reload a number of files in any mix of formats known to IconEdit at program start-up, and it can save all changes to the DataSets at program exit.

This reload is achieved with the ***File -> Reload Files at Startup*** option. IconEdit keeps a list of recent files and opens them all at program start if the option is activated.

To make it possible to work on several design projects more or less simultaneously the recent files list can be saved in a *Name.rll* file for later use. The command ***File -> Open All Files in a Previously Saved Reload List...*** can prompt you for a file of the type *Name.rll*, activate the ***File -> Reload Files at Startup*** option and restart IconEdit with the new list of reloadable files.

This means that you can switch between different groups of DataSets with a single command.

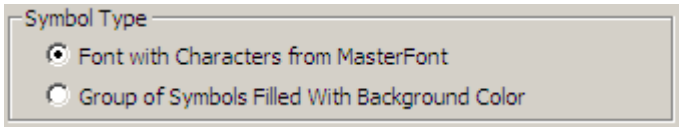
03: Different Ways of Creating Fonts in IconEdit

Basic Assumption: *All examples in this manual assume that IconEdit is reset to factory defaults except for continuations.*

To make a new font press the ***New Font or Symbol Group*** button in the Main Toolbar:

This opens the create dialog box:

Choose how characters or symbols should be organized and shown initially:

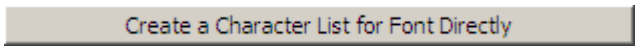


From here there are several ways:

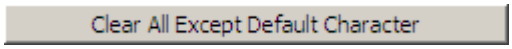
A: Create a Character List in the New Dialog Box:

The New dialog box has several ways of creating a Code Point Character List:

1: Create Manually:



One default character has already been selected, if more than one character is selected use:



This does not clear the default character.

Mark from SPACE to TILDE with the mouse. The characters are highlighted in orange to indicate insert:

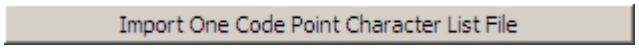
Press Insert:

Press **OK** to leave the New dialog box with the Code Point Character List as Font.

Your Font is ready for saving.

2: Based on Existing Font:

Create Font by Import of a Code Point with a list of characters:



Open NAME.cp and press ***Insert Characters***.

Press **OK** to leave the New dialog box with the Code Point Character List as Font.

Your Font is ready for saving.

B: Language & Region Filters:

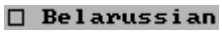
Press OK to leave the New dialog box with only the default character SPACE.

Warning: Language & Regions only works with Master Fonts according to the Unicode standard as described at www.unicode.org. See the chapter **How to Check that an Existing Font is According to Unicode**.

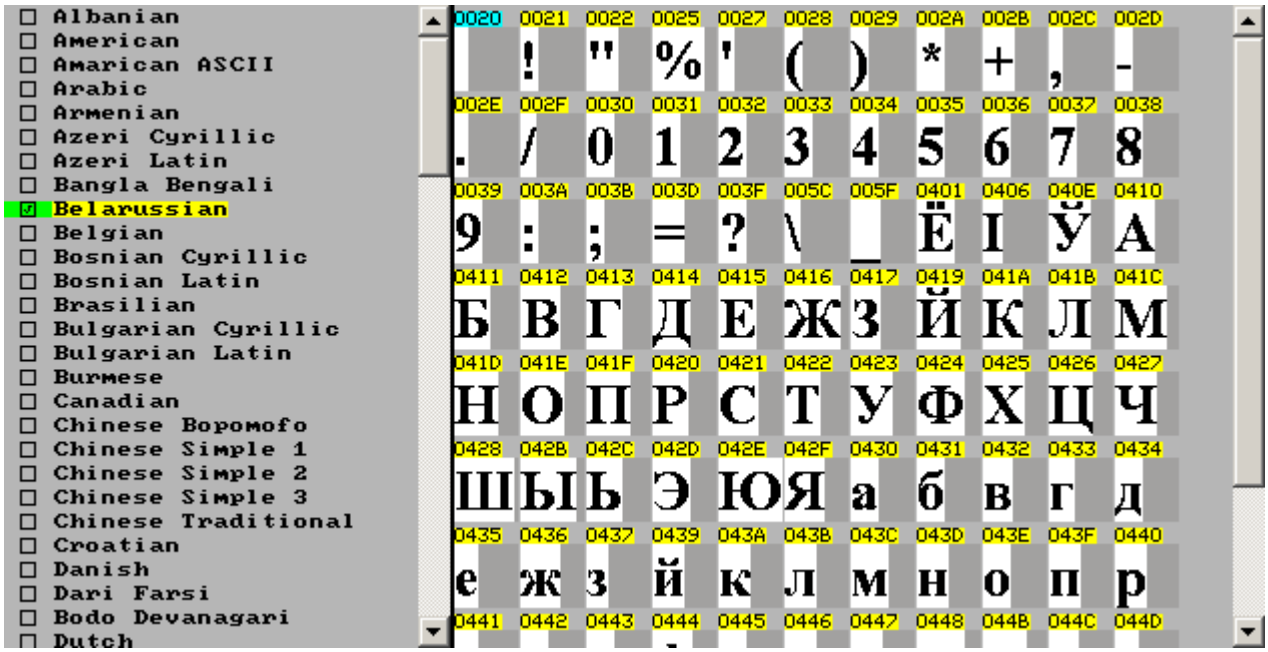
Press ***Language & Region*** Filters:

To enter language edit mode:

Press the tick-box for a language, for example Belarusian:



This highlights the chosen language and adds the missing characters. All the characters included in the language are marked in yellow:



Your Font is ready for saving.

C: Unicode Scripts & Symbols:

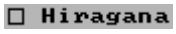
Press OK to leave the New dialog box with only the character SPACE.

Warning: Scripts & Symbols only works with Master Fonts according to the Unicode standard as described at www.unicode.org. See the chapter **How to Check that an Existing Font is According to Unicode**.

Press ***Unicode Scripts & Symbols:***

To enter Unicode edit mode:

Press the tick-box for a script, for example Hiragana:



This highlights the chosen script and adds all the characters in the script. All the characters included in the script are marked in yellow:

Your Font is ready for saving.

D: Read Text, Text Catalogue or Code Point Character List File as Marks:

This uses a file menu command that marks the characters in the font that are already present, and it adds those that are not present in the font from the Master Font. You can build one single Font for any number of input files by importing them one by one in any order.

Press OK to leave the New dialog box with only the character SPACE.

Change to ***Font Edit*** mode:

Fonts can either be made to fit one or more texts, only contain the actual texts in several text catalogues, or based on the Code Point Character List of one or more existing fonts.

1: Font that Fits a Text:

This option interprets Files with the extension **.txt**.

Open a Unicode text such as *Mkhedruli.txt* with **Text –>Import Text or Text Catalogue to Mark or Create Characters**:

სექციების სია [დამალვა]
1 ასტრონომიის დარგები
2 ასტრონომიის ისტორია
3 ასტრონომიის საერთო ცნებები და ტერმინები
3.1 ასტრონომიული ობიექტები
3.2 ასტრონომიული ხელსაწყოები
4 გამოჩენილი ასტრონომები
5 გამოჩენილი ქართველი ასტრონომები
6 ასტრონომიული დაწესებულებები
7 ქართული რესურსები ინტერნეტში

Press **Yes** to **Insert New Additional Characters**.

The program finds all the different Basic Characters, Combined Characters and Presentation Forms including non-displayable control characters, and marks the already present characters with green and the newly added characters from the Master Font with grey:

IconEdit automatically opens the **Show Imported Text** window so you can see how the text will look with the new font:

სექციების სია [დამალვა]
1 ასტრონომიის დარგები
2 ასტრონომიის ისტორია
3 ასტრონომიის საერთო ცნებები და ტერმინები
3.1 ასტრონომიული ობიექტები
3.2 ასტრონომიული ხელსაწყოები
4 გამოჩენილი ასტრონომები
5 გამოჩენილი ქართველი ასტრონომები
6 ასტრონომიული დაწესებულებები
7 ქართული რესურსები ინტერნეტში

Your Font is ready for saving.

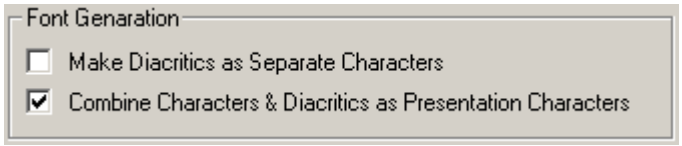
2: Font that Fits a Text Catalogue:

This option interprets Files with the extensions **.c .h .cpp** and **.cs**.

Open the Unicode text catalogue *Astronomy.cpp* with **Text -> Import Text or Text Catalogue to Mark or Create Characters**:

```
// Disclaimer: These texts are only for demonstration of the principle
// and may not make any sense to someone familiar with the language
#ifdef ARABIC
wchar_t szAstronomy_00[]={L"الدراسة العلمية للأجرام"};
wchar_t szAstronomy_01[]={L"مثل النجوم، والكواكب،"};
#elif THAI
wchar_t szAstronomy_00[]={L"ดาราศาสตร์ คือวิชาวิทยา"};
wchar_t szAstronomy_01[]={L"ลาที ดาวฤกษ์ ดาวเคราะห์"};
#endif
```

This opens the **Choose Autogenerated Characters** dialogbox:

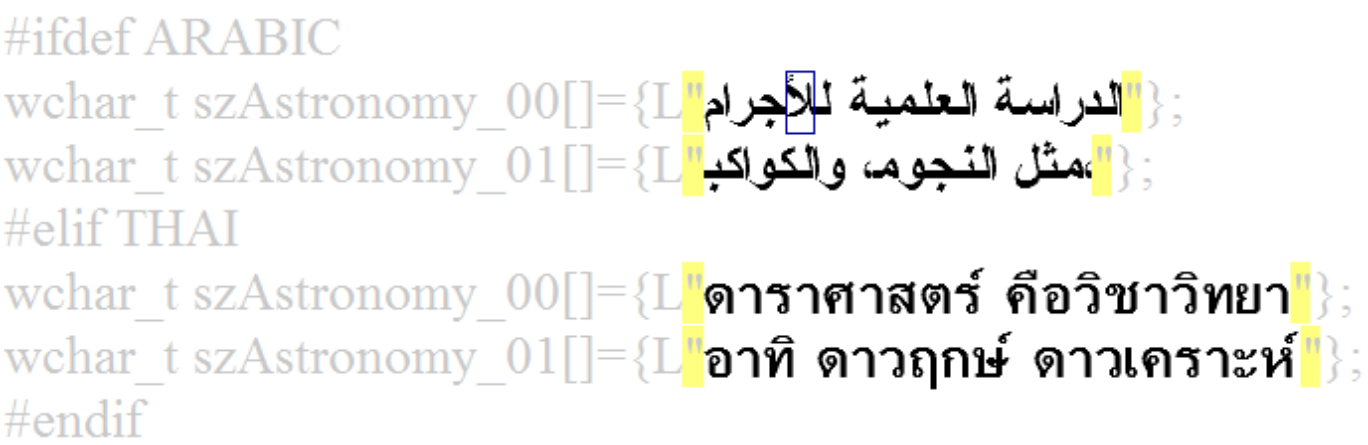


Press **OK**.

Press **Yes** to **Insert New Additional Characters**.

The program finds all the different Basic Characters, Combined Characters and Presentation Forms including non-displayable control characters inside the ‘C’ strings, and marks the already present characters and symbols with green and the newly added characters from the Master Font with grey:

The texts in the strings are displayed automatically with the new font and a blue mark to identify the characters individually:



(The text window can be closed with the **Show Imported Text** button)

Your Font is ready for saving.

3: Font that Fits a Code Point Character List:

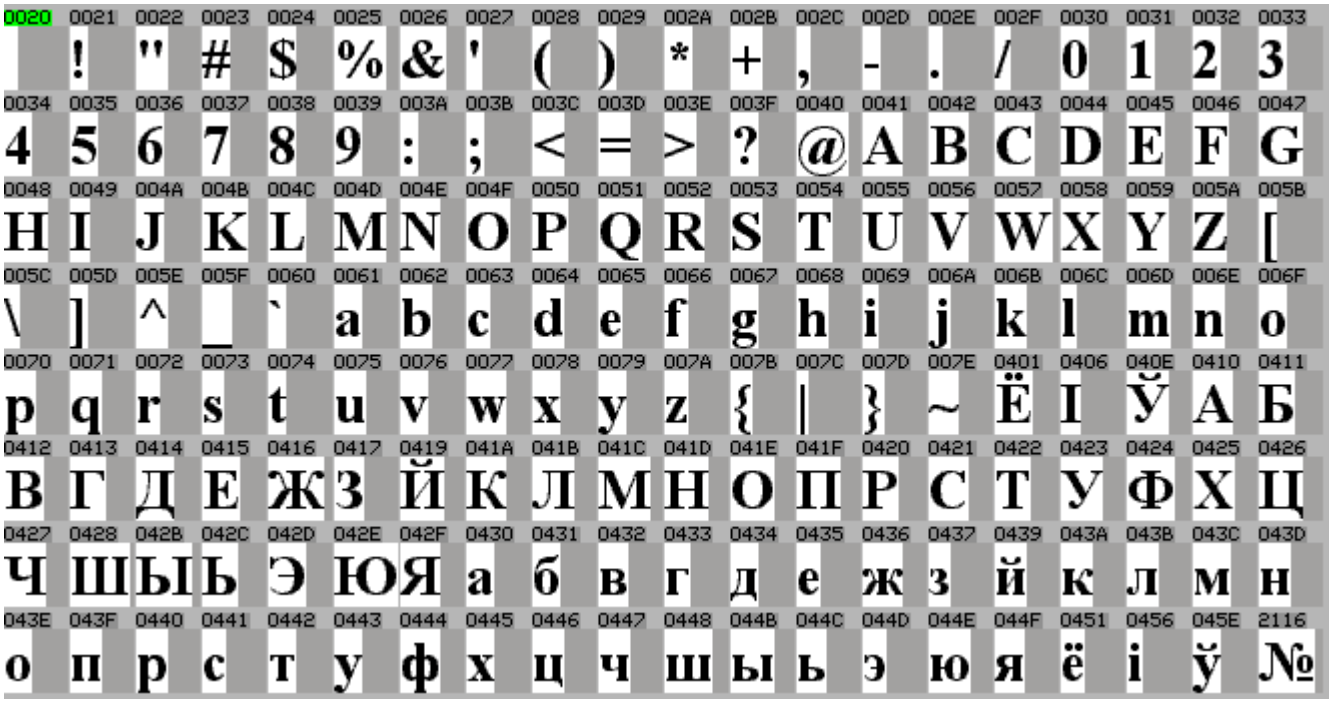
This option interprets Files with the extension **.cp**.

Open the Code Point *BelarussianASCII.cp* with **File –>Import Code Point Character List to Mark or Create Characters**:

Press **Yes** to **Insert New Additional Characters**.

Change to ***Font Edit***

The program marks the already present characters with green and the newly added characters from the Master Font with grey:



Your Font is ready for saving.

E: Paste Text as Font:

From a text editor such as Notepad or Word or from a Browser mark a text and copy it to the Clip Board with <Ctrl>C:

```
Tørt og skyet vejr, men i løbet af eftermiddagen opklaring.
```

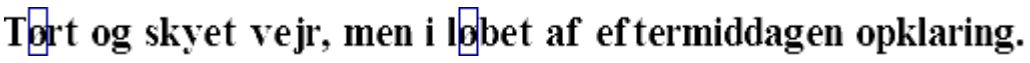
In IconEdit press OK to leave the create box.

Change to ***Font Edit***.

Paste the text with <Ctrl>V, and select Put All as Characters:

This generates a small Latin font including the necessary punctuation marks:

The text displayed automatically with the new font and blue marks to identify the characters individually:



Your Font is ready for saving.

F: Import Text or Text Catalogue File to Create a Text Optimized Font:

This file menu command does not operate on an already existing font, but create a new font based on the Master Font.

The function can be accessed in two ways, either as a menu command with **Text –>Import Text, Text Catalogue to Create a Text Optimized Font** or directly by opening a text file:

1: Plain Text with Diacritics generating Auto-generated Characters

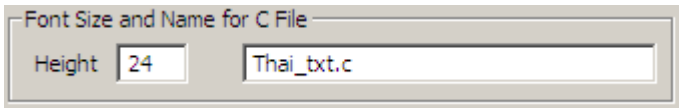
This option interprets Files with the extension **.txt**.

Press the ***File Open*** button in the Main tool bar:

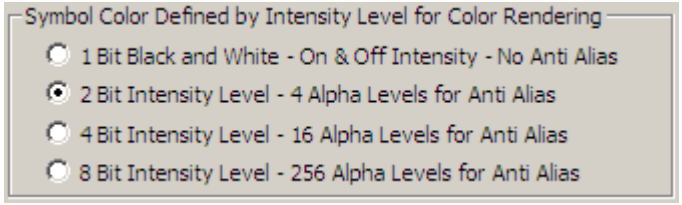
Open the Unicode text *Thai.txt*:

ปรินส์ออฟเปอร์เซีย เป็นเกมแนวอาร์เคด
1 เนื้อเรื่อง
2 ตัวละคร
2.1 เจ้าชาย
2.2 เจ้าหญิง
2.3 จาฟฟาร์
2.4 เจ้าชายร่างกระจก
และหากเห็นผู้ใดเข้ามารักนางถือเป็นตัว

This opens the ***Create New Text Font*** dialog box:

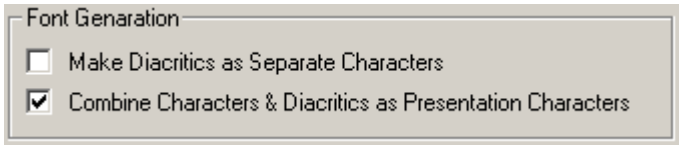


In the color version of IconEdit there is an additional choice of Color Mode:

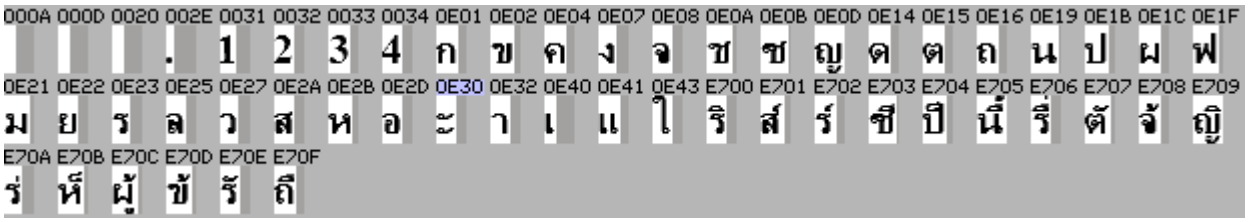


Press **OK**.

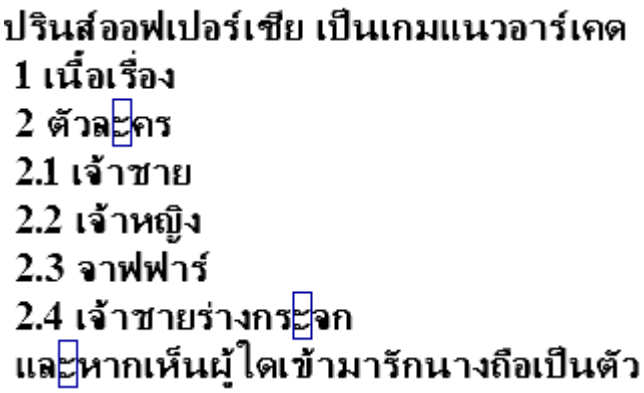
This opens the **Choose Autogenerated Characters** dialogbox:



Press ***OK***, and IconEdit analyses the input for combinations of basic characters and diacritics and generates the necessary Auto-generated Characters:



The text displayed automatically with the new font and blue marks to identify the characters individually:



Your Font is ready for saving.

2: “C” Text Catalogue with Auto-generated Characters instead of Diacritics

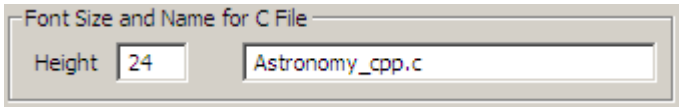
This option interprets Files with the extensions **.c .h .cpp** and **.cs**.

Press the ***File Open*** button in the Main tool bar:

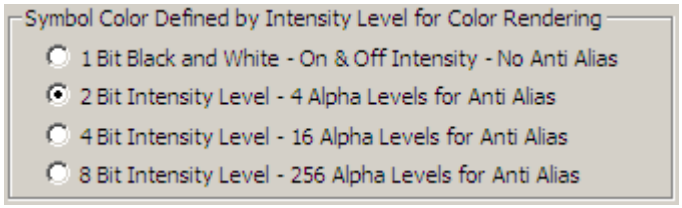
Open the Unicode text *Astronomy.cpp*:

```
// Disclaimer: These texts are only for demonstration of the principle
// and may not make any sense to someone familiar with the language
#ifdef ARABIC
wchar_t szAstronomy_00[]={L"الدراسة العلمية للأجرام"};
wchar_t szAstronomy_01[]={L"مثل النجوم، والكواكب،"};
#elif THAI
wchar_t szAstronomy_00[]={L"ดาราศาสตร์ คือวิชาวิทยา"};
wchar_t szAstronomy_01[]={L"อาทิ ดาวฤกษ์ ดาวเคราะห์"};
#endif
```

This opens the ***Create New Font*** dialog box:

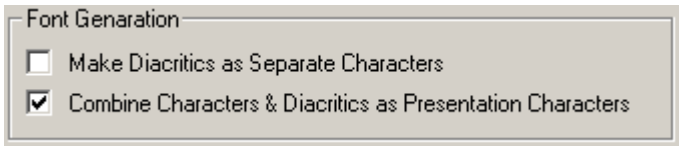


In the color version of IconEdit there is an additional choice of Color Mode:



Press **OK**.

This opens the **Choose Autogenerated Characters** dialogbox:



Press ***OK***, and IconEdit analyses the input for combinations of basic characters and diacritics inside the "C" strings and ignores the code. This generates a small Thai and Devanagari font including the necessary Auto-generated Characters necessary for rendering the texts:

The texts in the strings are displayed automatically with the new font and a blue mark to identify the characters individually:

```
// Disclaimer: These texts are only for demonstration of the principle
// and may not make any sense to someone familiar with the language
#ifdef ARABIC
wchar_t szAstronomy_00[]={L"لدراسة العلمية للأجرام"};
wchar_t szAstronomy_01[]={L"مثل النجوم والكواكب"};
#elif THAI
wchar_t szAstronomy_00[]={L"ดาราศาสตร์ คือวิชาวิทยา"};
wchar_t szAstronomy_01[]={L"อาทิ ดาวฤกษ์ ดาวเคราะห์"};
#endif
```

Your Font is ready for saving.

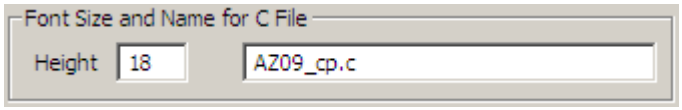
3: Make a New Font Based on an Existing a Code Point Character List:

This option interprets Files with the extension **.cp**.

Press the ***File Open*** button in the Main tool bar:

Open the Code Point Character List *AZ09.cp*:

This opens the ***Create New Font*** dialog box:



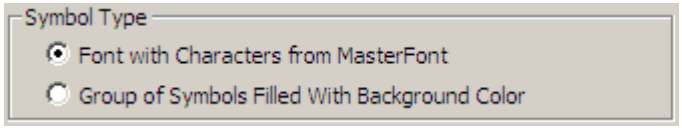
Press ***OK.*** This creates a new font:

Your Font is ready for saving.

04: How to Build Fonts from Scratch

All examples in this manual assume that IconEdit is reset to factory defaults except for continuations.

We recommend filling the character symbols with auto-generated characters from a Master Font as a guideline:



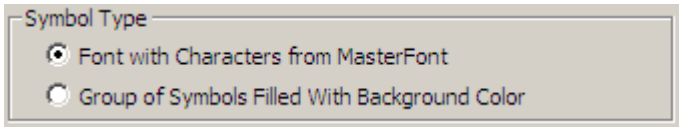
Such auto-generated fonts are usually sufficient for the development and debug phase of a project, but before the first release of the project all characters should be checked for readability and harmony. The characters can usually be edited on the font or script level by changing the Thickness and Relative Width settings for the Master Font so the characters can be used as they are without much individual editing.

A: How to Draw a Black & White ASCII Font with 95 Characters

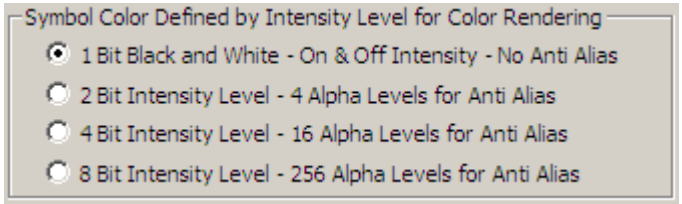
To make a new font press the ***New Font or Symbol Group*** button in the Main Toolbar:

This opens the create dialog box:

Choose how characters or symbols should be organized and shown initially:

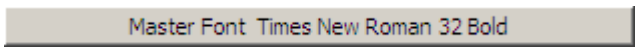


Choose a color format, here Black & White. This option is only available in the color version of IconEdit:

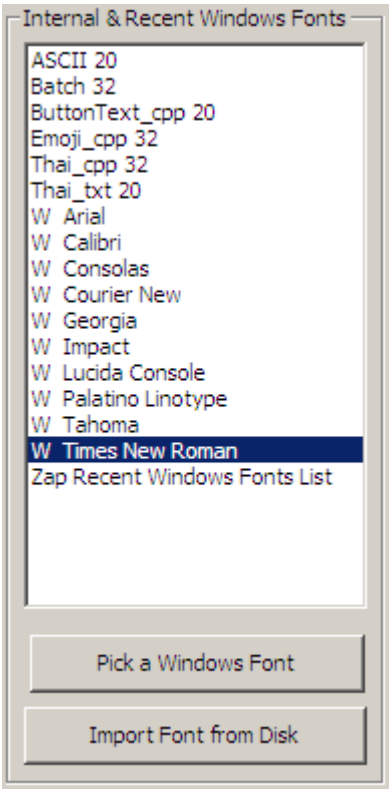


Choose a master font, in this example a TrueType or OpenType vector font already installed in Windows:

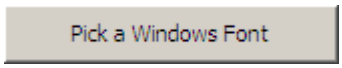
The exact text on the Master Font buttons may vary depending on the operating system version and prior settings in IconEdit.



First, select a Windows font either from the Recent list or pick a new Windows font:



If the recent fonts are not ideal, pick another Windows font as this example shows:



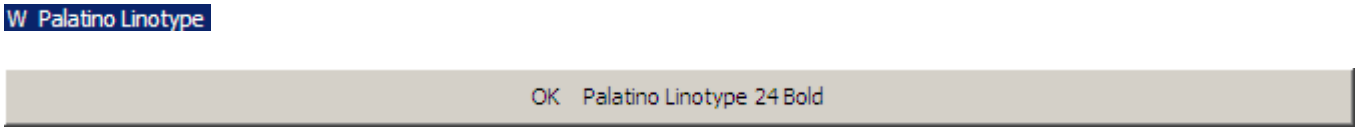
This opens the Windows font picker, the appearance of which is depends greatly on the Windows version, Windows native language, what kind of foreign language support is installed, and any additional non-Windows fonts added later.

Fill in the 3 top fields to choose for example Palatino Bold and 24. The last field is the height of the whole Character including top and bottom white space. This number will be the height of the each symbol in pixels.

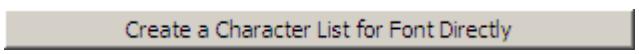
Palatino Linotype Bold 24

Press OK.

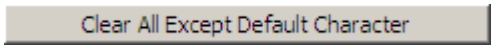
Confirm chosen Windows font:



The displayable ASCII characters range from 0x20 to 0x7E. Open the Character List dialog box:



One or more characters in addition to the default character may already selected, so:

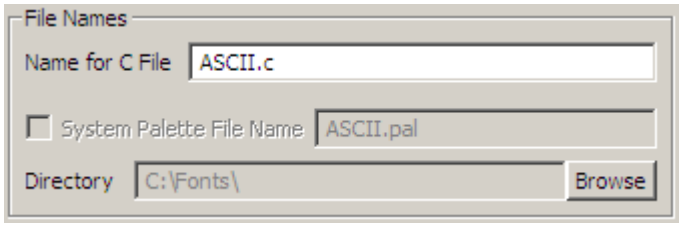


This does not delete the default character that is used when a character is not present in the font.

Mark from SPACE to TILDE with the mouse. The characters are highlighted in orange to indicate insert:

Press Insert:

Give the font a name, only valid ‘C’ names are valid here and the extension should not be changed:



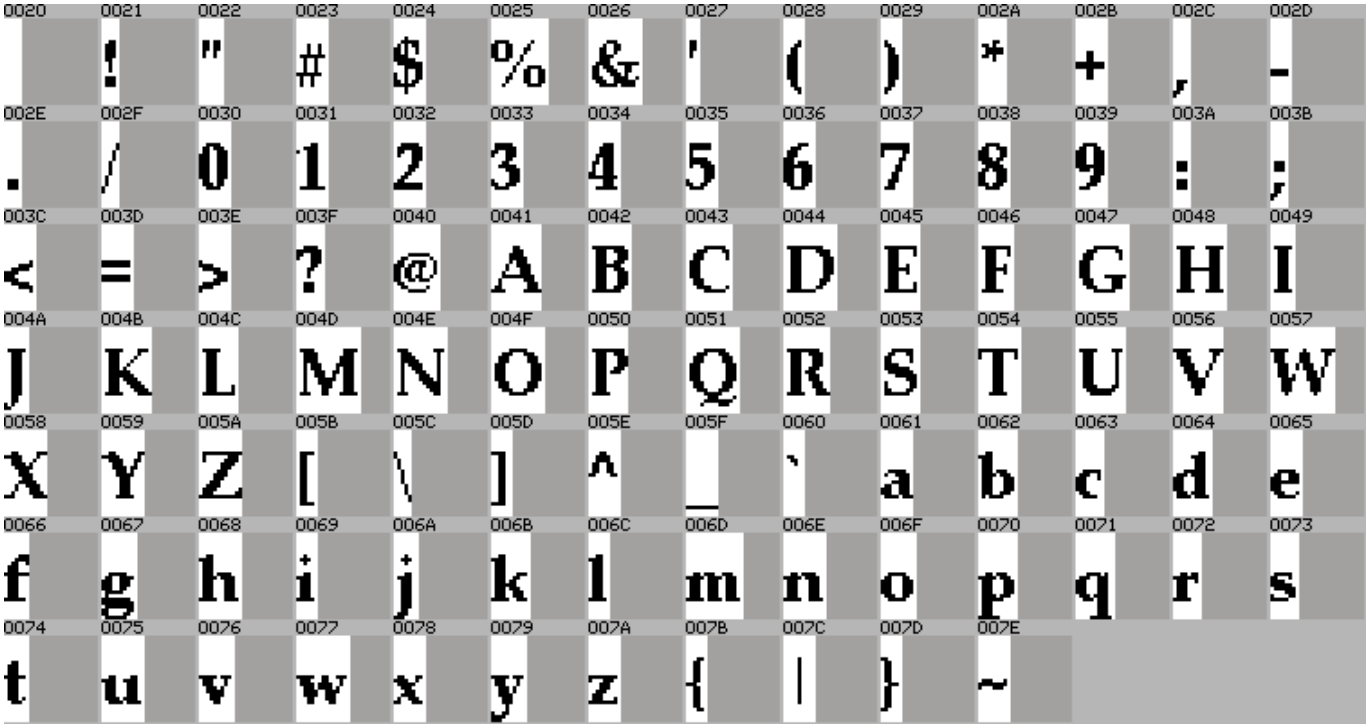
The name of the Code Point Character List file is automatically added.

Press OK.

To better view the font, use ***Change Zoom***:

Make the picture smaller or bigger with the ***+ - /2 *2*** button.

The font now looks like this:



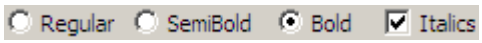
The small number above each character is the Unicode Code Point. This value will follow the image of the character during all editing, and eventually end up in the Code Point Character List file that is associated with the symbol file when the font is saved.

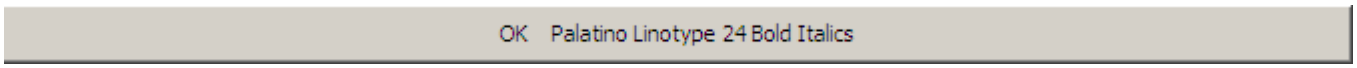
Modification at character level can be done in this edit mode.

To change the numbers in the font to italics first change the master font press ***MasterFont***:

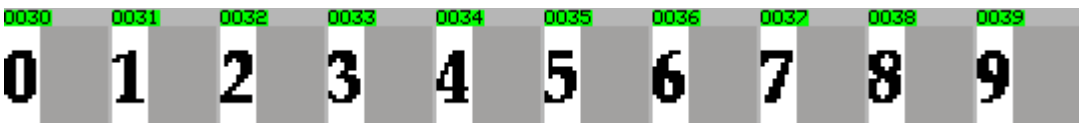
To enter the master font dialog box:

Press Italics:





Mark the numbers with the mouse in normal text editor style:



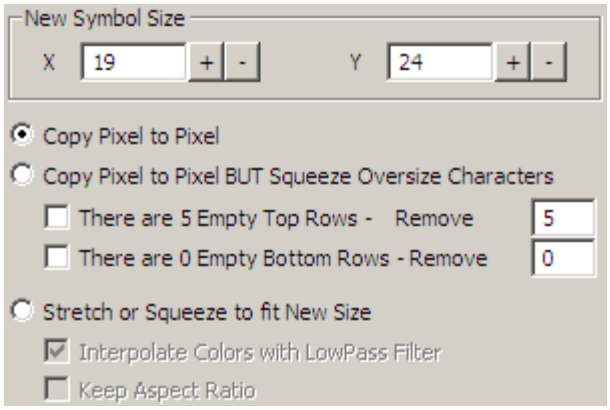
The ***Redraw Characters*** now has a green frame to indicate that only the marked characters will be redrawn, press it to redraw the marked characters:

New numbers are drawn and the markings are now grey to indicate new characters:

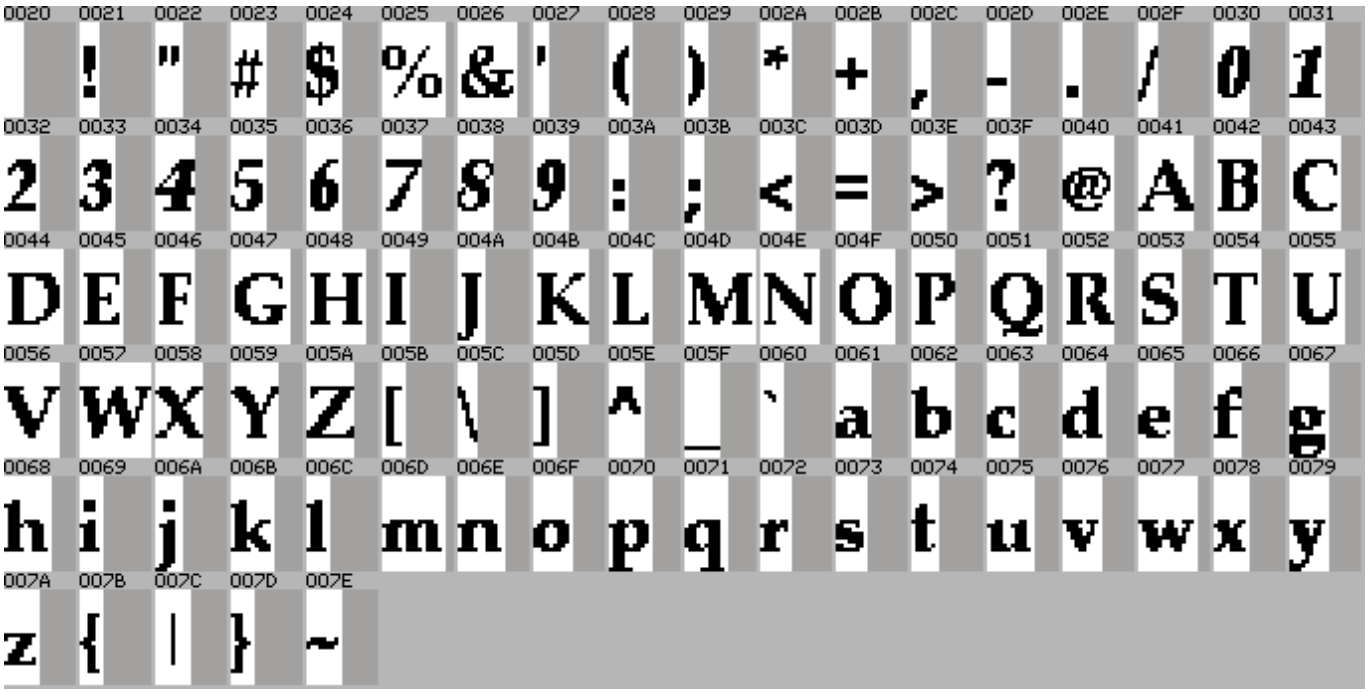
The width of the font is auto-generated to have room to draw any character in the master font.

To remove surplus unused pixels press ***Resize All Symbols***:

The smallest symbol size that can contain the largest character is suggested:



Press OK to reduce the font to its minimum size:



Your Font is ready for saving....

Press the ***Save All As…*** button in the Main tool bar to save the data as *ASCII.c* in the proper directory. The Sym and Code Point Character List files with the corresponding glyphs and Code Points are saved automatically.

1: How to Add a Foreign Language to an Existing Font

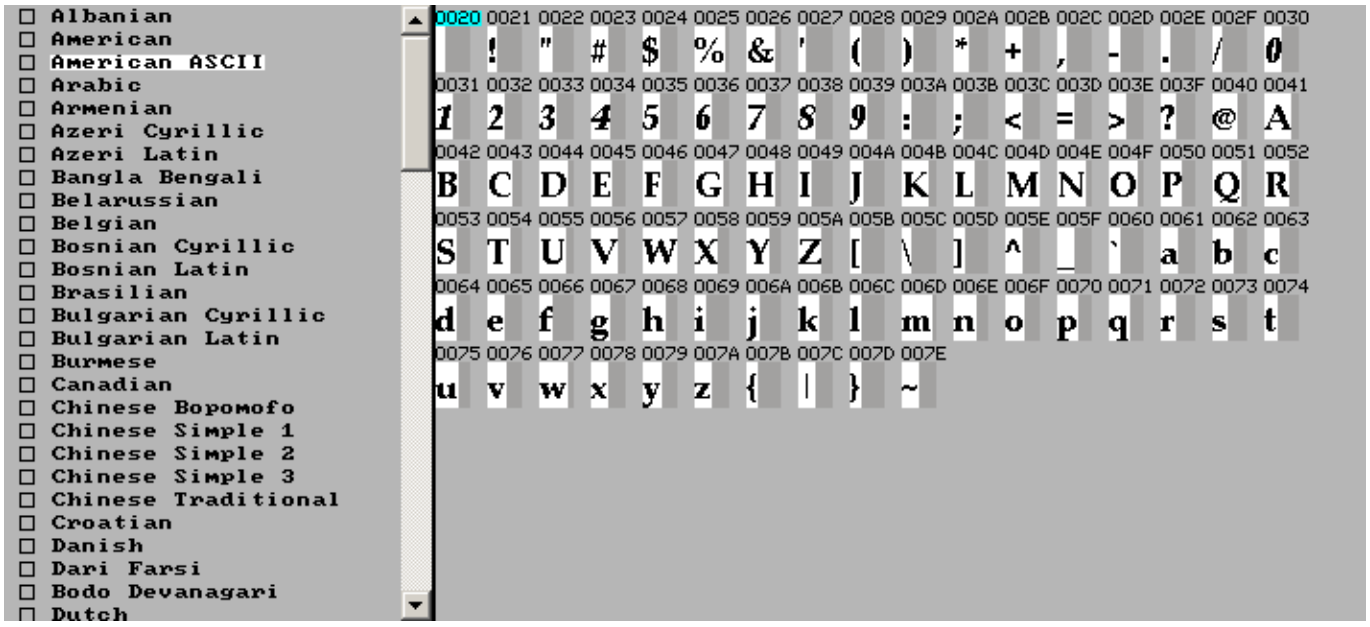
This example is a continuation and uses the font and settings of the previous example.

Warning: *Language & Region filters are probabilistic in nature; they contain the characters normally associated with a Language or a Region, but they may not cover technical terms, foreign words or names. This will do fine during development, but before release, when all the texts in the project are defined, the font should be checked with the function:* **Text -> Import Text or Text Catalogue to Mark or Create Characters…**

Warning: *Language & Region only works with Master Fonts according to the Unicode standard as described at www.unicode.org. See the chapter* **How to Check that an Existing Font is According to Unicode**.

Press ***Language & Region*** Filters:

To enter language edit mode:



Upon entry to Language & Region edit mode, the present font is already marked to avoid accidental erasure of existing symbols in this edit mode:

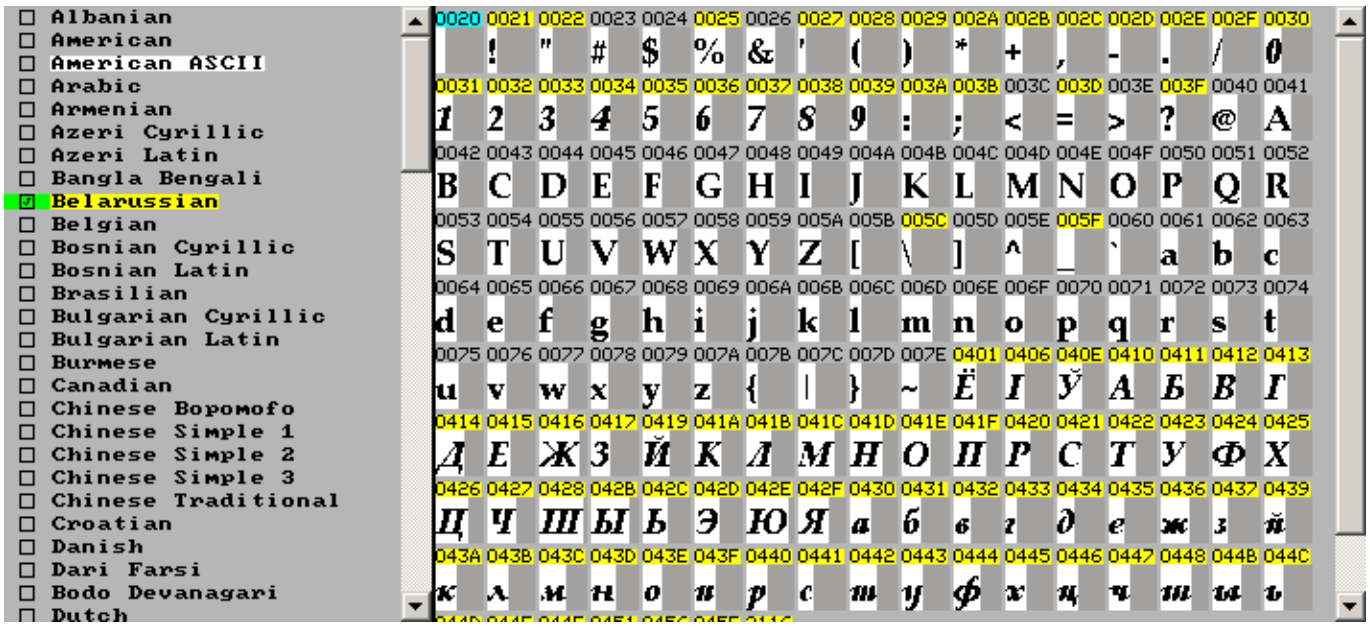
Warning: If this is un-ticked and no other filter is chosen by a tick, all characters except the default character are marked for delete.

The default character is marked by highlighting the number in cyan, and the languages already included in the font are highlighted in white.

Press the tick box for a language, for example Belarusian:

☐ Belarussian

This highlights the chosen language and adds the missing characters:



The language tick box is highlighted in green and the language and the new characters are marked in yellow; the tick box can be used for removing the language again.

Switch back to ***Font Edit*** mode to check:

To remove surplus unused pixels press ***Resize All Symbols***:

The smallest symbol size that can contain the largest character is suggested. Select ***Copy Pixel to Pixel***.

Press OK to reduce the font to its minimum size.

Your Font is ready for saving....

Press the ***Save All As…*** button in the Main tool bar to save the data as *BelarussianASCII.c* in the proper directory. The Sym and Code Point Character List files are saved automatically.

2: How to Check that an Existing Font is According to Unicode

This example is a continuation and uses the font and settings of the previous example.

Press ***MasterFont*** button:

Check that the Master Font is a Unicode Windows font:

It is not important which Windows Master Font it is as long as it is Unicode and the font actually has defined the characters you want to check.

Press OK.

Press ***Compare Symbols: Show Symbols in Actual Size***:

Press again as ***Compare Symbols: Show Symbols and Master Font Together***:

This opens a window for comparing the existing font on the top row with the Master Font on the bottom row:

Scroll through the whole font with the mouse and check that the top and bottom glyph represent the same character.

3: How to Squeeze and Mono-Space an existing Font Manually

This example is a continuation and uses the font and settings of the previous example.

In some applications it is preferable that all characters have the same width, it simplifies the calculation of string lengths and reduces flicker when strings are changed. In addition mono-space fonts usually require less memory space because the widest characters are squeezed somewhat.

For Black & White fonts we recommend that squeezing is done for one character at a time with the **Squeeze or Stretch Character with Mouse** in **Character Edit** mode to keep track on the deformation of the characters whereas squeezing of Grey fonts can normally be done safely for the whole font as one operation in **Font Edit** mode with the use of **Squeeze Oversize Characters** in the **Resize All Symbols** function.

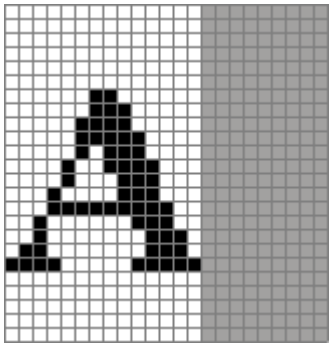
These two methods are described in chapter 04.A.3 and 05.C.1 respectively.

In this example we will squeeze the B&W font *BelarussianASCII.c* from 23x24 to 16x24.

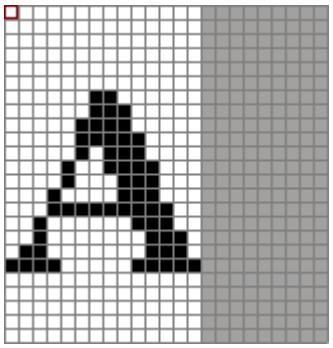
First select all characters with the ***Mark All Symbols*** button:

Then leftset all characters with the ***Leftset Character*** button:

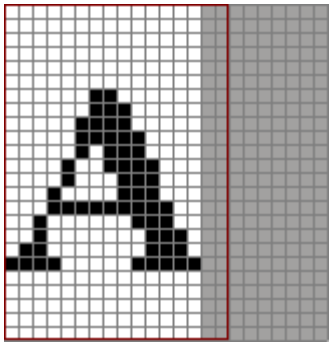
Change to **Character Edit** by right click on the letter A to make a reference frame 16x24.



Use ***Place Drawing Reference Frame*** to make a 16x24 frame for future reference, ***Show Drawing Reference Frame*** is automatically activated:



Catch the lower left corner of the frame with the mouse and move it to **Frame 16x24:**



The reference frame is now visible and the character is within the frame:

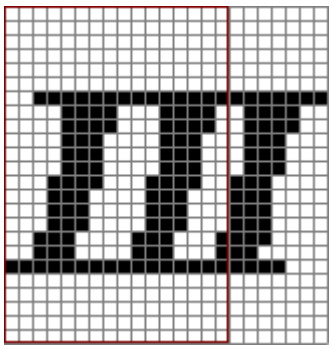
Disable the ***Place Reference Frame*** by pressing it again, or by activating another tool such as ***Draw Pixels***:

To see the reference frame for many characters press the ***Show Symbols in Actual Size*** and ***Show Symbols and Master Font for Comparison*** buttons if the functions are not already active. The reference frame is now shown on all symbols:

This provides a quick check to see if any of the symbols are outside the frame.

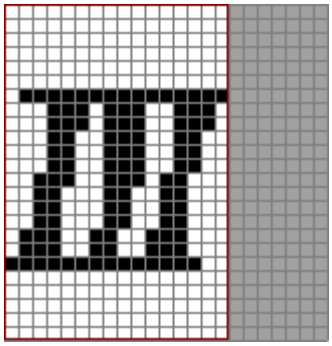
To find the largest of the characters that is larger than the frame click ***Get Biggest Character***:

This shows the four extremes of the glyphs and the widest character, click the button for ***Right***:



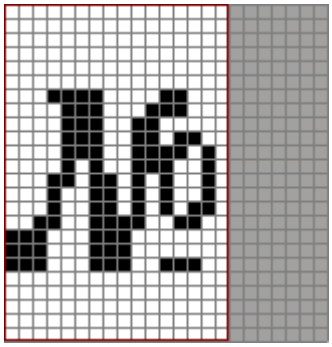
Activate the ***Squeeze or Stretch Character with Mouse*** tool:

Click on the lower right corner and move the cursor left until the symbol is inside the frame:

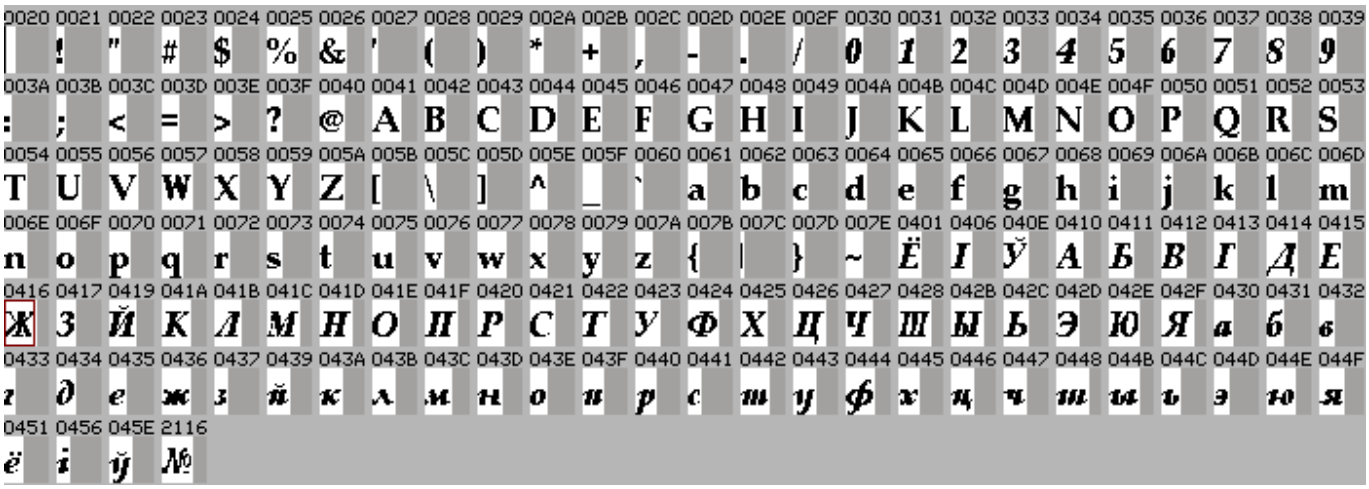


Repeat this procedure until all characters are inside the frame.

In case of *italics* characters it is acceptable that the glyph takes up the whole width of the symbol, but if there are vertical lines at the right side of the glyph there should be at least one empty column:



When all glyphs are inside the frame change to **Font Edit** by a right click on the symbol:



To remove surplus unused pixels press ***Resize All Symbols***:

The smallest symbol size that can contain the largest character is suggested. Select **X = 16** and ***Copy Pixel to Pixel***. Press OK to reduce the font to the reference frame size:

The Squeezed Font is ready for saving....

Press the ***Save All As…*** button in the Main tool bar to save the data as *BelarussianASCII_S.c* in the proper directory. The Sym and Code Point Character List files are saved automatically.

To make a mono-space font select all characters with the ***Mark All Symbols*** button:

Then mono-space all characters with the ***Monospace and Center Character*** button:

The mono-spaced font is ready for saving....

Press the ***Save All As…*** button in the Main tool bar to save the data as *BelarussianASCII_M.c* in the proper directory. The Sym and Code Point Character List files are saved automatically.

B: How to Draw a Grey Alpha Level Font with Greek Anti-Aliased Characters

This example applies only to the color version of IconEdit.

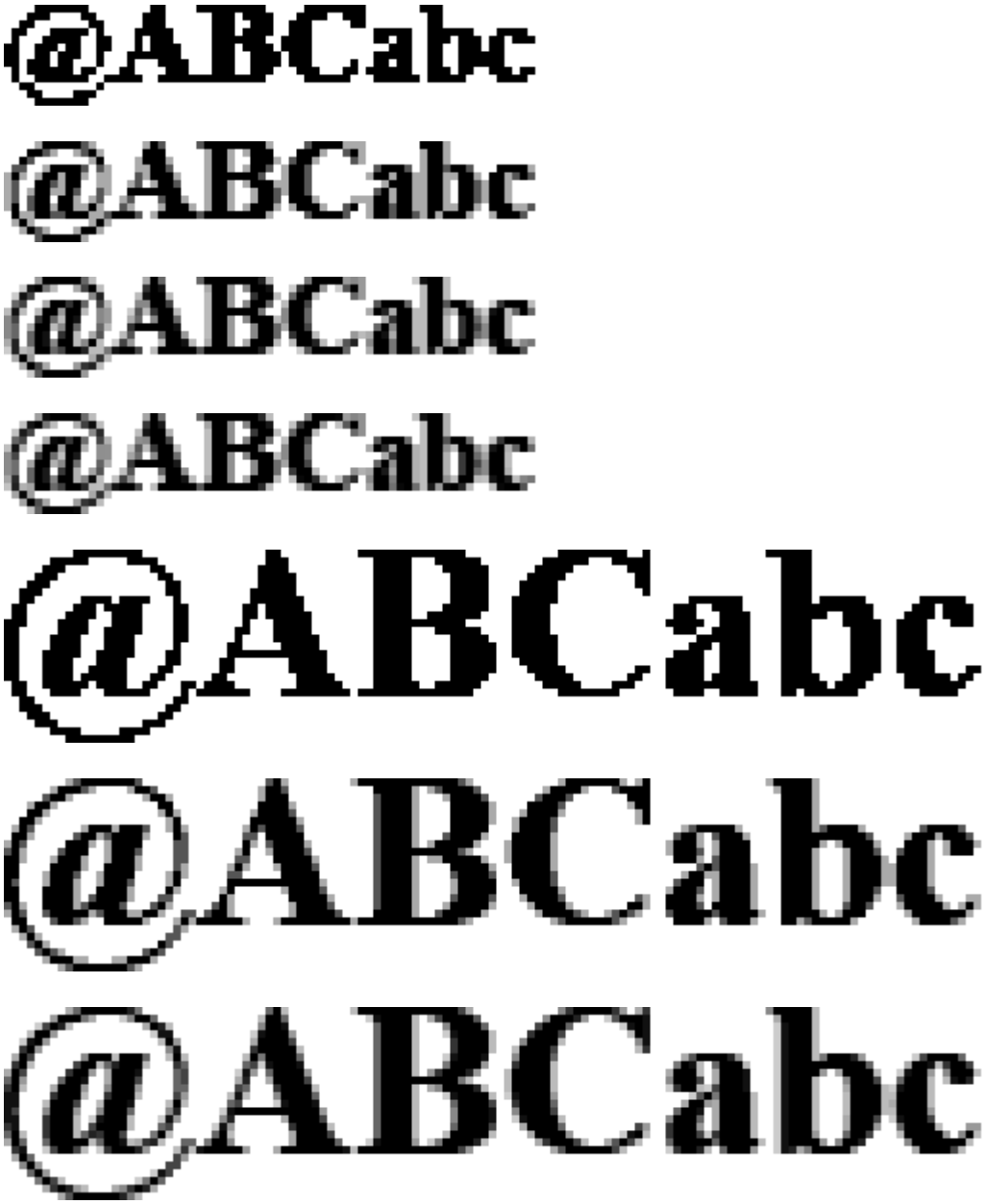
1: How Many Bits per Pixel are Really Necessary

The first thing to decide is how many bits to use per pixel.

Below are a Times New Roman font in sizes 18 and 32 drawn with 1 bit per pixel black and white and 2, 4 and 8 bits per pixel grey alpha levels, all of which are made by the Windows rendering engine.

These engines are slightly different on different versions of Windows, XP makes 16 shades, Windows Vista, 7 and 8 makes 64 shades, and Windows 10 makes 80 shades, but even 4 shades are usually enough to make the text look smooth.

The difference for the grey resolutions seems important if the characters are blown up:



But when shown in actual size, the different sizes of grey anti aliased texts are difficult to tell apart:

@ABCabc
@ABCabc
@ABCabc
@ABCabc

@ABCabc
@ABCabc
@ABCabc
@ABCabc

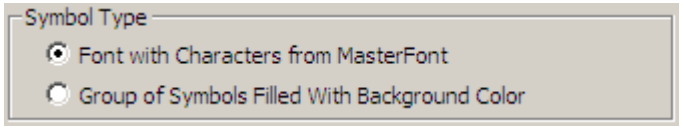
To limit the amount of memory needed to store the font, 2 bits per pixel will be used, because it is usually sufficient.

2: How to Make a Grey Alpha Level Font

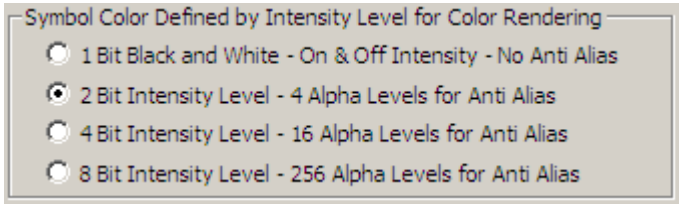
To make a new font press the ***New Font or Symbol Group*** button in the Main Toolbar:

This opens the create dialog box:

Choose how characters or symbols should be organized and shown initially:

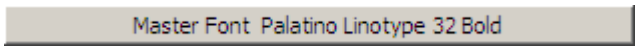


Choose a color format:

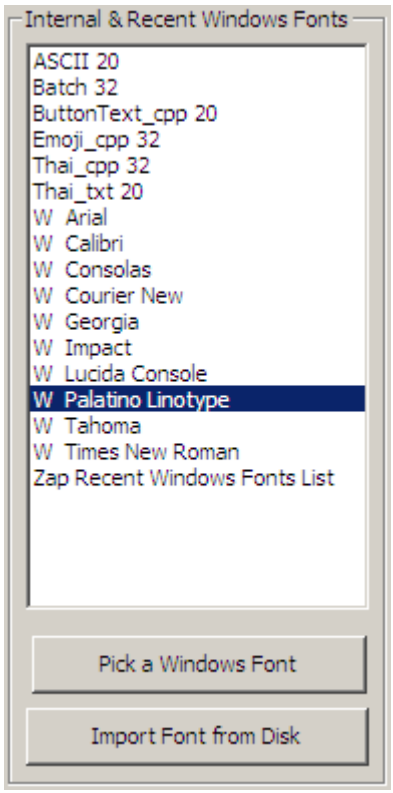


Choose a master font, in this example a font already installed in Windows:

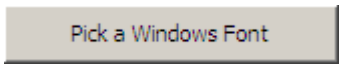
The exact text on the Master Font buttons may vary depending on the operating system version and prior settings in IconEdit.



First, select a Windows font either from the Recent list or pick a new Windows font:



If the recent fonts are not ideal, pick another Windows font as this example shows:



This opens the Windows font picker, the appearance of which is depends greatly on the Windows version, Windows native language, what kind of foreign language support is installed, and any additional non-Windows fonts added later.

Fill in the 3 top fields to choose for example Arial Regular and 24. The last field is height of the whole Character including top and bottom white space. This number is the height of each symbol in pixels.

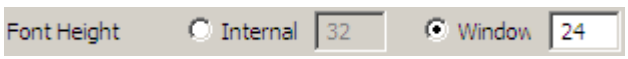
Arial Regular 24

Press OK.

Confirm chosen Windows font:

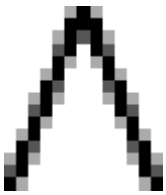
This is a grey tone vector font, and the conversion to grey with limited resolution can be fine trimmed.

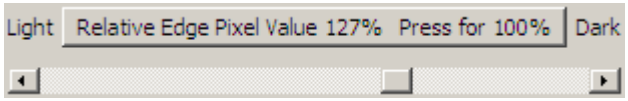
Choose the conversion:



Adjust the thickness of the edge of the character until the connectivity is OK.

Characters with diagonal lines like / @ A W are usually the most critical.



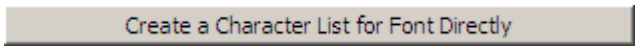


Or turn off Zoom to see several characters.

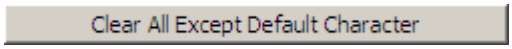
Press OK.

The displayable Greek characters range from 0x386 to 0x3CE.

Open the Character List dialog box:

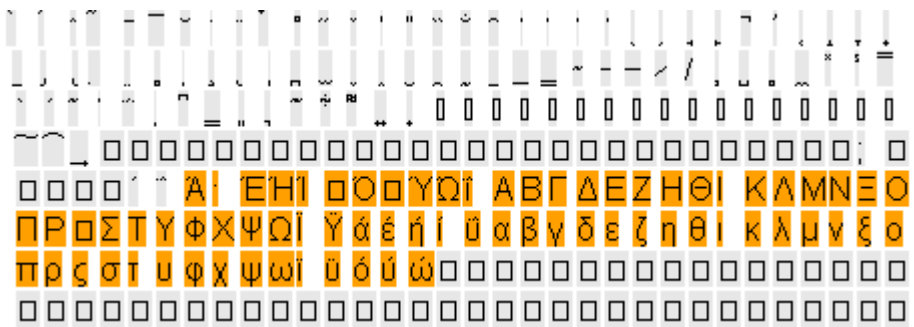


One or more characters in addition to the default character may already selected, so:



This does not delete the default character that is used when a character is not present in the font.

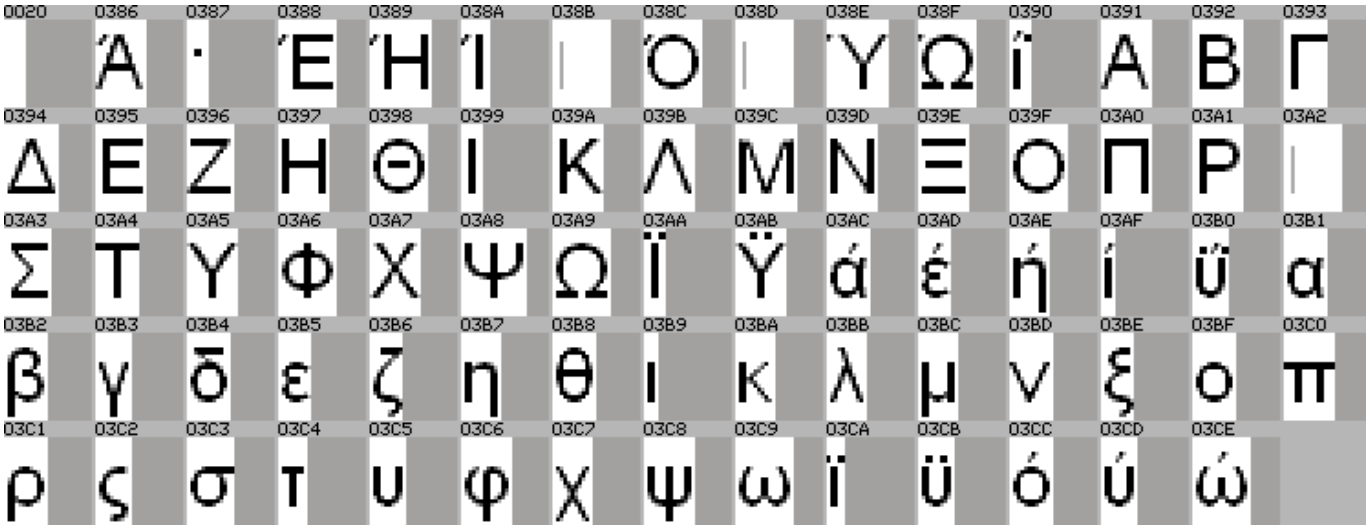
Select SPACE 0x20 and scroll to page 03xx, select 0x0386 to 0x03CE with Ctrl + mouse:



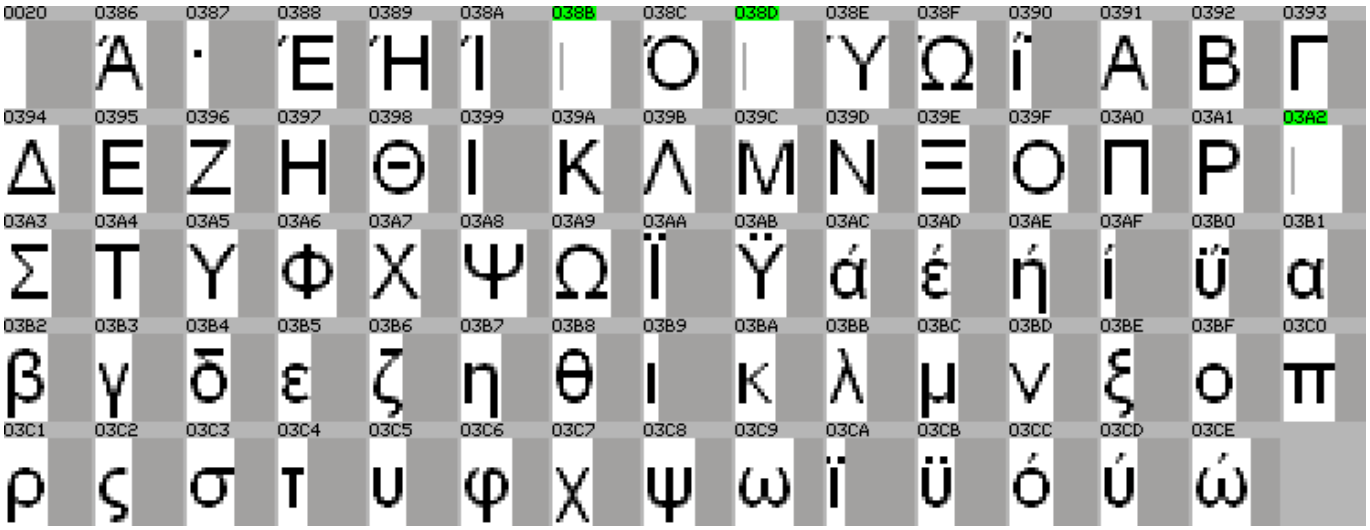
Press Insert.

Press OK.

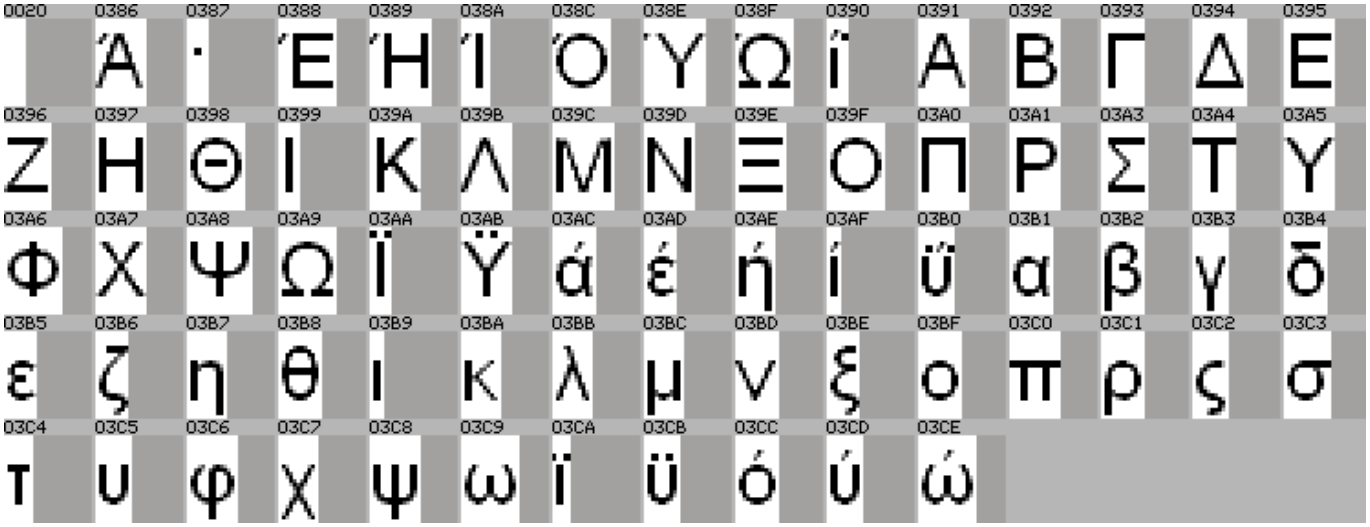
The font now looks like this:



The characters 0x38B, 0x38D and 0x3A2 are undefined. Mark with control + mouse click:



Delete the marked characters:



To remove surplus unused pixels press ***Resize All Symbols***:

The smallest symbol size that can contain the largest character is suggested. Select ***Copy Pixel to Pixel***. Press OK to reduce the font to its minimum size.

Your Font is ready for saving....

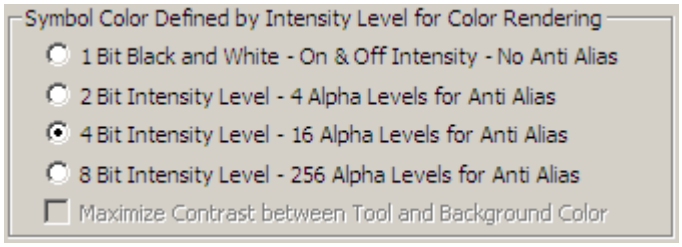
Press the ***Save All As…*** button in the Main tool bar to save the data in the proper directory. The Sym and Code Point Character List files are saved automatically with the .c file.

3: How to Make a Semi Transparent Alpha Level Font

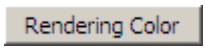
This is a continuation of the previous example.

In order to get a reasonable grey-tone resolution the number of alpha level should be higher, press ***Modify Grey Anti Alias Font***:

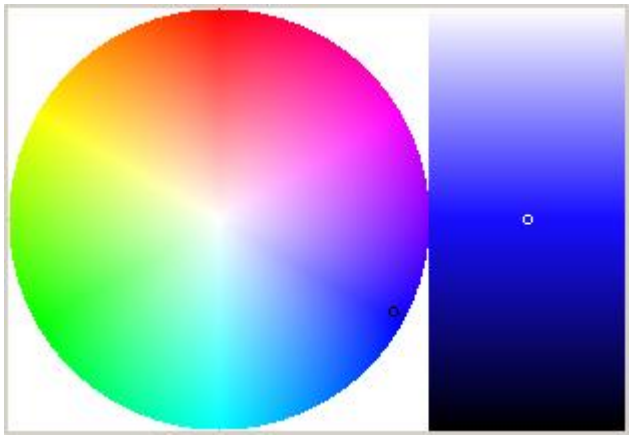
Change to 4 bit resolution:



Set a rendering color for local use to show the transparency, this color has no influence on how the font can be displayed on the target display:



Click on the color circle to get a color:



Press OK.

Use the rendering color:

Press OK.

The font look is unchanged, but it is now shown in color on a chequered background:

Activate the tool color palette:

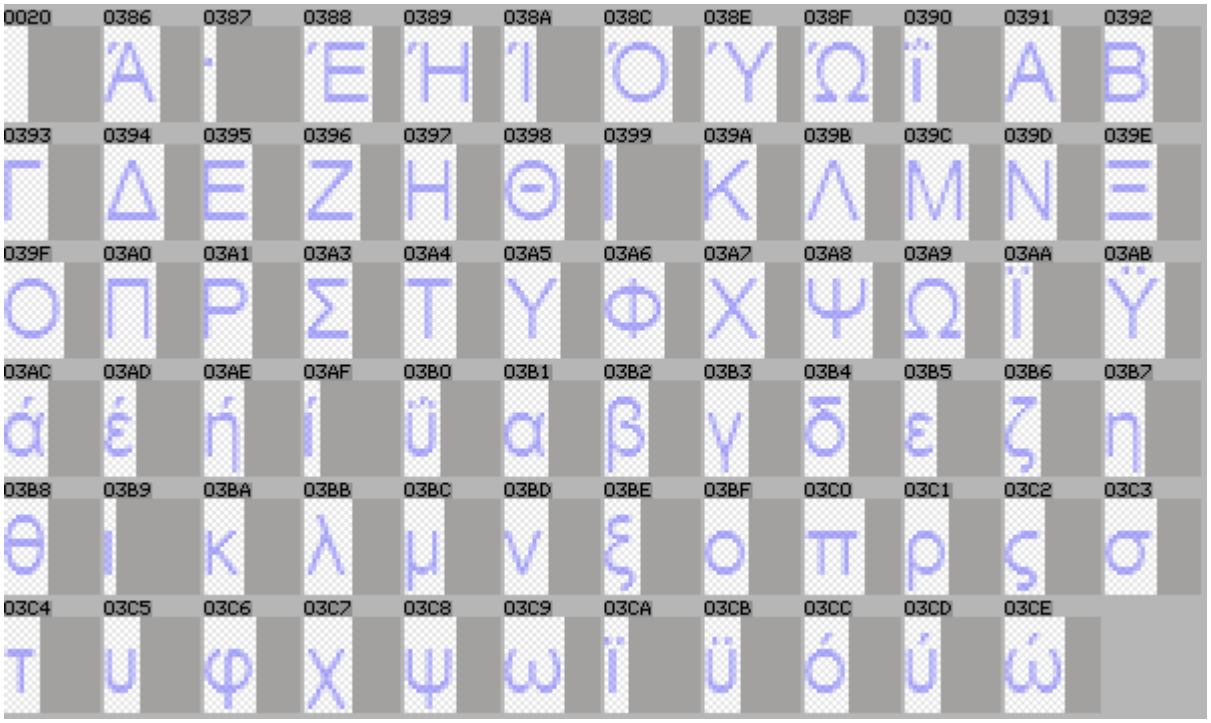
The top half of the buttons show the rendering color, the bottom half shows the degree of transparency the tool color represents.

Choose a transparency level:

Mark All Symbols for redrawing:

Redraw Characters with the new tool color:

The font is now semitransparent:



Your Font is ready for saving....

Press the ***Save All As…*** button in the Main tool bar to save the data in the proper directory. The Sym and Code Point Character List files are saved automatically with the .c file.

C: How to Draw a Color Font with Emojis

This example applies only to the color version of IconEdit.

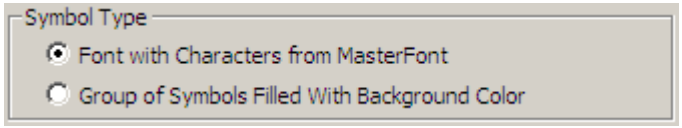
Normally fonts only require a foreground and a background color and should be created with one of the **Intensity Level** formats for color rendering at runtime, but if more colors are required for the characters the font should be created with the rendering colors already in the font.

Warning: *Emojis and other high plane characters are only supported on Windows version 8 and newer.*

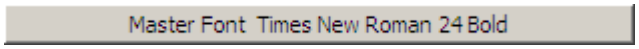
To make a new color font press the ***New Font or Symbol Group*** button in the Main Toolbar:

This opens the create dialog box:

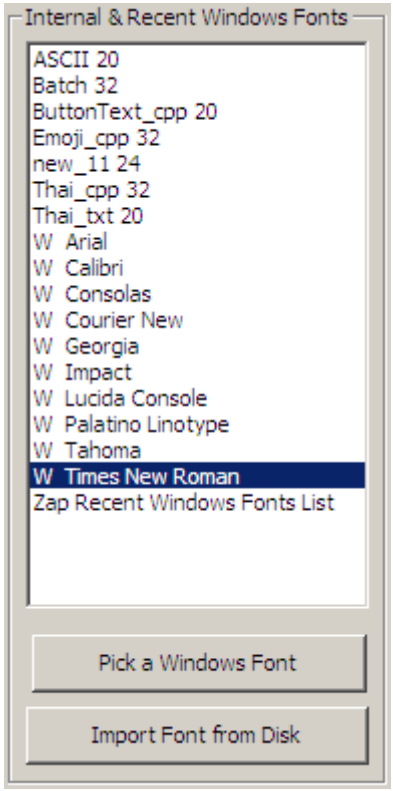
Choose how characters or symbols should be organized and shown initially:



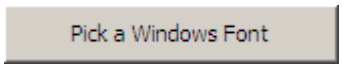
Choose a master font, in this example a font already installed in Windows:



First, select a Windows font either from the Recent list or pick a new Windows font:



If the recent fonts are not ideal, pick another Windows font as this example shows:



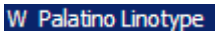
This opens the Windows font picker, the appearance of which is depends greatly on the Windows version, Windows native language, what kind of foreign language support is installed, and any additional non-Windows fonts added later.

Fill in the 3 top fields to choose for example Palatino Bold and 32. The last field is the height of the whole Character including top and bottom white space. This is the height of each symbol in pixels.

Palatino Linotype Bold 32

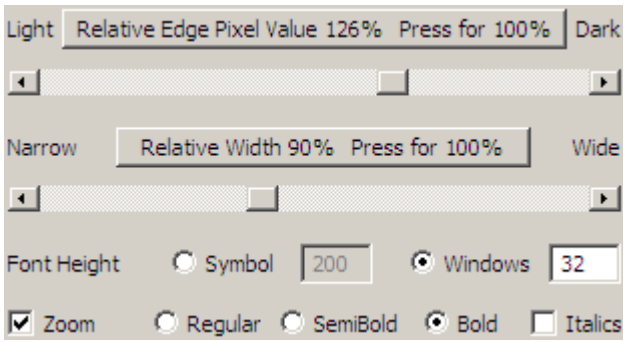
Press OK.

Confirm chosen Windows font:



This is a grey-tone vector font, and the conversion to grey levels can be fine trimmed.

Choose the conversion:

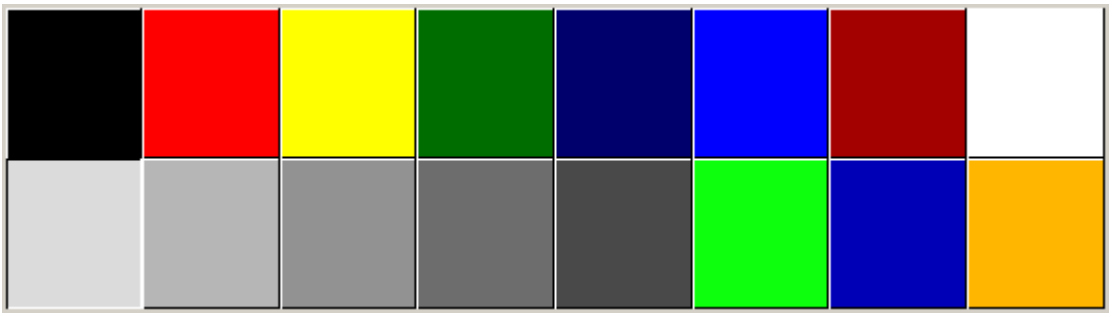


Press OK.

The New dialog box has several color modes, for anti alias with colors use **32 Bit ARGB** and without anti alias use **4 Bit Color Palette** to save memory space:

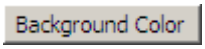
Choose a color mode:

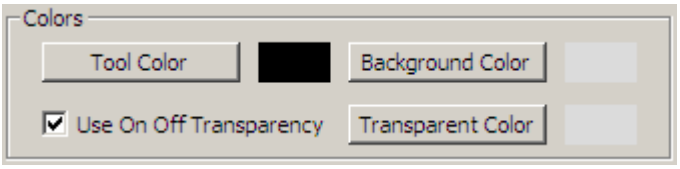
Enable **On Off Transparency** and select the single transparent color;



Press light grey:

Do the same for background:



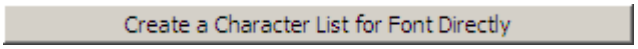


The Emoji are in the ranges 0x1F300 to 0x1F6FF and 0x1F900 to 0x1F9FF.

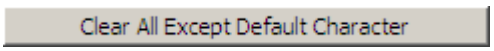
From here we will explore two options, a few emojis without anti aliasing in the private area and the whole range of faces with anti alias in the high plane.

1: Two emojis without anti alias in private area

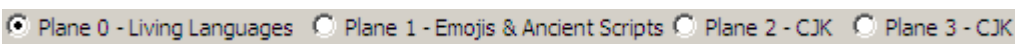
Open the Character List dialog box:



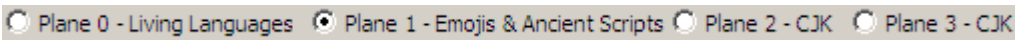
One or more characters in addition to the default character may already selected, so:



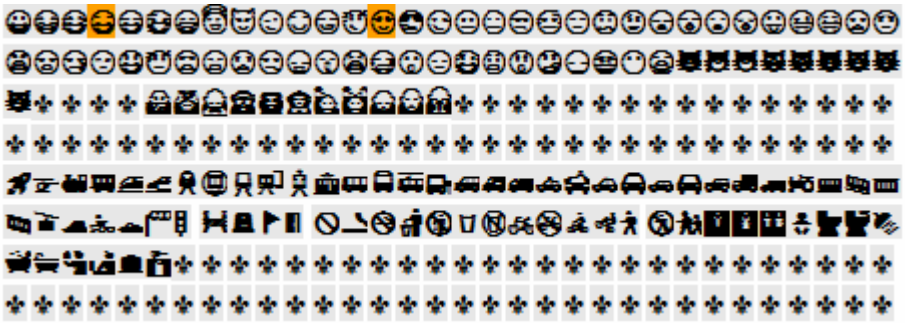
This does not delete the default character that is used when a character is not present in the font.



Change to Plane 1

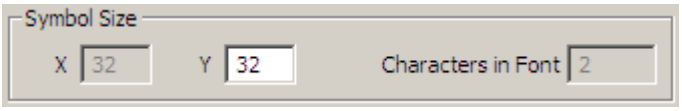


Scroll to F6xx and use the Ctrl and mouse to select 0x1F603 and 0x1F60D:



Press **Insert 2 Characters**.

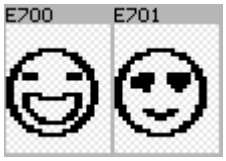
Keep the default size 32



Press **OK** to create the font.

Press **Show CheckerBoard** to show the transparent areas:

The font is now a 4 Bit palette font with transparent background:

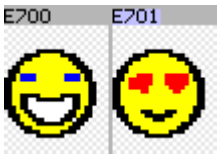


In the 16 bit version of IconEdit the two emoji are relocated automatically to the Unicode private area to stay inside the 16-bit address room, in the 32 bit version use the **Move High Plane Symbols** commands in the **Script & Symbols** window to move them.

Right click the first emoji to enter **Character Edit** mode and show the color palette:

Use **Flood Fill** to change the color of the characters:

Choose the colors and redraw the emoji with new colors, change to the next character with TAB:



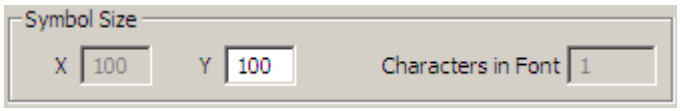
Your Font is ready for saving....

Press the ***Save All As…*** button in the Main tool bar to save the font as *Emoji.c* in the proper directory. The Sym and Code Point Character List files are saved automatically.

2: A range of faces in high plane with anti alias by squeezing

Floodfill and anti alias smoothing together are described in chapter **06.D How to Combine Smoothing with Floodfil**l, here we will do floorfill and smoothing separately by colorind and then squeezing an oversize font.

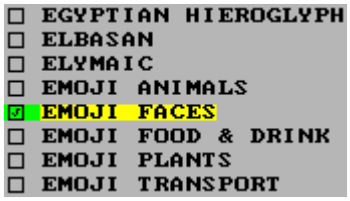
Change size to 100:



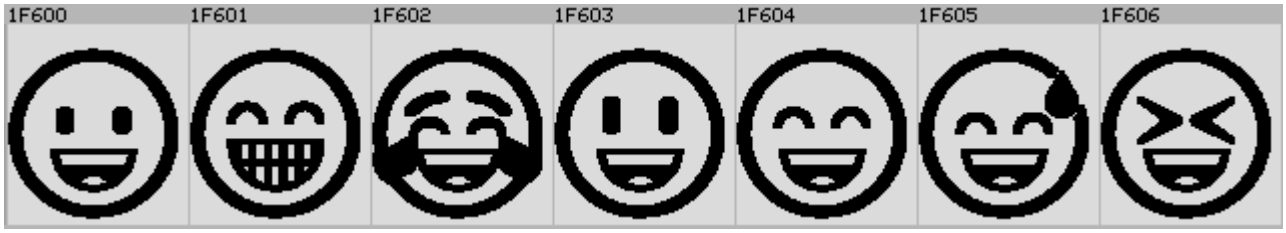
Press **OK** to create a font with only the default character.

Switch to **Scripts & Symbols**:

Click **Emoji Faces**:



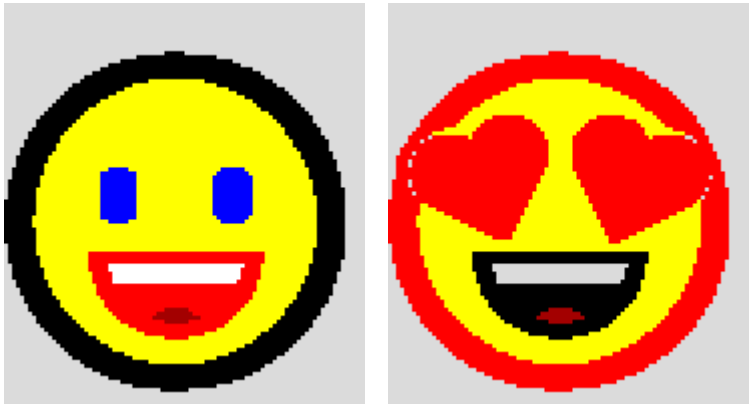
This creates a font with the faces in the plane 1:



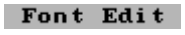
Change to **Character Edit**:

Use Tab to find the emoji you want to color, and activate **Surface Flood Fill**:

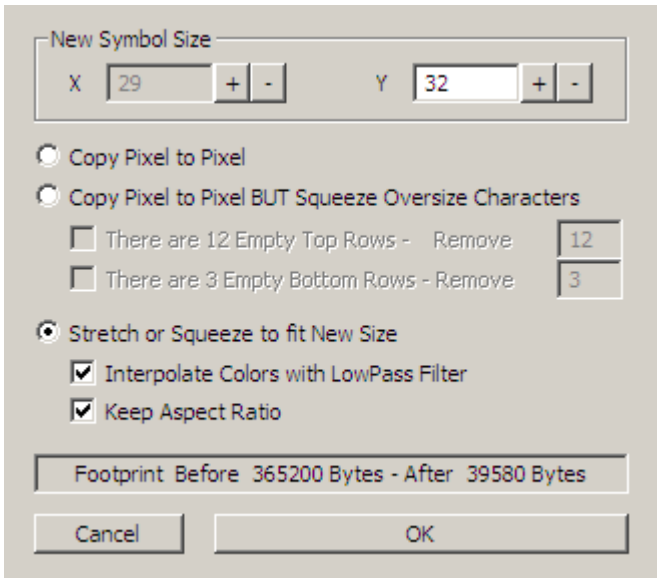
Choose colors other then the transparent color and fill:



When you are finished coloring the ones you want change to **Font Edit**:



Press **Resize All** to squeeze the font:



Press OK:

The font is now 32 bit high and anti aliased with transparent background

Your Font is ready for saving....

Press the ***Save All As…*** button in the Main tool bar to save the font as *Faces.c* in the proper directory. The Sym and Code Point Character List files are saved automatically.

D: How to Make a Text Optimized Font for Text in many Languages

The Code Point Character list system makes it possible to make ROM optimized fonts to fit texts with Unicode characters. Only the characters actually needed to write the text is included in the font, and this can lead to substantial savings in memory space.

This example uses several texts about astronomy in different languages.

Most fonts in Windows support a large number of different languages as Unicode.

The site www.unicode.org contains the official information about how and where the characters for a specific language or alphabet are encoded.

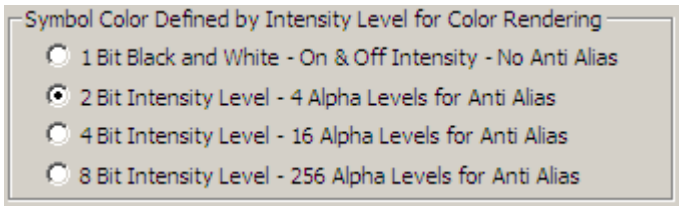
In this example we will use Armenian and Mkhedruli.

Open the first text file. The order in which the text files are opened is irrelevant.

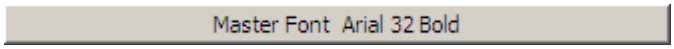
Press the ***File Open*** button in the Main tool bar:

Choose *Armenian.txt*. This opens the ***Create New Text Font*** dialog box:

In the color version of IconEdit there is a choice of Color Mode:

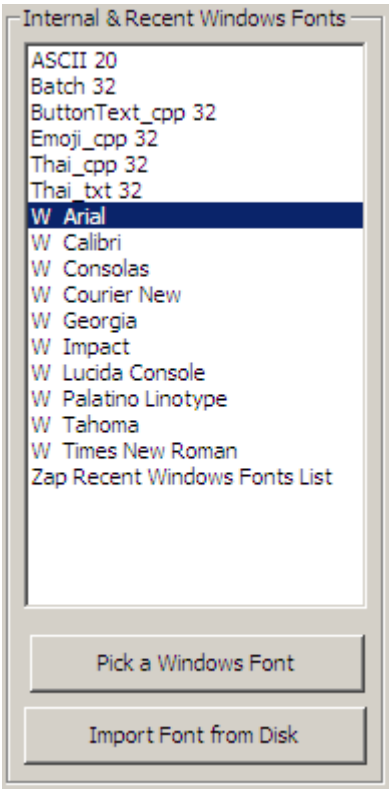


To select a master font press the ***Change Master Font*** button:

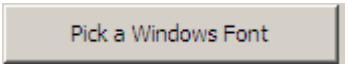


The exact text on the Master Font buttons may vary depending on the operating system version and prior settings in IconEdit.

First, select a Windows font either from the Recent list or pick a new Windows font:



If the recent fonts are not ideal, pick another Windows font as this example shows:



This opens the Windows font picker, the appearance of which is depends greatly on the Windows version, Windows native language, what kind of foreign language support is installed, and any additional non-Windows fonts added later.

Fill in the 3 top fields to choose for example Times New Roman, Bold and 24. The last field is height of the whole Character including top and bottom white space. This is the height of the each symbol in pixels.

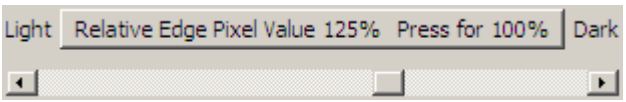
Times New Roman Bold 24

Press OK:

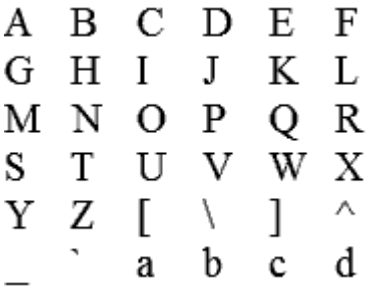
Check chosen Windows font:

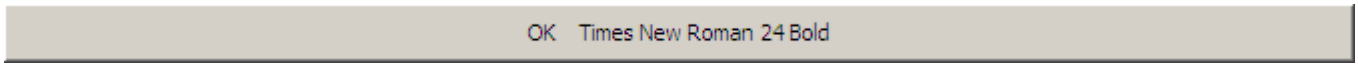
This is a grey-tone vector font, and the conversion to black & white or to grey level can be fine trimmed.

Characters with diagonal lines like / @ A W are usually the most critical.



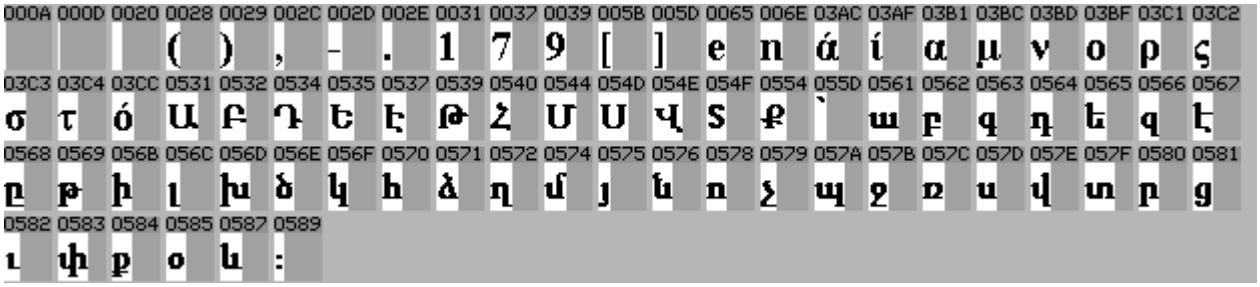
Or remove Zoom to see more characters together.





Press OK.

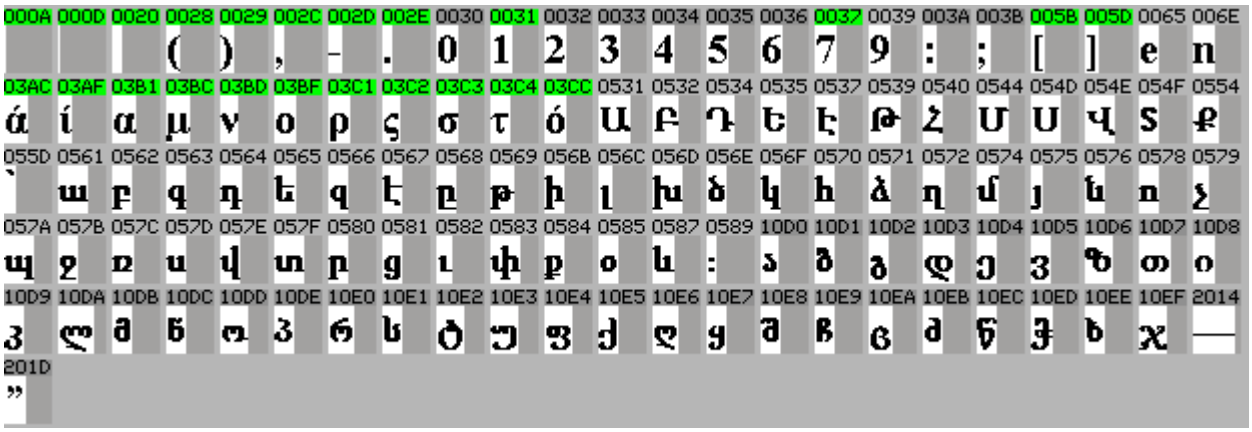
The *Armenian.txt* font looks like this:



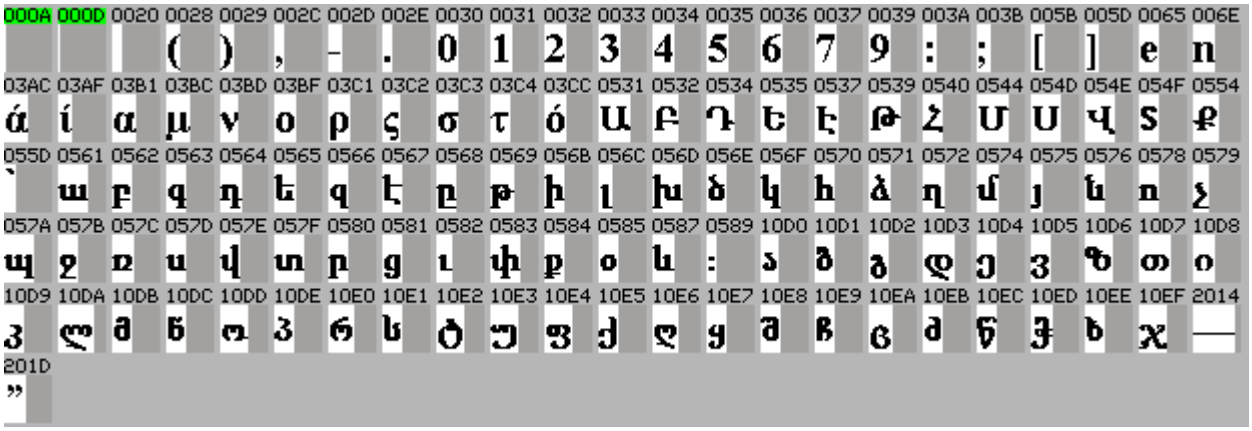
The program does not do any filtering of the text, so Carriage Return and Line Feed are still there. The new characters are marked in dark grey. The next languages have to be added to this font this way:

Text -> Import Text or Text Catalogue to Mark or Create Characters…

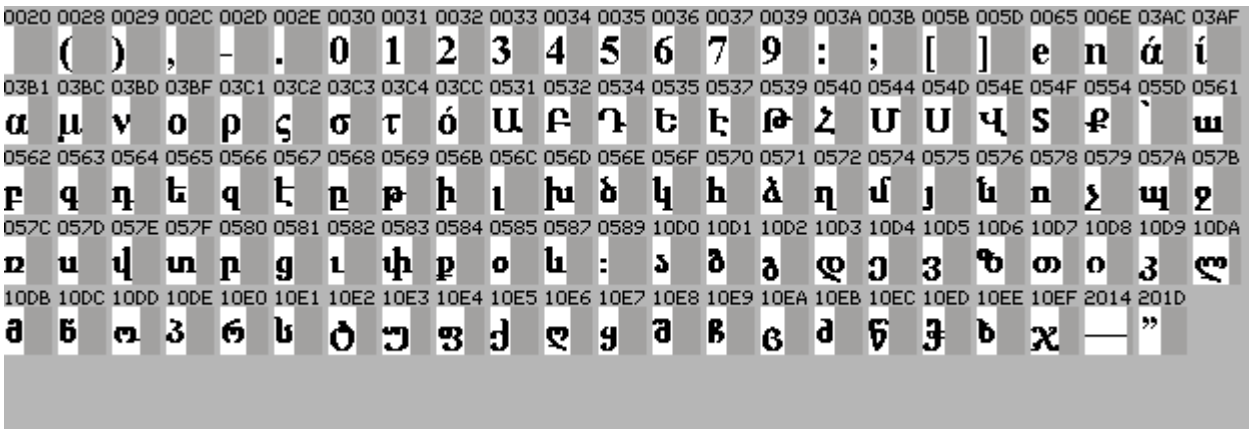
Open *Mkhedruli.txt* and answer **Yes** to **New Additional Characters** to get new characters marked in dark grey. Characters already present, but used again are marked in green:



The control characters below 0x0020 are not needed, mark them with the mouse:

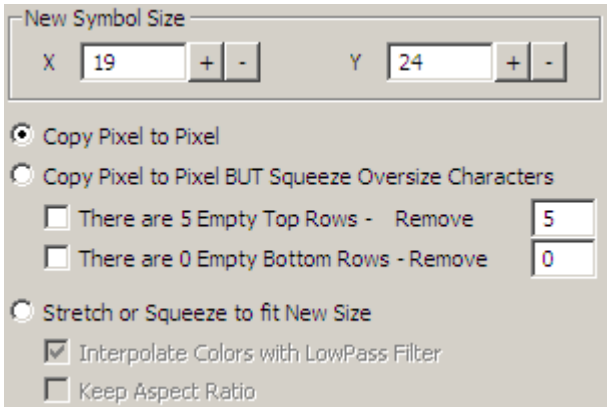


And ***Delete***



To remove surplus unused pixels press ***Resize All Symbols***:

The smallest symbol size that can contain the largest character is suggested.



Press OK to reduce the font to its minimum size.

Your combined Font is ready for saving.

Press the ***Save All As…*** button in the Main tool bar to save the pixel data in the *ArmenianMkhedruli.c* file in the proper directory. The Sym and Code Point Character List files are saved automatically.

For quality control and later reference the data can be saved as a Unicode text file:

1: How to Save the Characters in a Font as a Text File

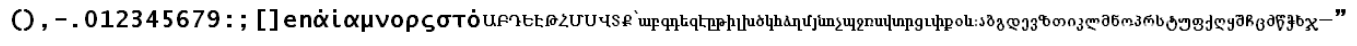
This is a continuation of the previous example.

Any font can be saved as a Unicode text file, all characters including control characters are saved in alphabetical order according to Unicode.

Warning: *Only the character list information is saved in this operation, the character symbols should be saved as sym or ief format.*

Press ***Text -> Export Font Character List as Unicode Text File…*** in the menu to save the text file in the proper directory.

This text file can be opened by any text editor that support Unicode, and will look like this in NotePad:

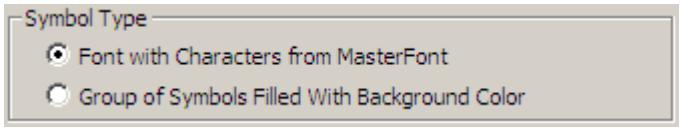


E: How to Draw a Proportional Font with only Numbers

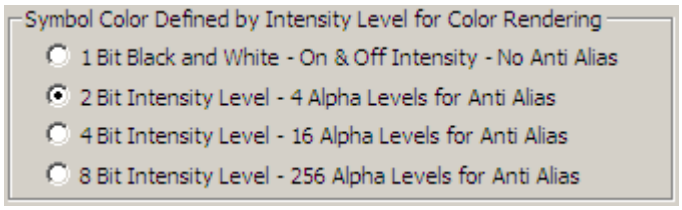
To make a new font press the ***New Font or Symbol Group*** button in the Main Toolbar:

This opens the create dialog box:

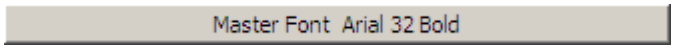
Choose how characters or symbols should be organized and shown initially:



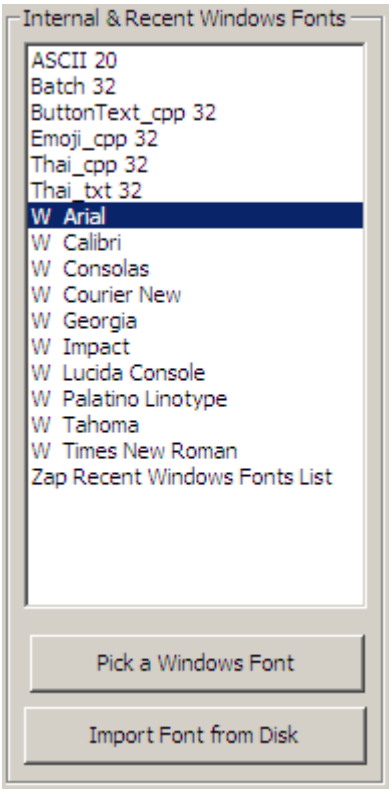
Choose a color format, here 2 Bit Intensity Level for Anti Alias. This option is only available in the color version of IconEdit:



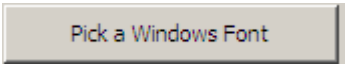
First, select a Windows font either from the Recent list or pick a new Windows font:



The exact text on the Master Font buttons may vary depending on the operating system version and prior settings in IconEdit.



If the recent fonts are not ideal, pick another Windows font as this example shows:



This opens the Windows font picker, the appearance of which is depends greatly on the Windows version, Windows native language, what kind of foreign language support is installed, and any additional non-Windows fonts added later.

Fill in the 3 top fields to choose for example Times New Roman, Bold and 240. The last field is the height of the whole Character including top and bottom white space. This is the height of the each symbol in pixels.

Times New Roman Bold 240

For orientation the character name and the size the glyph is listed under the image of the chosen character:

Press OK.

OK Times New Roman 240 Bold

The numbers and operators range from 0x20 to 0x3F. Open the Character List dialog box:

Create a Character List for Font Directly

One or more characters in addition to the default character may already selected, so:

Clear All Except Default Character

This does not delete the default character that is used when a character is not present in the font.

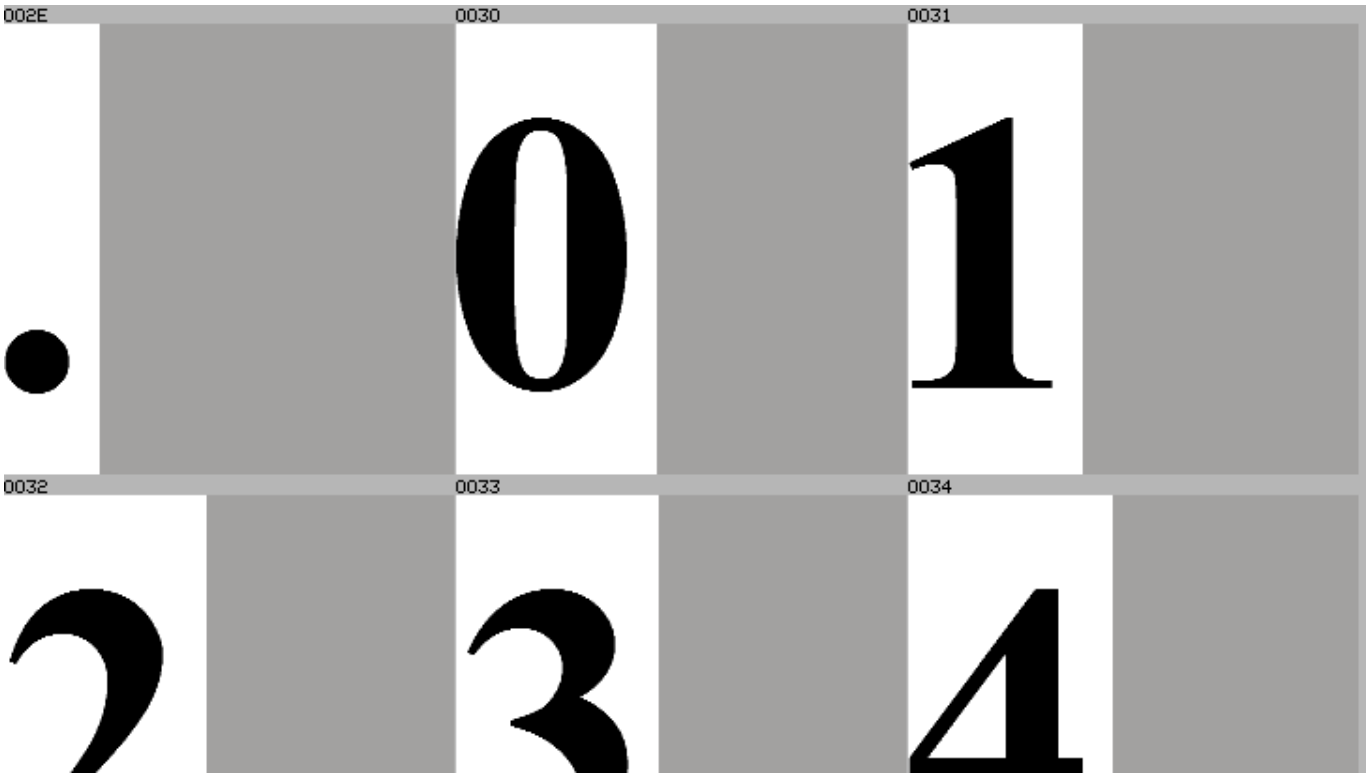
Select . and from 0…9 with Ctrl mouse, this deletes the default character:

Press Insert 11 characters.

Insert 11 Characters

Press OK.

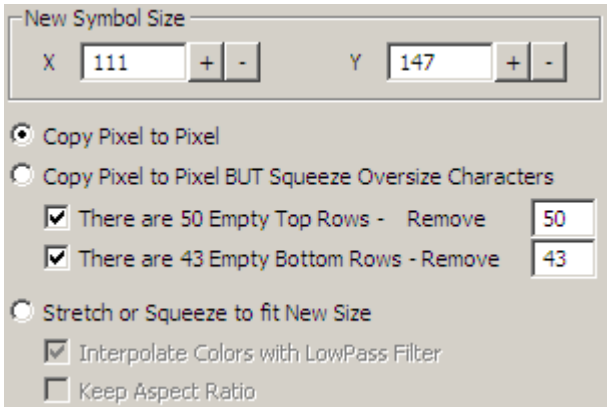
The font now looks like this:



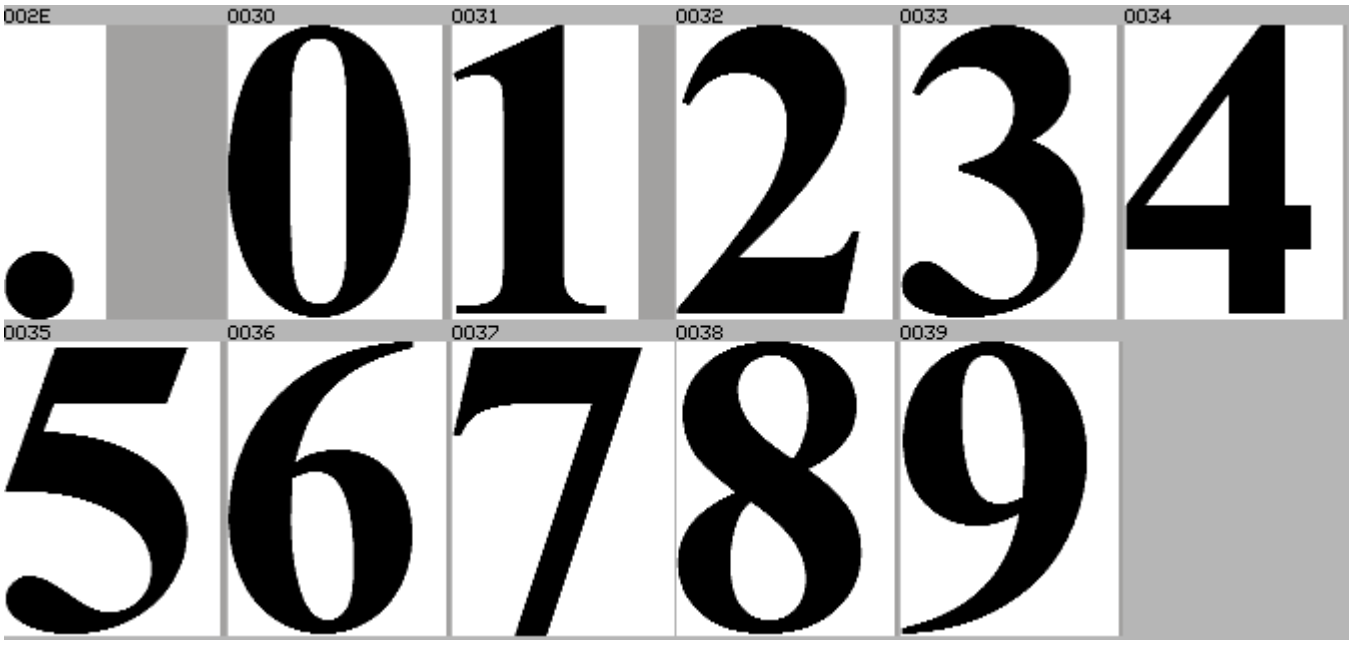
The font now has a lot of unused space on the right-hand end of each character and in top and bottom. Getting rid of that can be done semi automatic or manually for greater control.

1: Semi Automatic Minimizing

To get rid of surplus white space press the resize button:



Press OK.

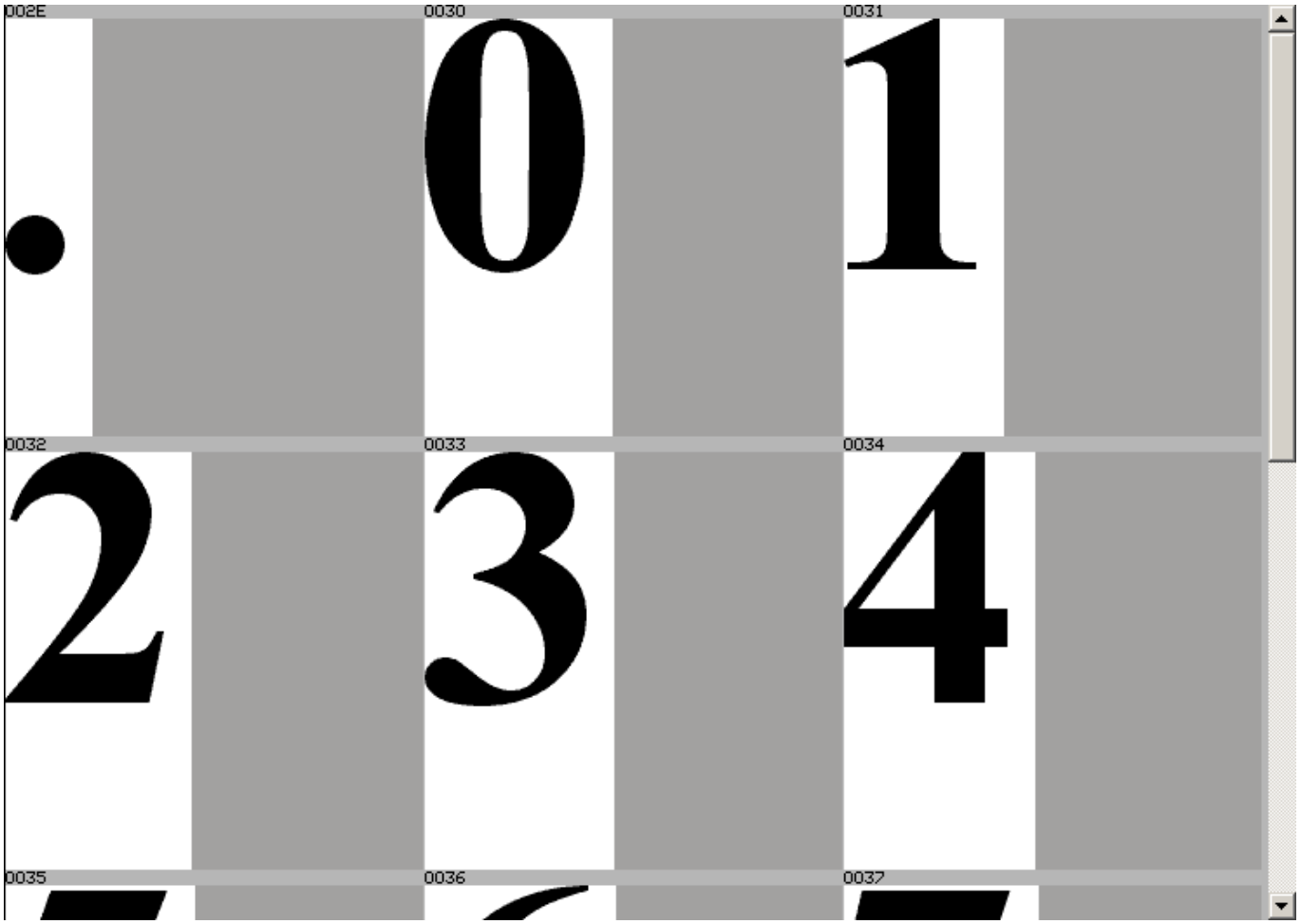


2: Manual Minimizing for Greater Control

To get rid of surplus white space mark the whole font by pressing ***Mark All Symbols***:

Press the ***Move Characters as High as Possible***:

Then press ***Leftset Character***:



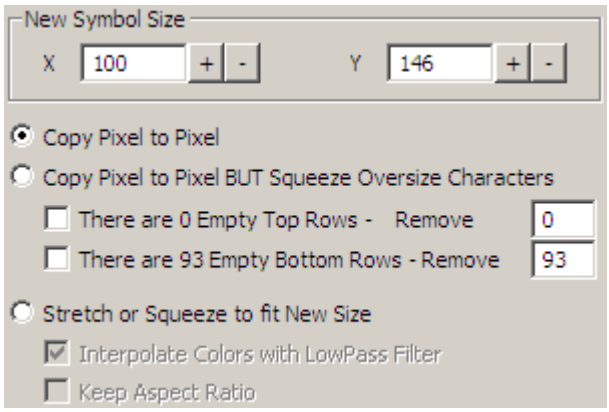
Press ***Get Biggest Character***:

This makes a list of the extremes of the glyphs and in which character they can be found. There may be more than one character with the same extreme. This function lists the first:

The biggest character is 95x146.

Press Cancel to close.

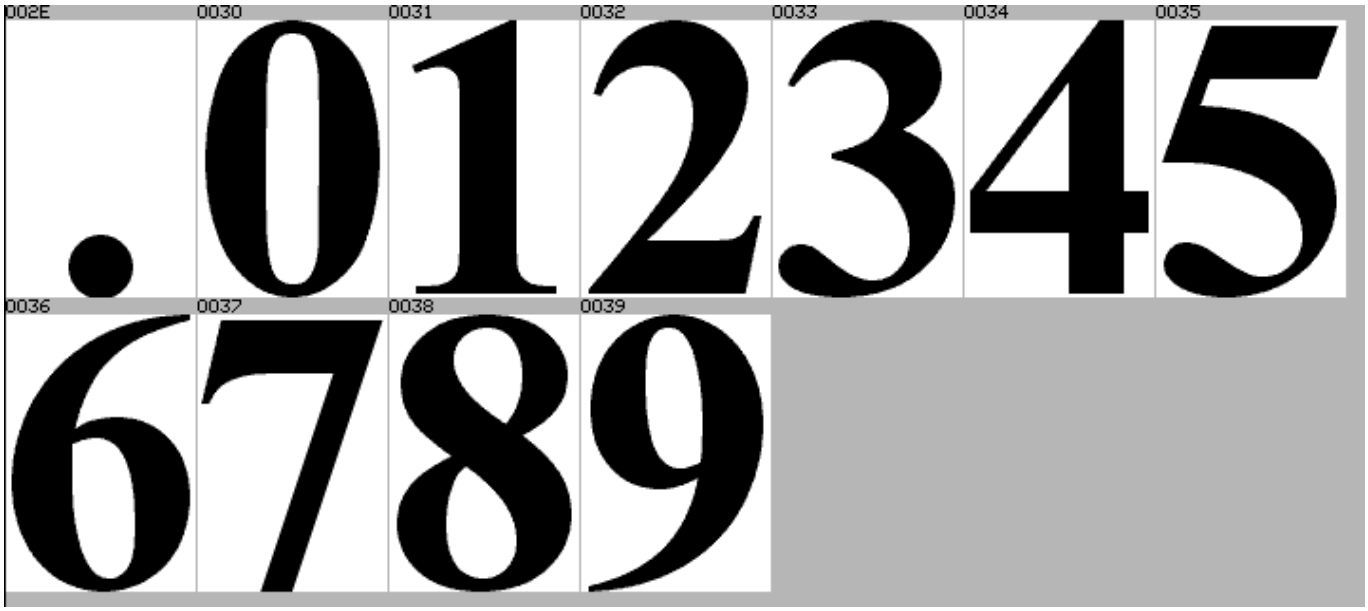
Press ***Resize all Symbols*** to make the font 100x146:



Press OK.

And then press ***Monospace and Center Character***

to make the symbols look like this:



Your Font is ready for saving....

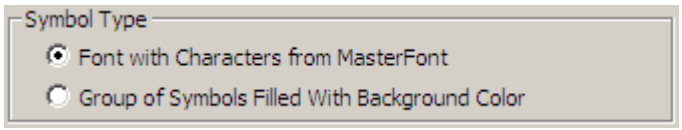
Press the ***Save All As…*** button in the Main tool bar to save the pixel data in the *Ten100x146.c* file in the proper directory. The Sym and Code Point Character List files are saved automatically.

F: How to Draw an 8-bit Classic Font with 256 Characters

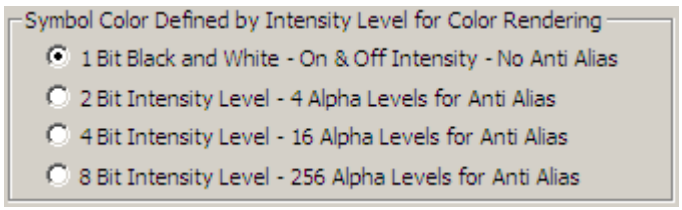
To make a new font press the ***New Font or Symbol Group*** button in the Main Toolbar:

This opens the create dialog box:

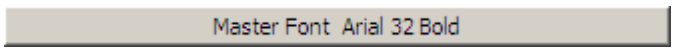
Choose how characters or symbols should be organized and shown initially:



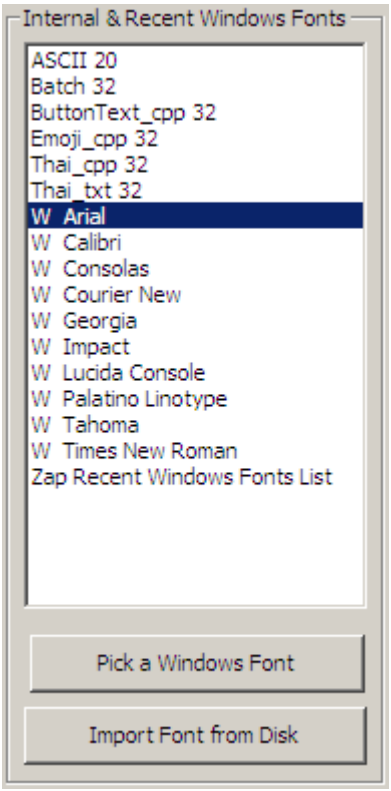
Choose a color format, here Black & White. This option is only available in the color version of IconEdit:



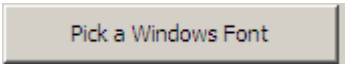
First, select a Windows font either from the Recent list or pick a new Windows font:



The exact text on the Master Font buttons may vary depending on the operating system version and prior settings in IconEdit.



If the recent fonts are not ideal, pick another Windows font as this example shows:



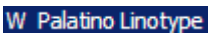
This opens the Windows font picker, the appearance of which is depends greatly on the Windows version, Windows native language, what kind of foreign language support is installed, and any additional non-Windows fonts added later.

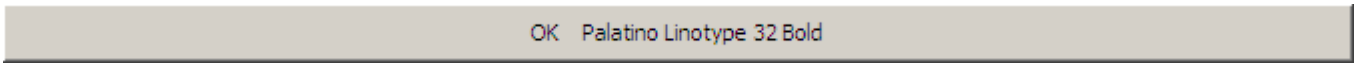
Fill in the 3 top fields to choose for example Palatino Bold and 24. The last field is the height of the whole Character including top and bottom white space. This number will be the height of the each symbol in pixels.

Palatino Linotype Bold 24

Press OK.

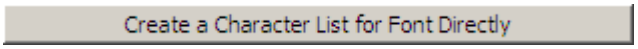
Check chosen Windows font:



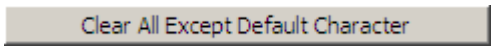


The ISO 8859 fonts normally contain all 256 symbols in the 8-bit address space, and therefore do not need or use a Code Point Character List. In this example we will build the font by using the Unicode to ISO converter, so we start with only the default character.

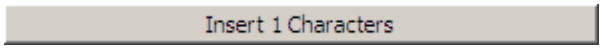
Open the Character List dialog box:



One or more characters in addition to the default character may already selected, so:

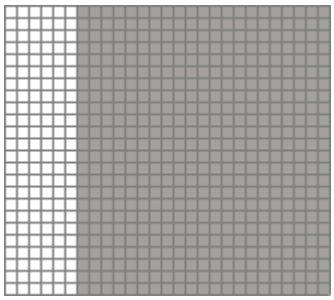


This does not delete the default character that is used when a character is not present in the font.

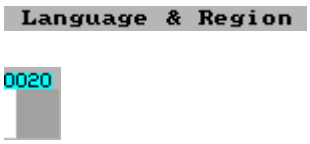


Press OK.

The font has only one empty character and looks like this in ***Character Edit*** mode:



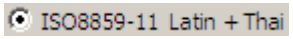
Change to ***Language & Region***:



Press ***Convert 16 bit Unicode Font to 8 bit Font***:

This opens the conversion dialog box:

Choose a conversion:



This shows the result of the conversion. Characters missing in the original font, in this case all except space will be added from the Master Font:

Press ***Convert***:

Convert to 8 bit Font

This draws the missing symbols, change to ***Font Edit*** to view them all:

The font now contains all 256 symbols required to address the characters correctly without Code Point Character List.

Your Font is ready for saving....

Press the ***Save All As…*** button in the Main tool bar to save the pixel data in the *iso11thai.c* file in the proper directory. The Sym file is saved automatically. There will be no Code Point Character List file.

Alternatively the font can be saved as a Code Point Character List font to save ROM space:

1: How to Save ROM Space with a Code Point Character List file

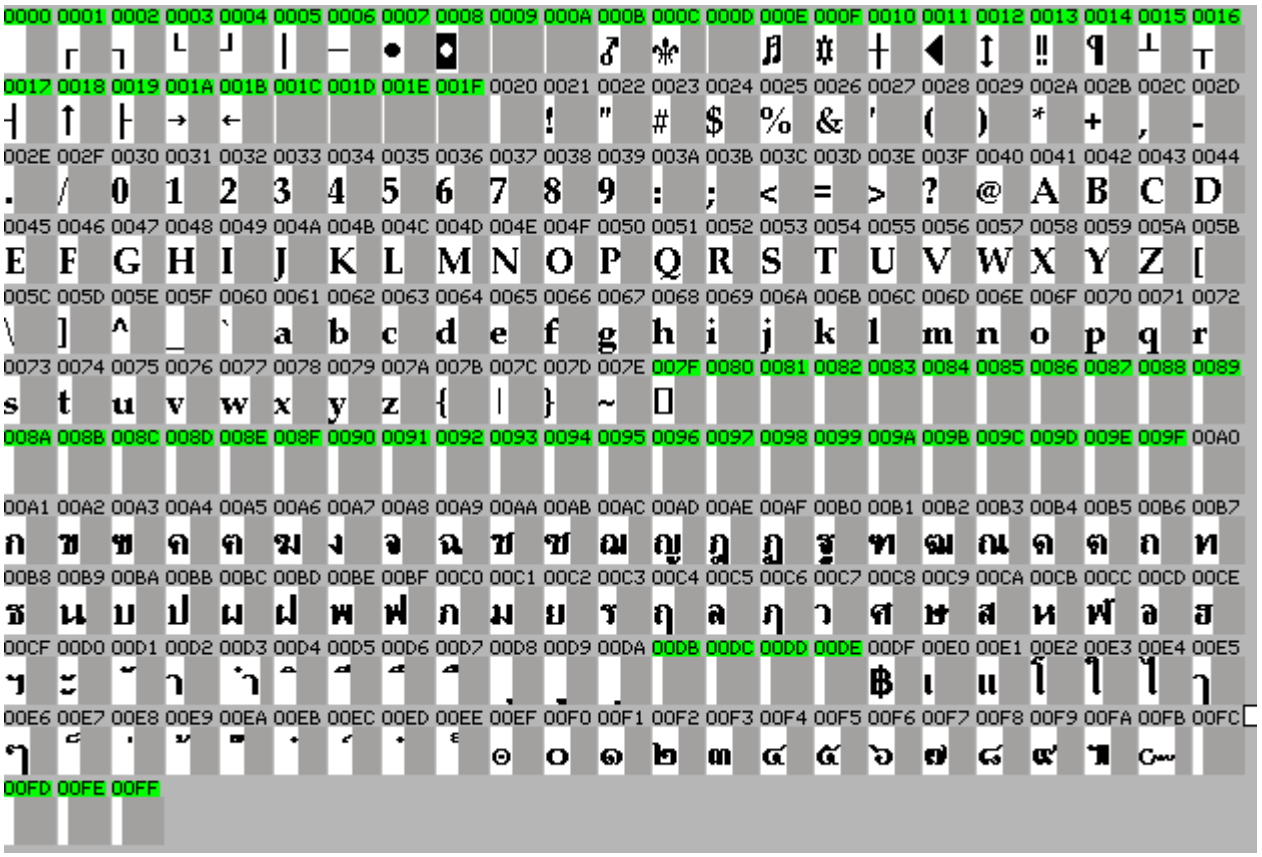
This is a continuation of the previous example.

ISO 8859 does not define symbols for the symbol ranges 0x00-0x1F and 0x80-0x9F. Often some the symbols in the range 0xA0-0xFF are not defined either, in this case a total of 72 symbols.

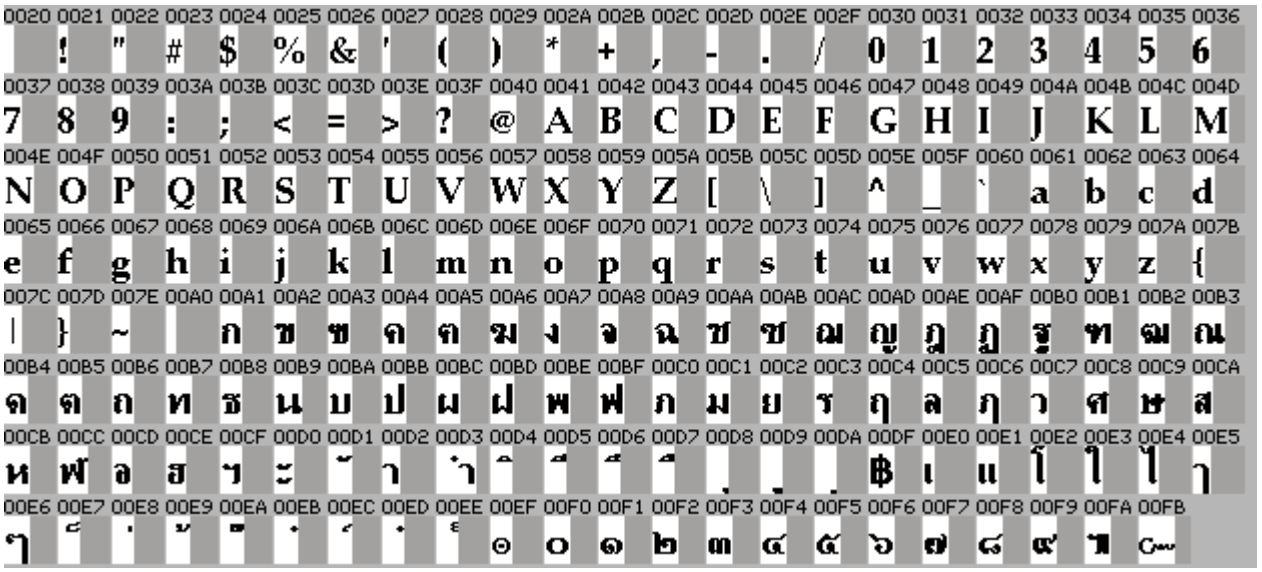
Change to ***Font Edit*** mode

Font Edit

Mark the undefined symbols with the mouse and Shift and Ctrl:



Press ***Delete***:



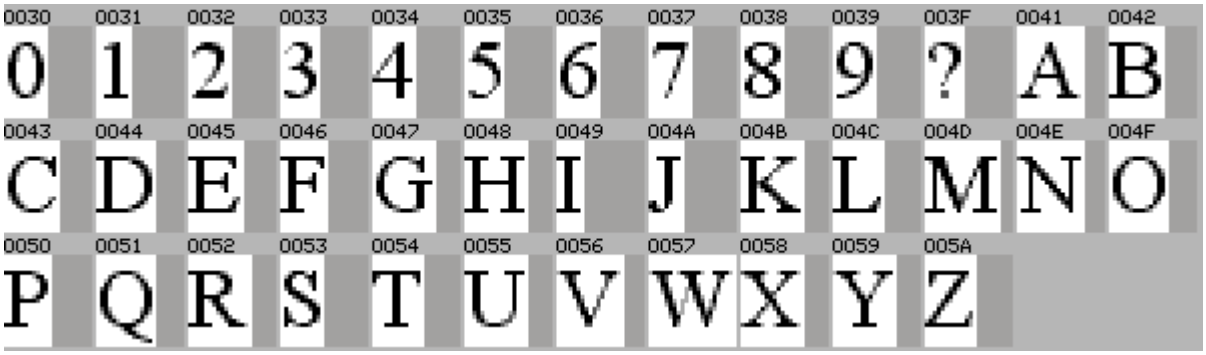
The font now contains only the 184 characters defined by ISO 8859-11.

Your Font is ready for saving....

Press the ***Save All As…*** button in the Main tool bar to save the pixel data in the *iso11thai.c* file in the proper directory. The Sym and Code Point Character List files are saved automatically.

G: How to Use a Code Point Character List as Template for a New Narrow Font

To make a new tall and narrow font based on an existing font such as AZ09:

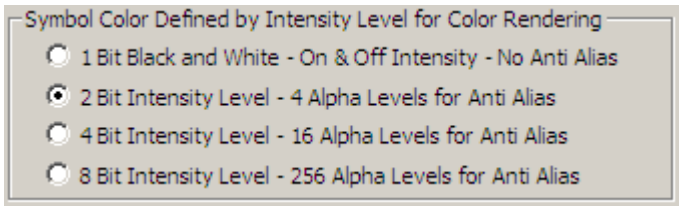


Press the ***File Open*** button in the Main tool bar:

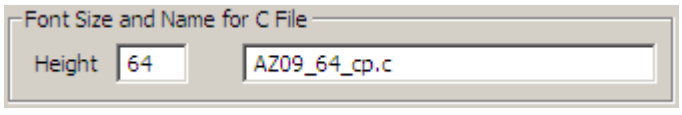
Open the existing Code Point Character List *AZ09.cp*.

This opens the create dialog box:

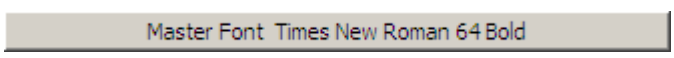
Choose a color format, here 2 Bit Intensity Level for smoothing the edges. This option is only available in the color version of IconEdit:



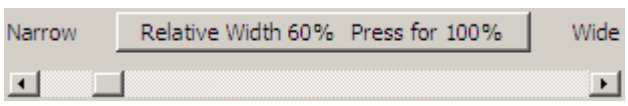
Choose a height and a new name:



The Master Font is automatically updated to the new size; press it to change the rendering:



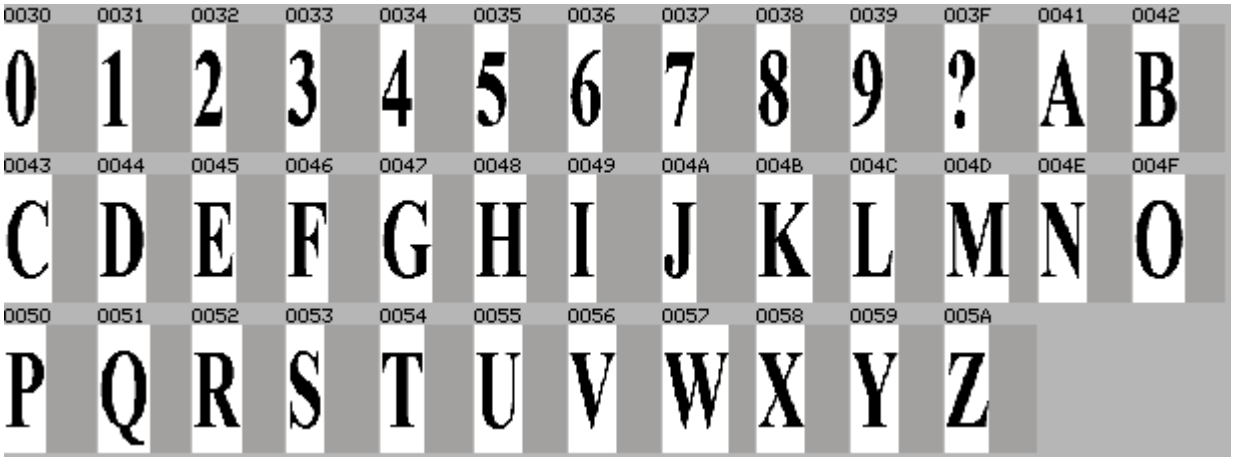
Change relative width from 100% to 60%:



This changes the width of the glyphs:

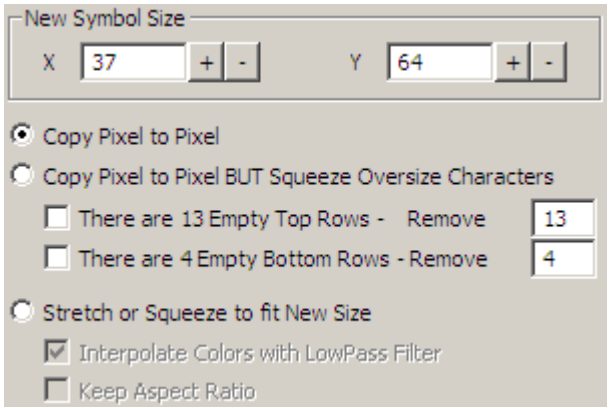
Press OK in the Master Font setup.

Press OK in the Create New Font dialog box:



1: How to Save ROM Space by Removing Unused Space

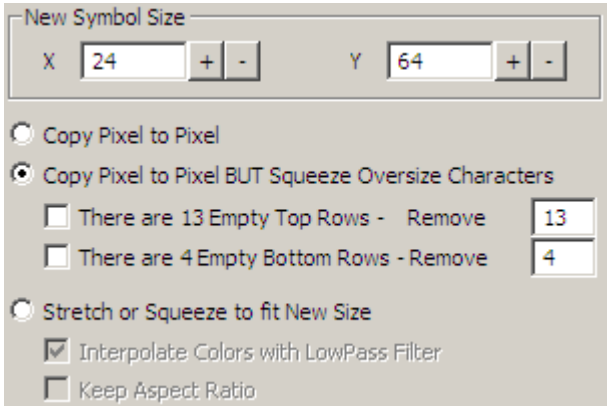
There is normally unused space to the left of the widest character, to remove this press ***Resize All Symbols*** and choose copy with the pre-selected new width:



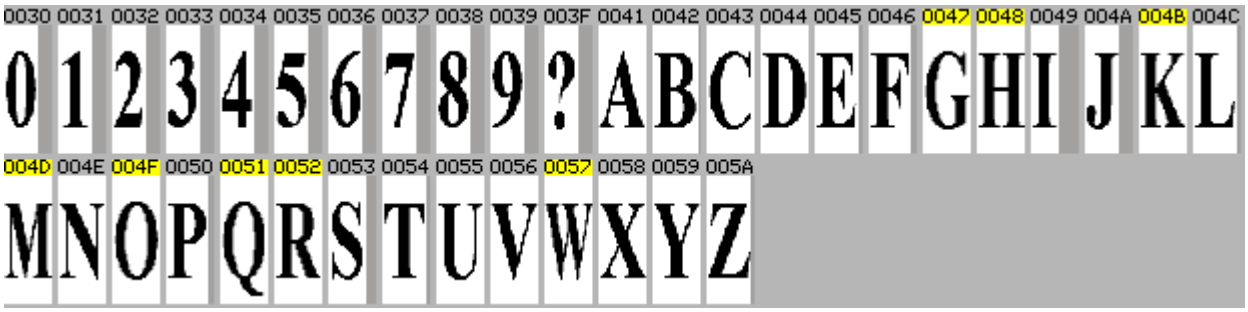
Press OK:

2: How to Save ROM Space by Squeezing to a Byte Border

Press ***Resize All Symbols*** and choose squeeze to a new width at an integral number of 8:

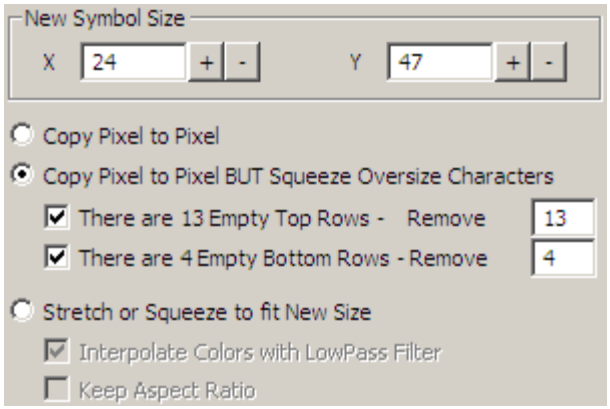


Press OK, the Code Point hexadecimal numbers above squeezed characters are highlighted in yellow:

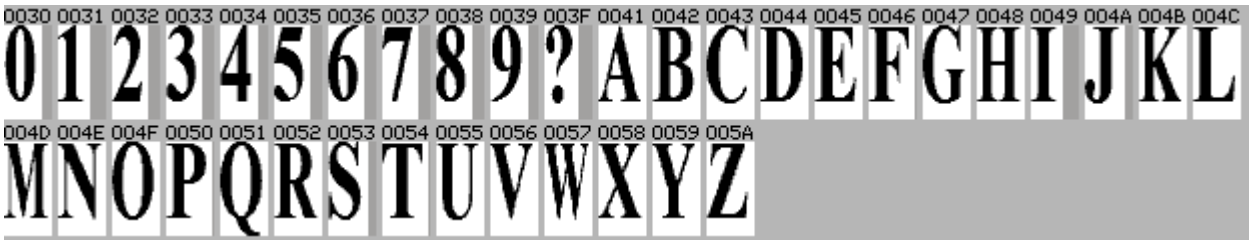


3: How to Save ROM Space by Removing White-Space

Press ***Resize All Symbols*** and choose remove empty rows:



Press OK:



Your Font is ready for saving....

Press the ***Save All As…*** button in the Main tool bar to save the pixel data in the *AZ09_64.c* file in the proper directory. The Sym and Code Point Character List files are saved automatically.

05: How to Build Fonts from Existing Fonts or Groups

All examples in this manual assume that IconEdit is reset to factory defaults except for continuations.

The Code Point Character List system is for removing unused characters and keeping track of used characters. This means that characters can be added to or removed from a font without losing the Code Points (Unicode code-point).

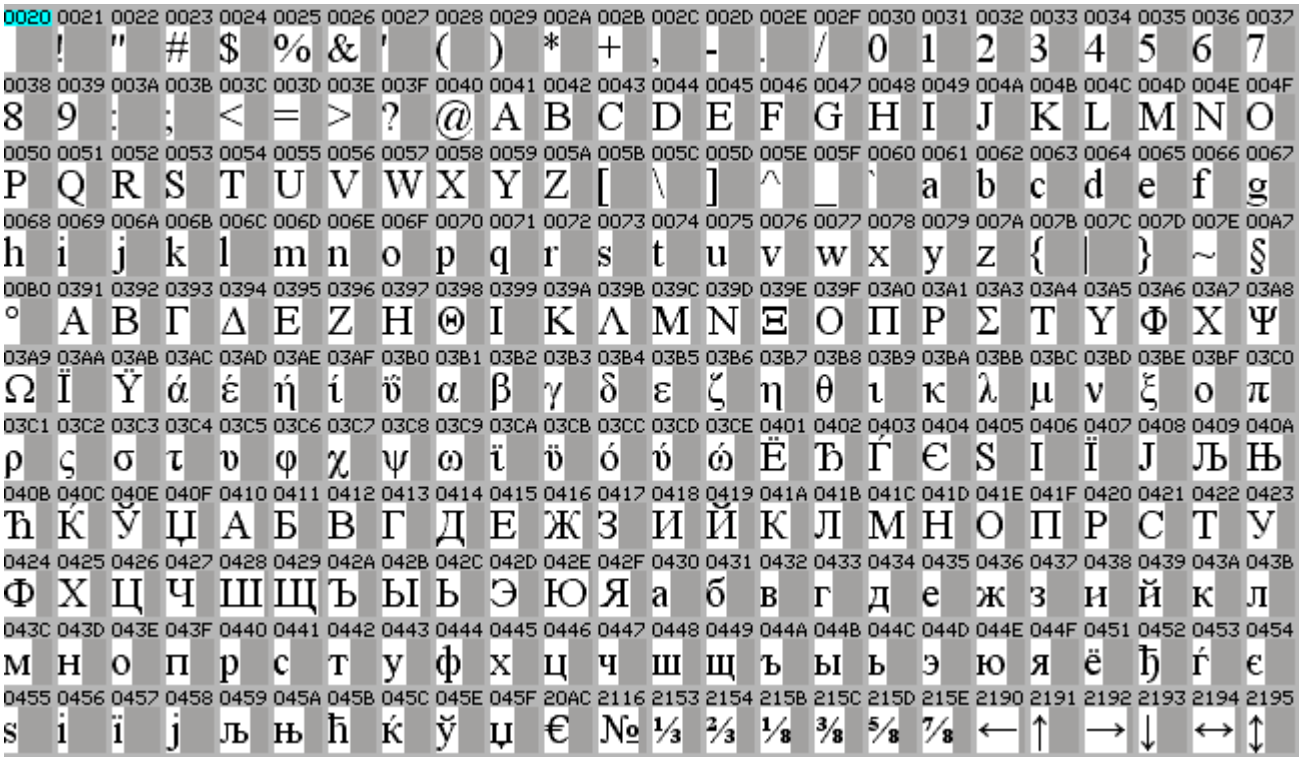
A: How to Reduce an existing Font to fit a Language

The Code Point Character List system is for removing unused characters and keeping track of used characters. This example shows how to make a font with only numbers and language specific letters from a larger font without loosing the Code Points.

This double example will show how to reduce a font to only Bulgarian or Bulgarian and fractions.

Press the ***File Open*** button in the Main tool bar:

Open an existing font, for example *LatinGreekCyrillic.c*.



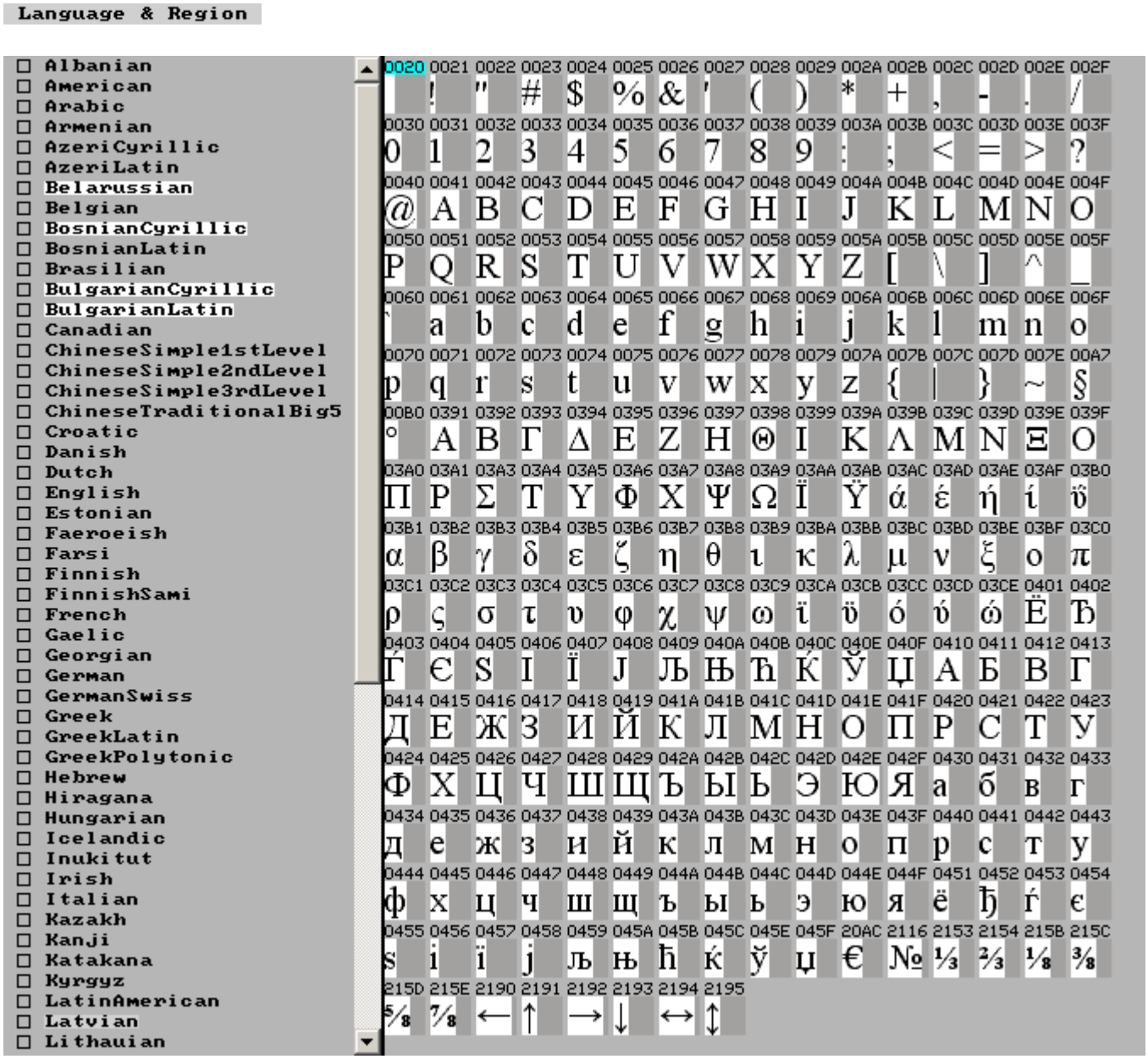
Press ***Modify Font***:

Give the font a new name:

Press OK.

Change to ***Language & Region***

Warning: Language & Regions and Unicode Scripts & Symbols only works with Master Fonts according to the Unicode standard as described at www.unicode.org. See the chapter **04.A.2 How to Check that an Existing Font is According to Unicode.**



Upon entry to Language & Region edit mode, the present font is already marked to avoid accidental erasure of existing symbols in this edit mode:

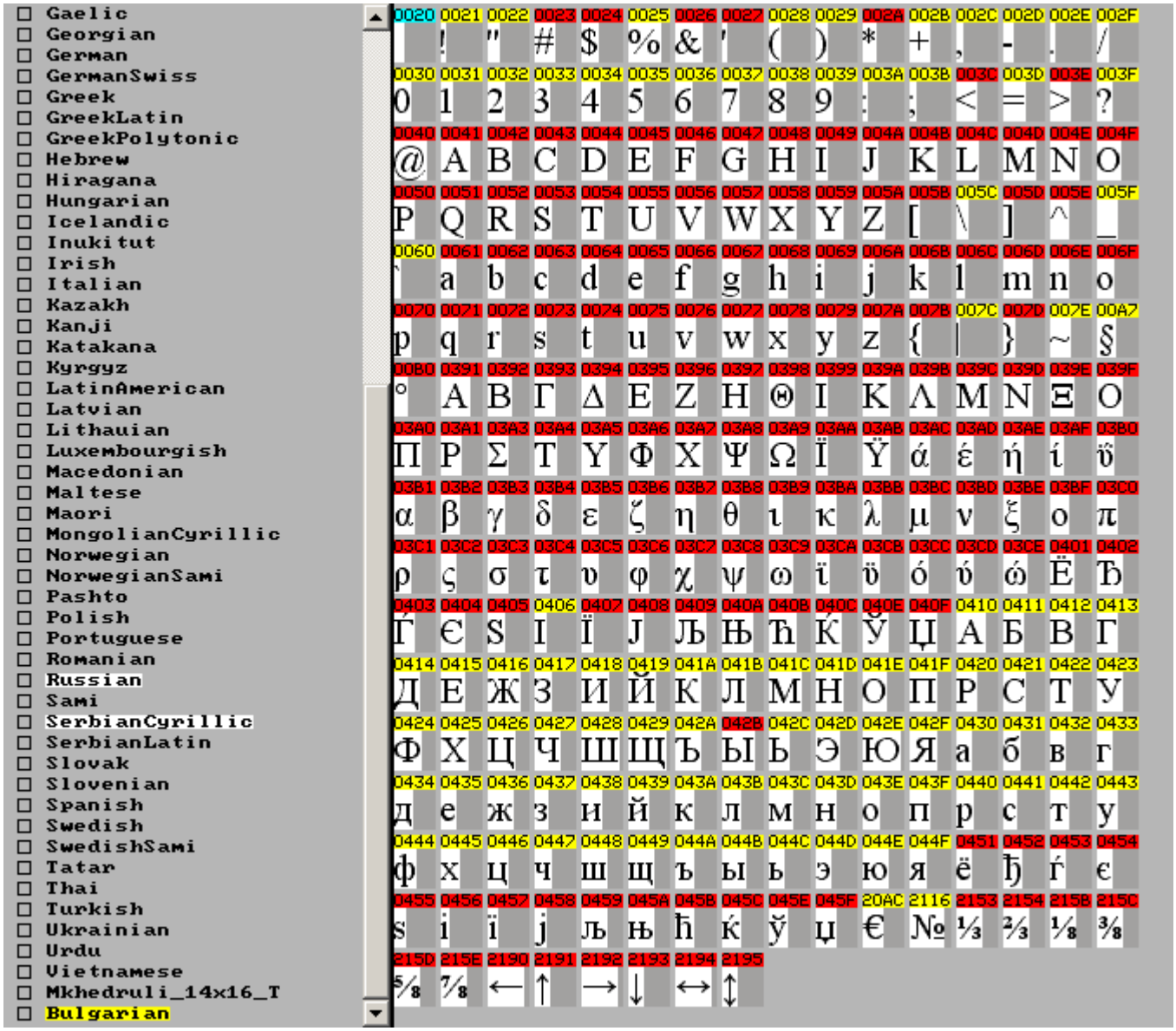
Warning: If this is un-ticked and no other filter is chosen by a tick, all characters except the default character are marked for delete.

The default character is marked highlighting the number in cyan, and the languages already included in the font are highlighted in white.

Press the tick box for a language, for example Bulgarian, which is normally written in Cyrillic:

All characters used by BulgarianCyrillic are marked by highlighting their numbers in yellow.

Remove protection of the original font by un-ticking Bulgarian:

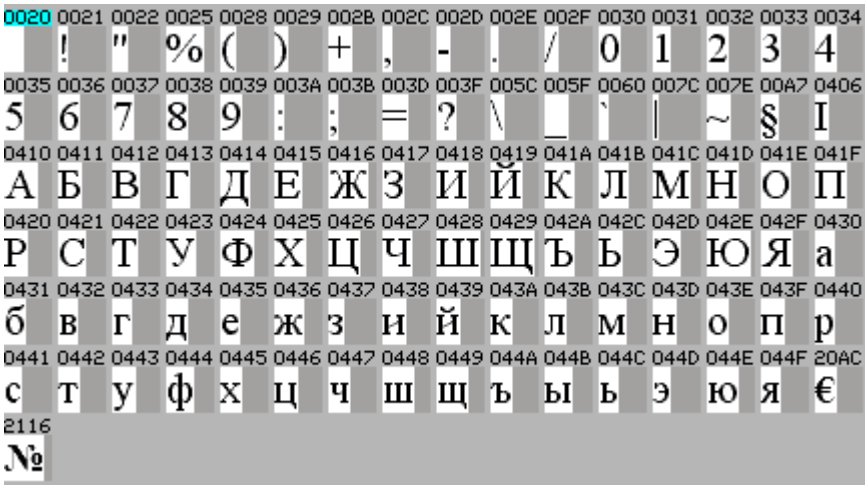


All characters are not used by BulgarianCyrillic are marked by highlighting their numbers in red.

1: How to Save only the Language

If you only want the language press the local delete, otherwise continue at section 2.

Press the ***Local Delete***



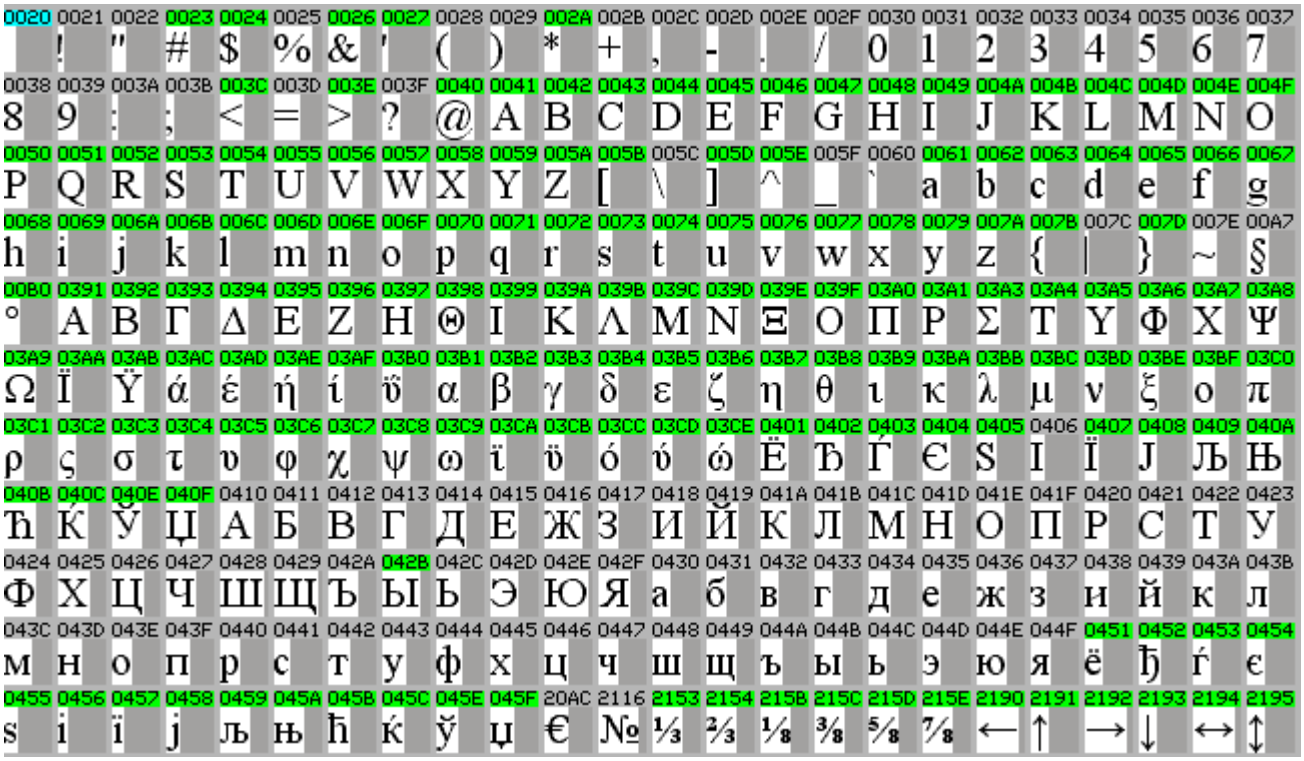
Your Font is ready for saving....

The Data set already has a new name, so press the ***Save Data Set*** button in the Main tool bar to save the data as *Bulgarian.c* in the same directory as the master font came from. The Sym and Code Point Character List files are saved automatically.

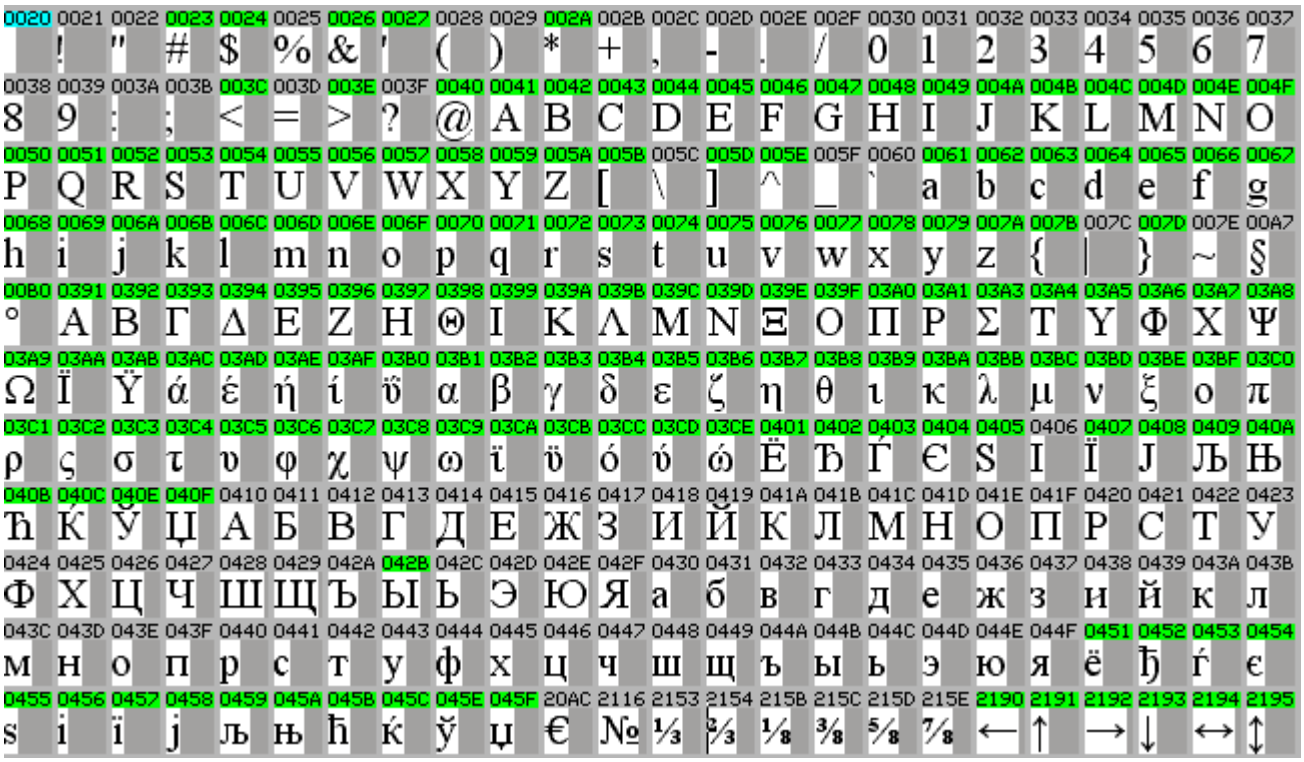
2: How to Save Language and Additional Signs

Change to ***Font Edit***:

All unused characters are now automatically marked in green and could be deleted. But we want to keep the fractions.



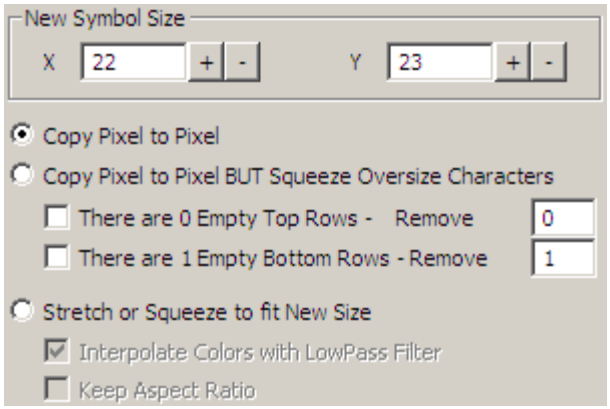
Remove the marks from the fractions with Ctrl-MouseClick on the first (0x2153) and Shift-Ctrl-MouseClick on the last (0x215E):



Delete the marked characters:

To remove surplus unused pixels press ***Resize All Symbols***:

The smallest symbol size that can contain the largest character is suggested.



Press OK to reduce the font to its minimum size.

Your Font is ready for saving....

The Data set already has a new name, so press the ***Save Data Set*** button in the Main tool bar to save the data as *Bulgarian.c* in the same directory as the master font came from. The Sym and Code Point Character List files are saved automatically.

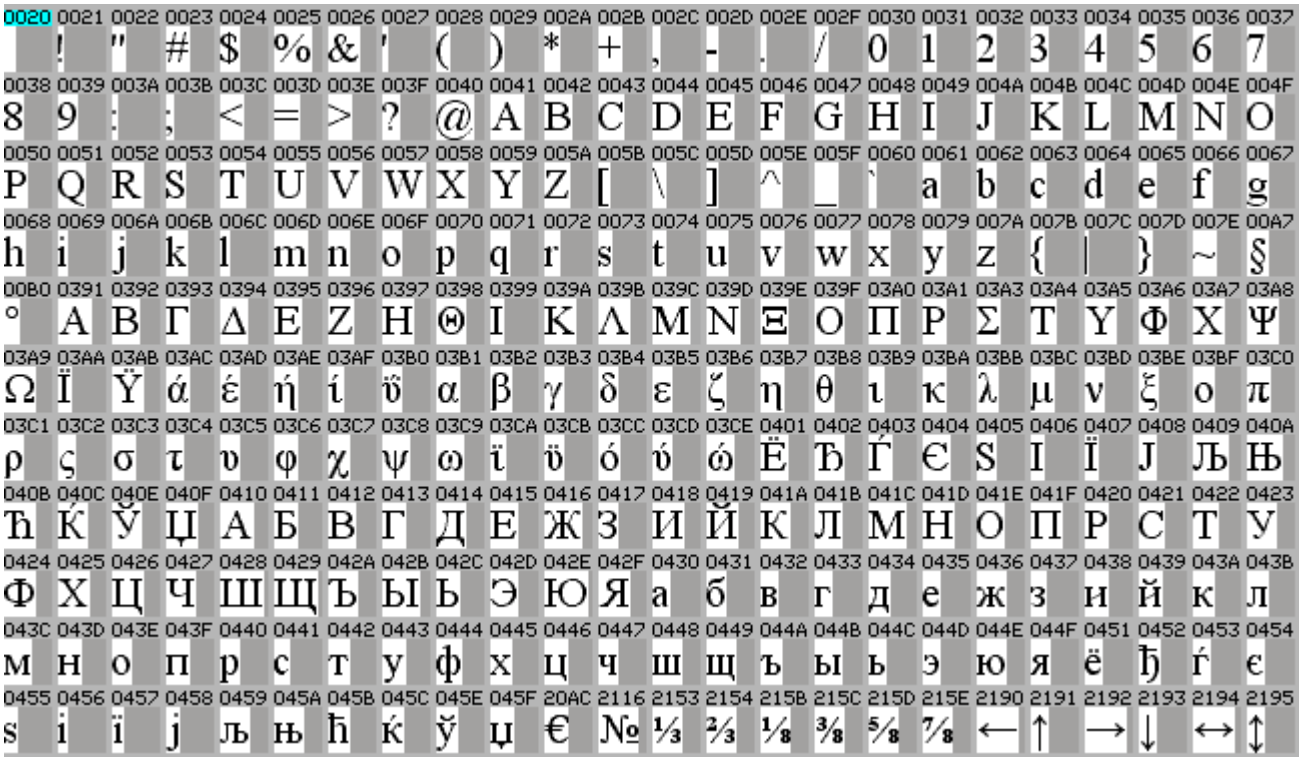
B: How to Reduce a Font to Fit a Text

The Code Point Character List system is for removing unused characters and keeping track of used characters. This example shows how to make a font with only numbers and language specific letters from a larger font without losing the Code Points.

This example will show how to reduce a font to Greek.

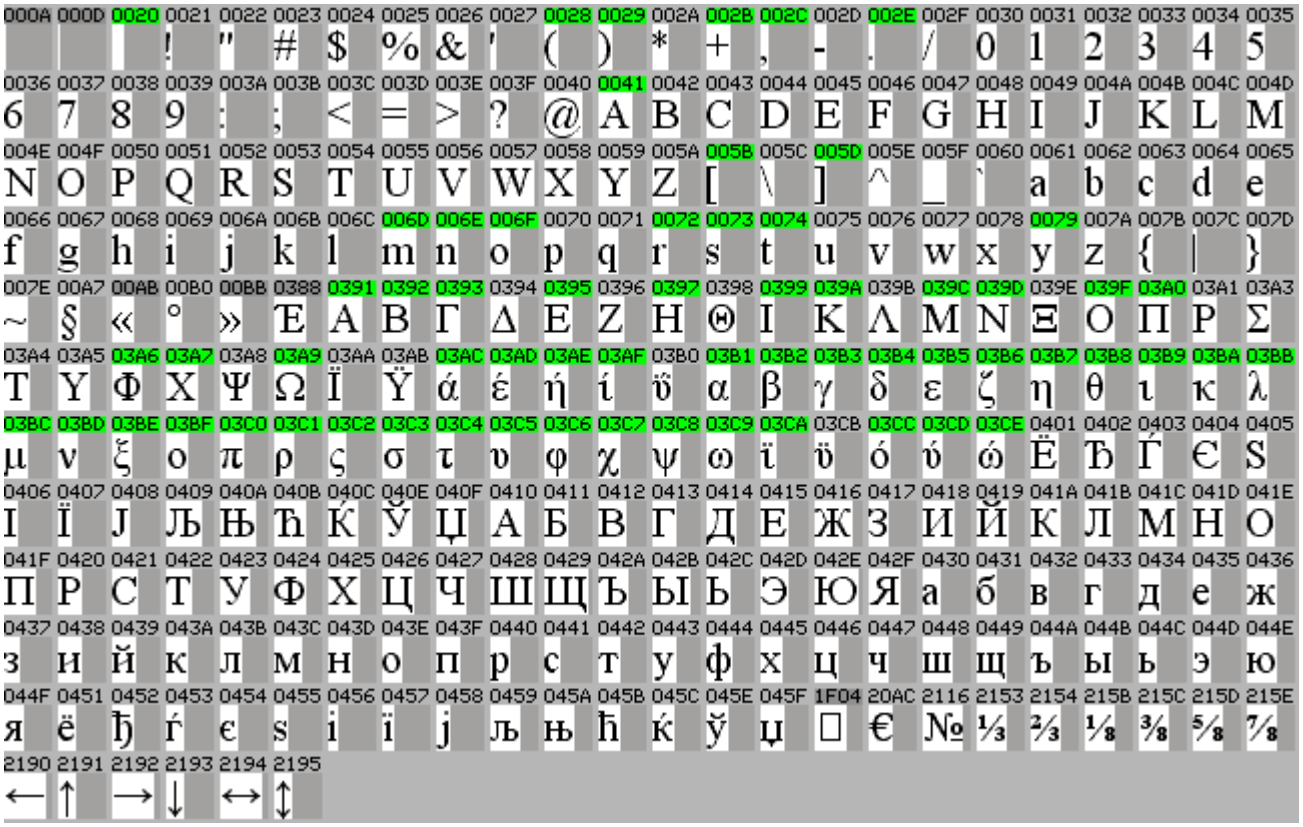
Press the ***File Open*** button in the Main tool bar:

Open an existing font, for example *LatinGreekCyrillic.c*.



Use ***Text –> Import Text or Text Catalogue to Mark or Create Characters*** to open the file *Greek.txt*:

Answer ***Yes*** to **Insert 6 New Additional Characters ?**



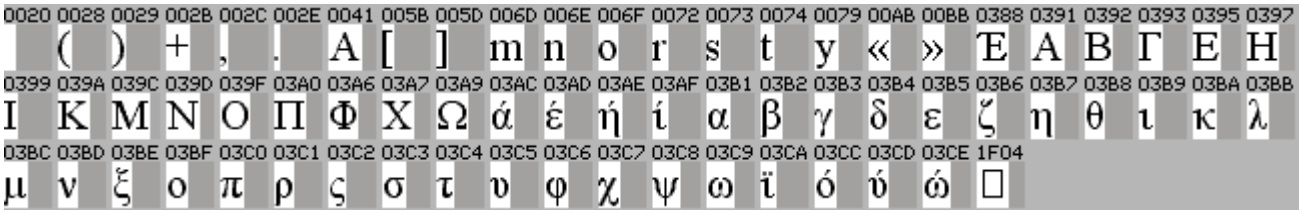
Characters already in the source font are highlighted in green, new characters are highlighted in grey.

Press the ***Invert Markings*** button:

Now the unused characters are highlighted.

Press the ***Delete***:

000A and 000D are control characters, mark and delete them:



The extracted font is ready to save:

Press the ***Save All As…*** button in the Main tool bar to save the pixel data in the *Greek_txt.c* file in the proper directory. The Sym and Code Point Character List files are saved automatically.

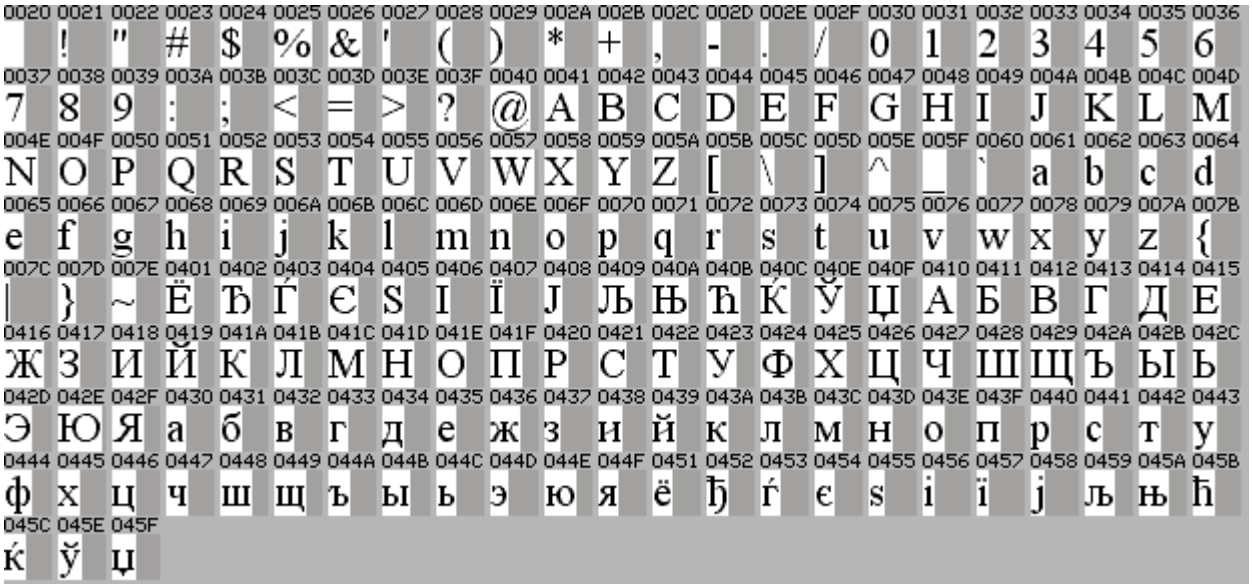
C: How to Make a New Font from an existing Font

The Code Point Character List system is for removing unused characters and keeping track of used characters. This example shows how to make a font with only numbers and capital letters from a larger font without losing the Code Points.

This example will show how to extract numbers and capital letters.

Press the ***File Open*** button in the Main tool bar:

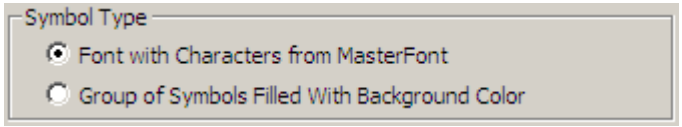
Open an existing font, for example *LatinCyrillic.c*.



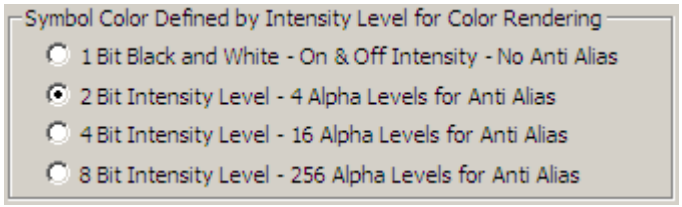
To make a new font press the ***New Font or Symbol Group*** button in the Main Toolbar:

This opens the create dialog box:

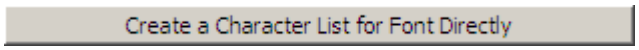
Choose how characters or symbols should be organized and shown initially:



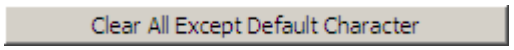
Choose a color format. This option is only available in the color version of IconEdit:



To select the numbers and letters:



One or more characters in addition to the default character may already selected, so:

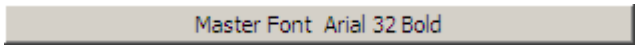


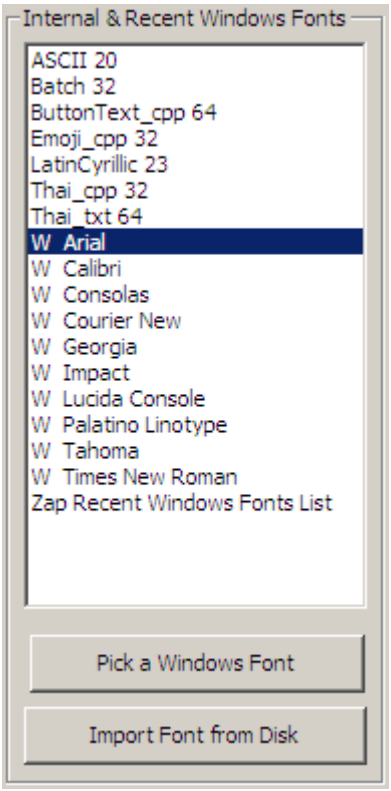
This does not delete the default character that is used when a character is not present in the font.

Select from 0 to 9 with the mouse, and then with Ctrl + mouse select A to Z:

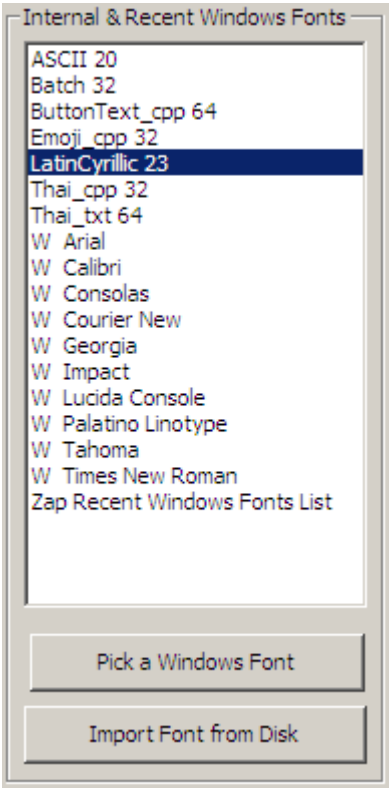
Press Insert 36 Characters.

Choose a master font, in this example to the font already opened:

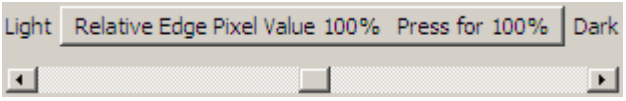




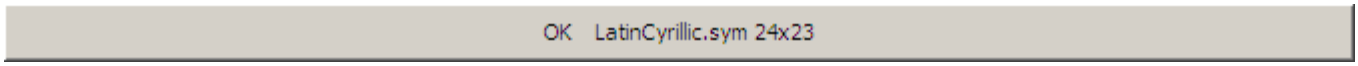
Choose Internal font LatinCyrillic 23:



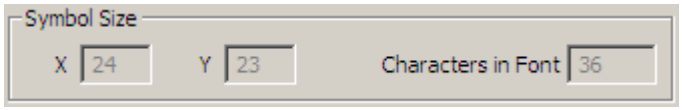
Set edge thickness offset to Zero i.e. 100%:



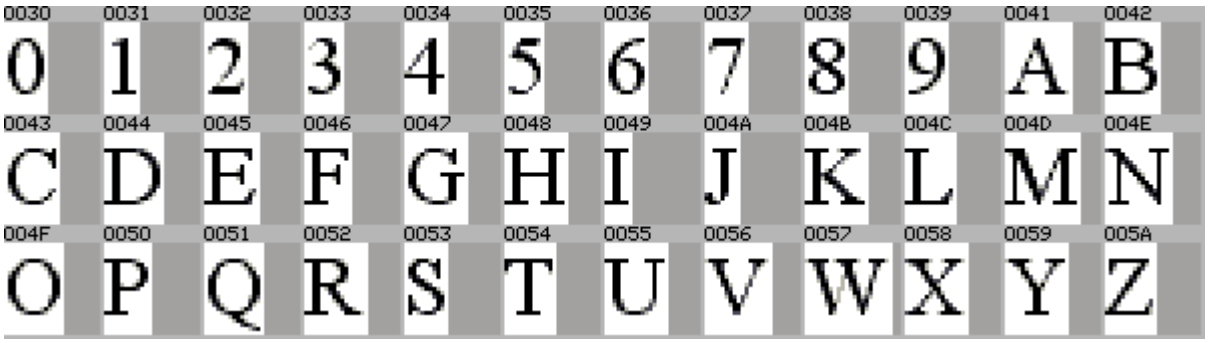
Press OK.



Symbol size and number is hereby defined as:



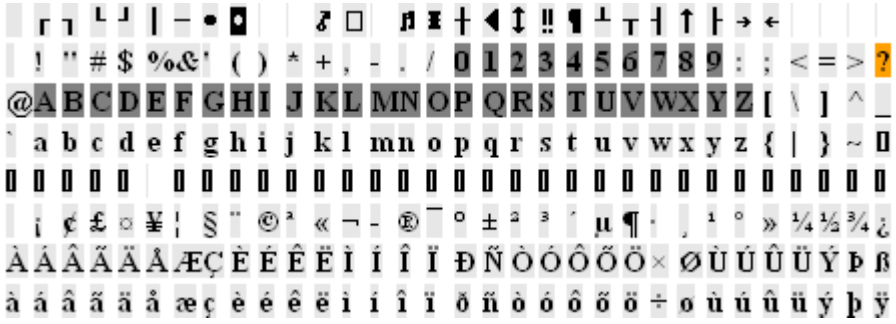
Press OK:



The Code Point Character List system also needs to know what Default Character to show if a character does not exist in the font. It could be a question mark.

Press ***Insert New Character***:

Choose the question mark:



Press Insert 1 Character.

The default character is automatically chosen as the first character in the font, to change this open the ***Code Point Edit View***:

Default character, if any, is marked bright cyan.

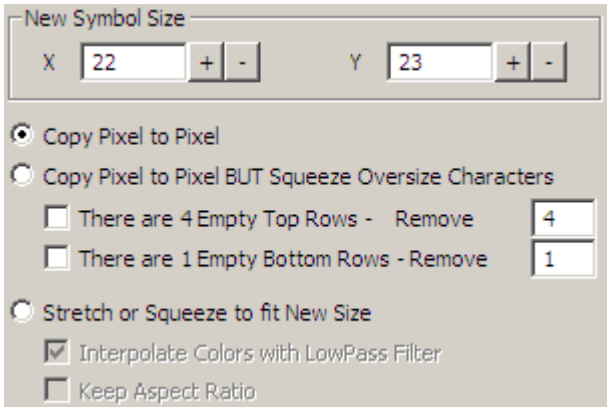
Press ***Choose Default Character with Mouse***:

Click on the question mark to make it the default character:

Switch back to ***Font Edit***:

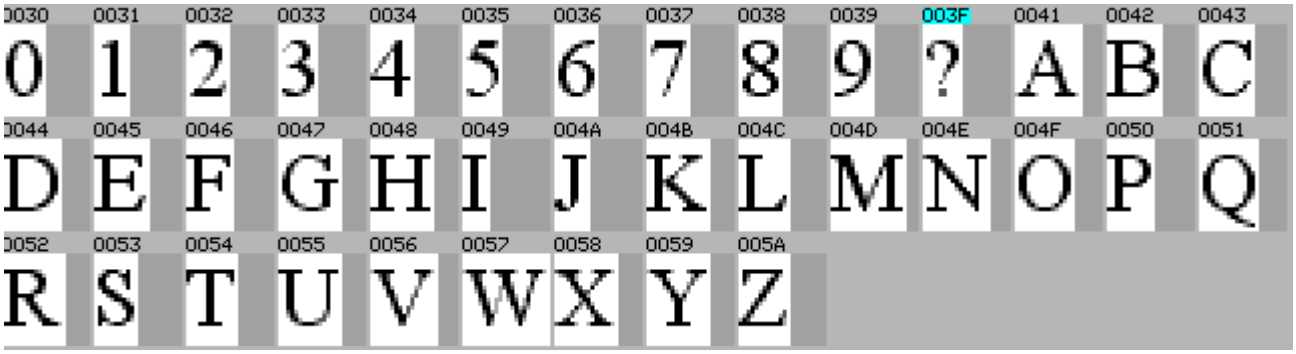
To remove surplus unused pixels press ***Resize All Symbols***:

The smallest symbol size that can contain the largest character is suggested.



The use of **Squeeze Oversize Characters** is only recommended for grey-level fonts, see the squeeze and mono-space continuation of this example.

Press OK to reduce the font to its minimum size.



The extracted font is ready to save:

Press the ***Save All As…*** button in the Main tool bar to save the pixel data in the *AZ09.c* file in the proper directory. The Sym and Code Point Character List files are saved automatically.

1: How to Squeeze and Mono-Space an existing Font Automatically

This example is a continuation and uses the font and settings of the previous example.

In some applications it is preferable that all characters have the same width, it simplifies the calculation of string lengths and reduces flicker when strings are changed. In addition mono-space fonts usually require less memory space because the widest characters are squeezed somewhat.

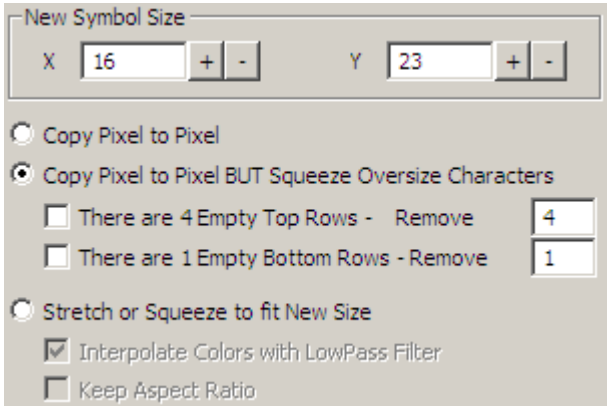
For Black & White fonts we recommend that squeezing is done for one character at a time with the **Squeeze or Stretch Character with Mouse** in **Character Edit** mode to keep track on the deformation of the characters whereas squeezing of Grey fonts can normally be done safely for the whole font as one operation in **Font Edit** mode with the use of **Squeeze Oversize Characters** in the **Resize All Symbols** function.

These two methods are described in chapter 04.A.3 and 05.C.1 respectively.

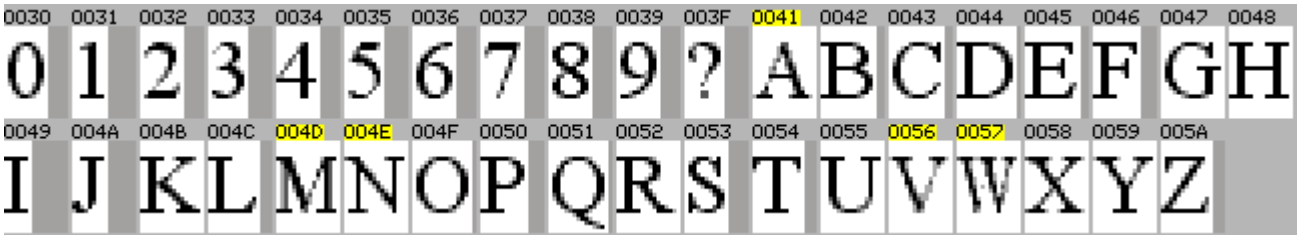
In this example we will squeeze the grey-level font *AZ09.c* from 22x23 to 16x23.

To squeeze the widest characters and remove surplus unused pixels press ***Resize All Symbols***:

The smallest symbol size that can contain the largest character is suggested. Change that to **X = 16** and select ***Copy Pixel to Pixel BUT Squeeze Oversize Characters***.



Press OK to reduce the font to the desired size:



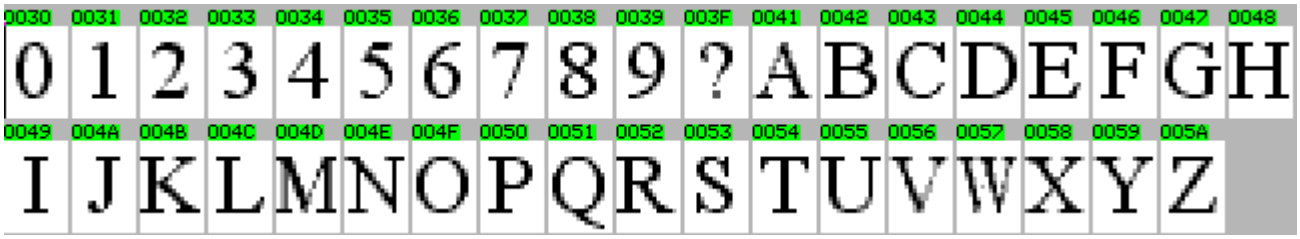
The Squeezed Characters are highlighted in yellow.

The Squeezed Font is ready for saving....

Press the ***Save All As…*** button in the Main tool bar to save the data as *AZ09_S.c* in the proper directory. The Sym and Code Point Character List files are saved automatically.

To make a mono-space font select all characters with the ***Mark All Symbols*** button:

Then mono-space all characters with the ***Monospace and Center Character*** button:



The mono-spaced Font is ready for saving....

Press the ***Save All As…*** button in the Main tool bar to save the data as *AZ09_M.c* in the proper directory. The Sym and Code Point Character List files are saved automatically.

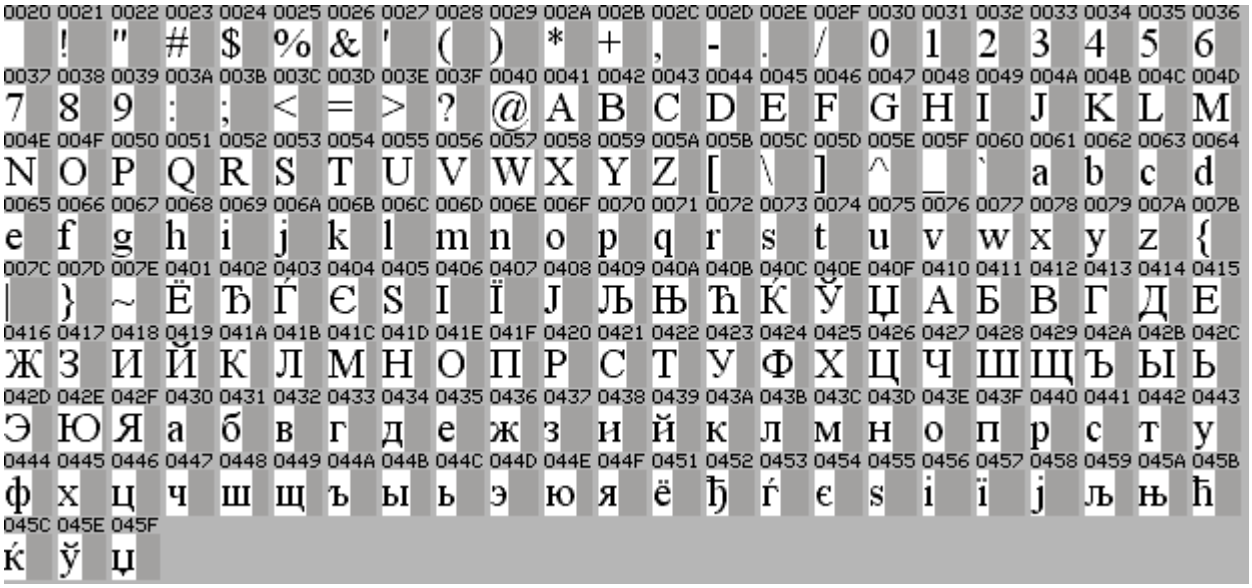
D: How to Make a New Language Font from an existing Font

The Code Point Character List system is for removing unused characters and keeping track of used characters. This example shows how to make a font for a specific language from a larger font without losing the Code Points.

This example will show how to extract Maltese.

Press the ***File Open*** button in the Main tool bar:

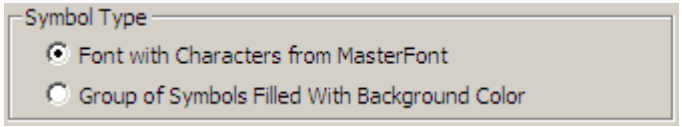
Open an existing font, for example *LatinCyrillic.c*.



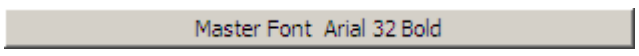
To make a new font press the ***New Font or Symbol Group*** button in the Main Toolbar:

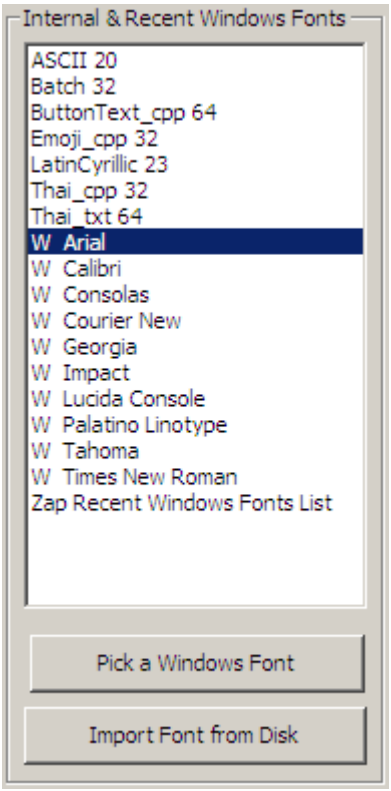
This opens the create dialog box:

Choose how characters or symbols should be organized and shown initially:

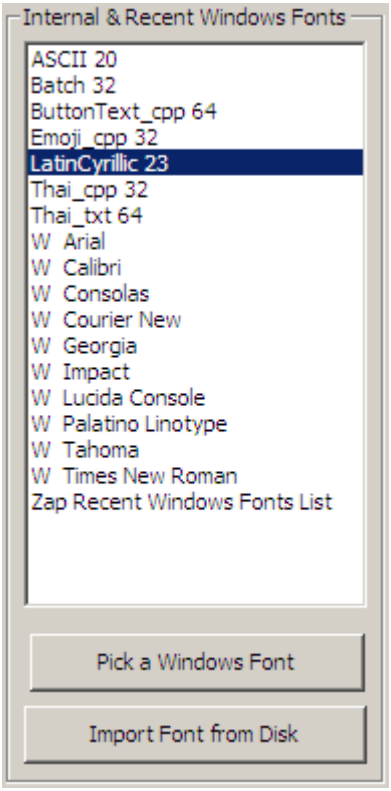


Choose a master font, in this example to the font already opened:

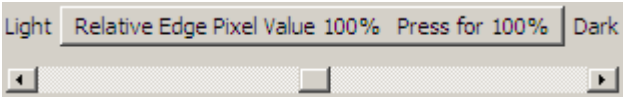




Choose Internal font LatinCyrillic 23:

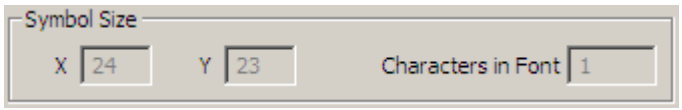


Set edge thickness offset to Zero i.e. 100%:

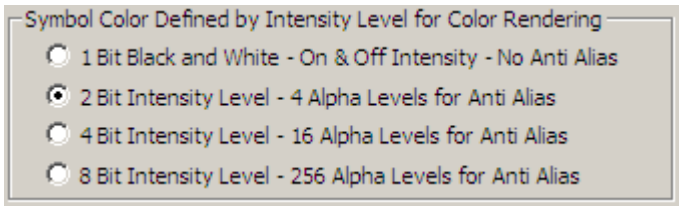


Press OK.

Symbol size and number is hereby defined as:



Choose a color format. The LatinCyrillic 23 is a 2 Bit Intensity Level font, so choose that. This option is only available in the color version of IconEdit:



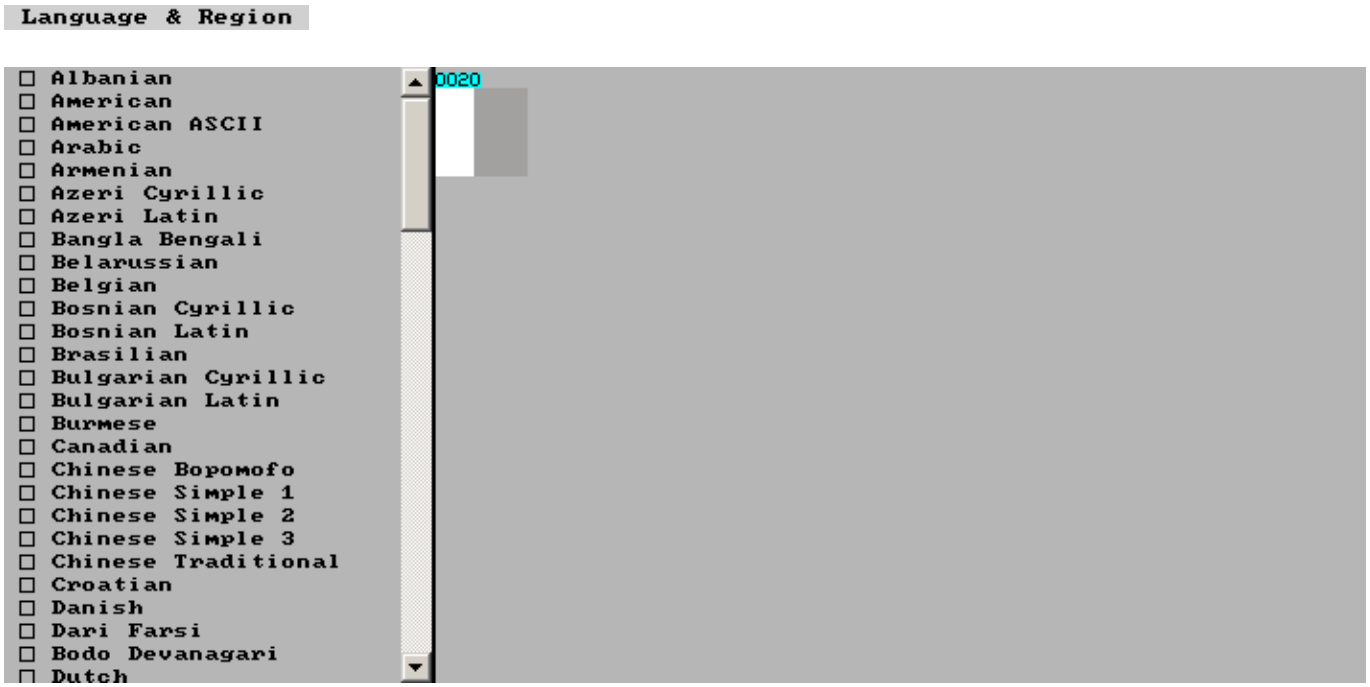
Press OK.

Change to ***Font Edit***:

The default character is automatically chosen as the first character in the font, in this case SPACE.

Change to ***Language & Region***

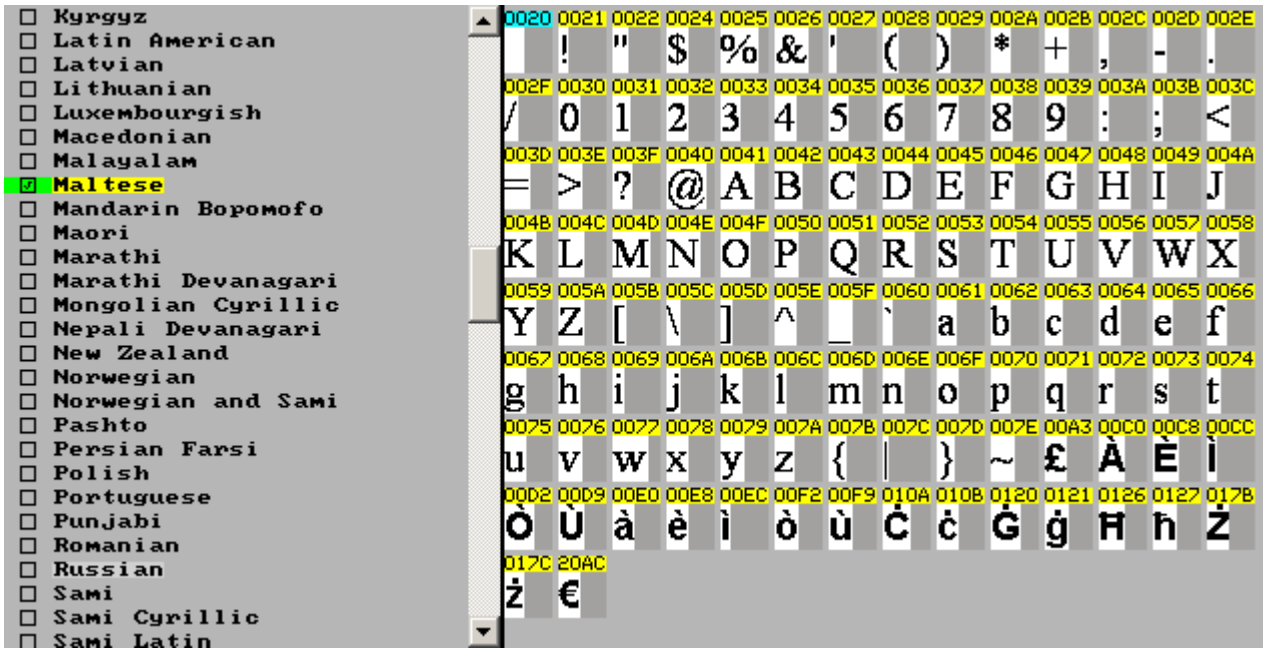
Warning: Language & Regions and Unicode Scripts & Symbols only works with Master Fonts according to the Unicode standard as described at www.unicode.org. See the chapter **How to Check that an Existing Font is According to Unicode.**



Upon entry to Language & Region edit mode, the present font is already marked to avoid accidental erasure of existing symbols in this edit mode:

Warning: If this is un-ticked and no other filter is chosen by a tick, all characters except the default character are marked for delete.

Press the tick box for a language, for example Maltese:

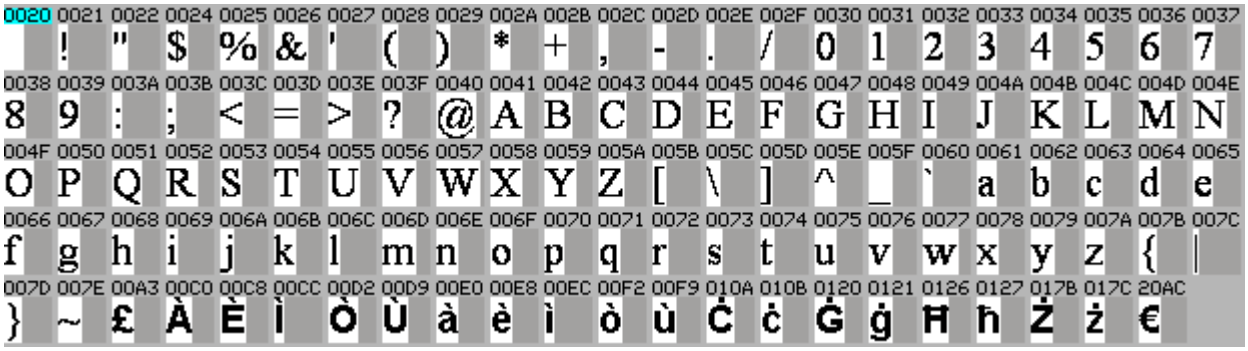


The default character is marked by highlighting the number in cyan. The rest of the characters in the language are marked by highlighting the number in yellow.

The characters 0x0021 to 0x007E are copied from the chosen Master Font. The rest are not present in LatinCyrillic, but are added from the default Windows font for orientation.

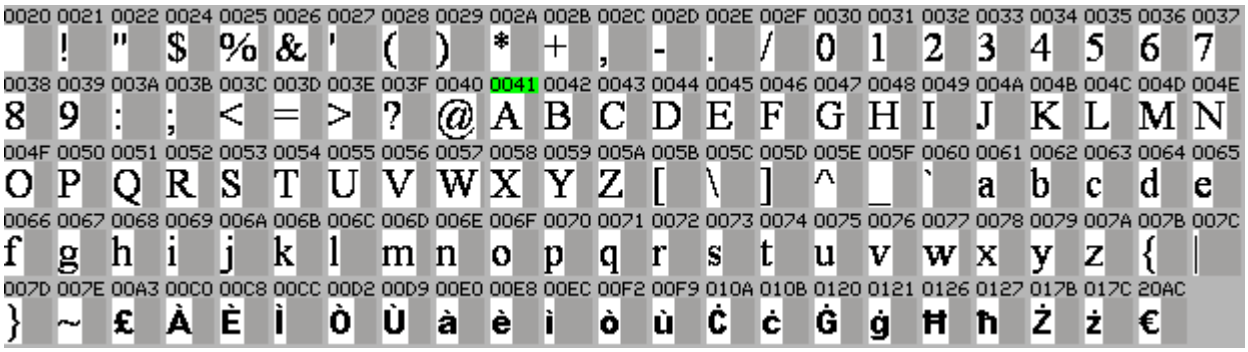
1: How to Fit New Characters to an Existing Font

The added characters may not look like the Master Font. To correct this, change to ***Font Edit***:



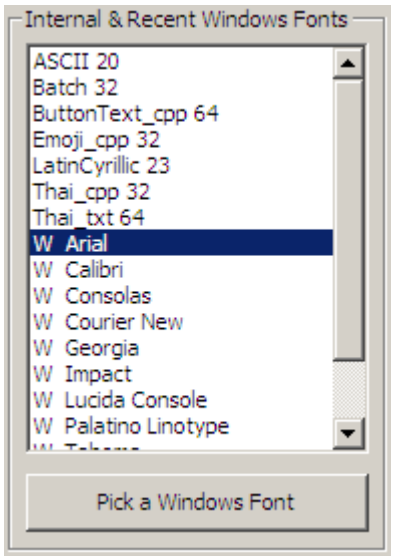
Windows comes with a large selection of fonts, and it is usually possible to find one that fits an existing font reasonably well.

Choose a typical character for reference:

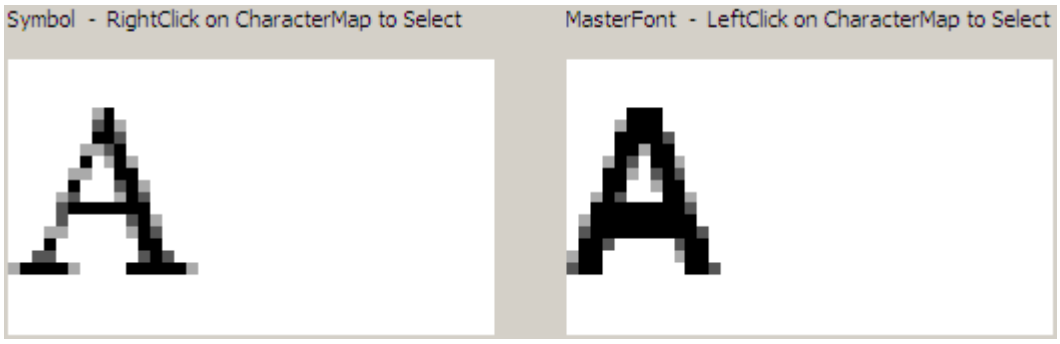


Open the ***Compare Symbols with MasterFont***:

Change the Master Font to a font indicated with W for Windows, which is always Unicode:

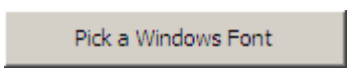


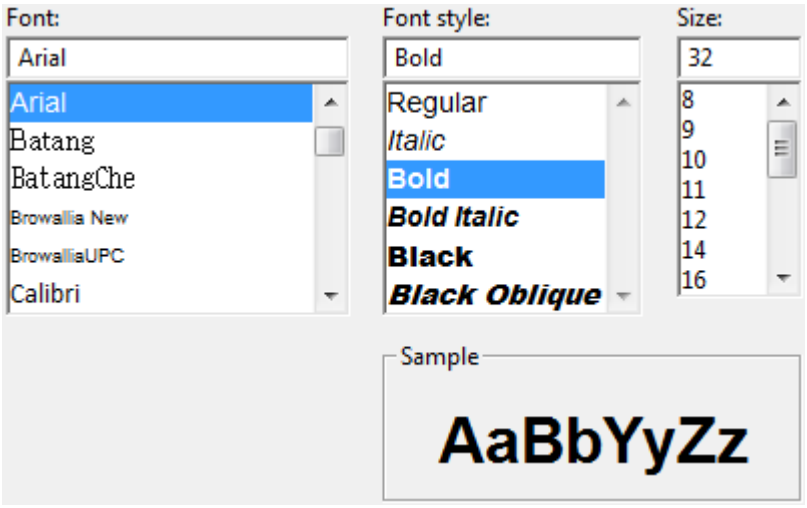
The compare windows show the difference between the symbol A in the font and the A from the Windows font:



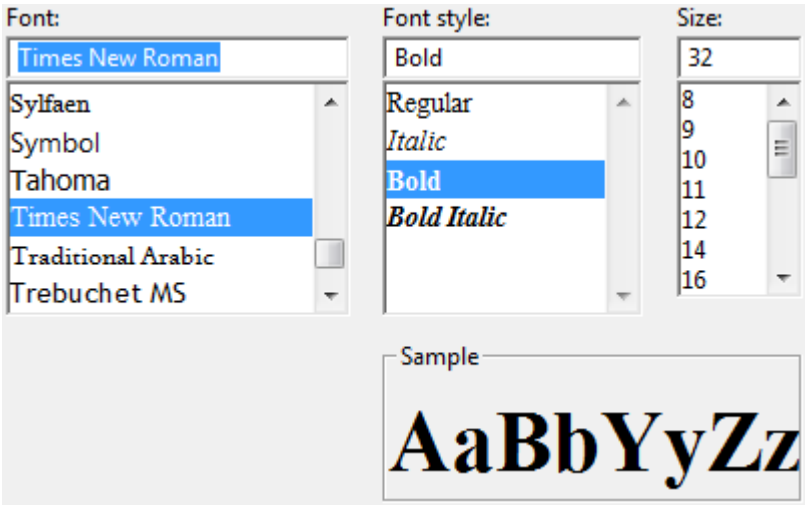
If the recent fonts are not ideal, pick another Windows font as this example shows.

Open the Windows font selector to find a better match:



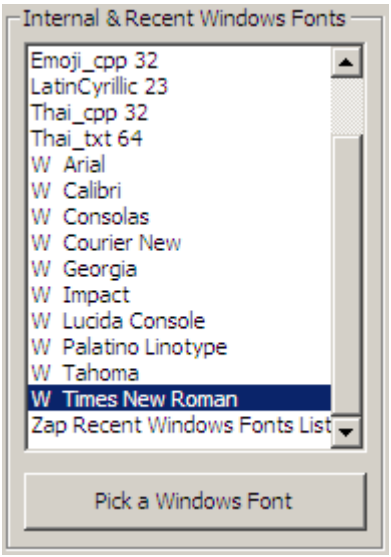


Experiment with different fonts:

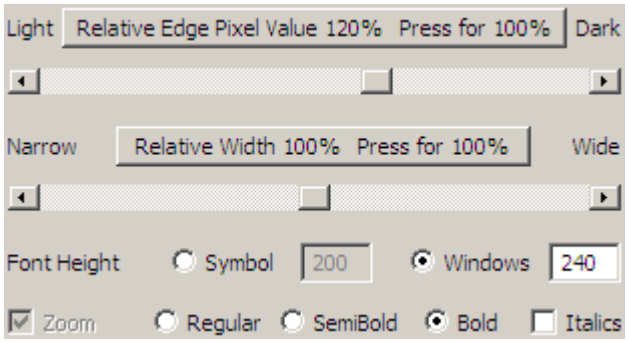


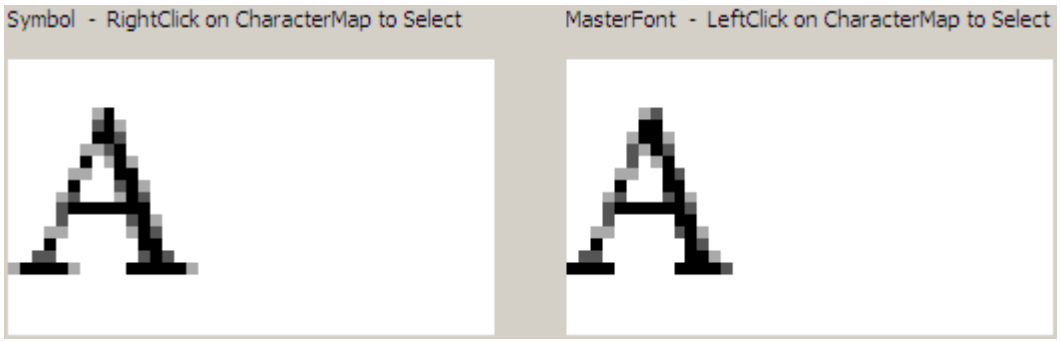
The LatinCyrillic looks similar to Times New Roman, so press **OK**.

This is now selected as master font:



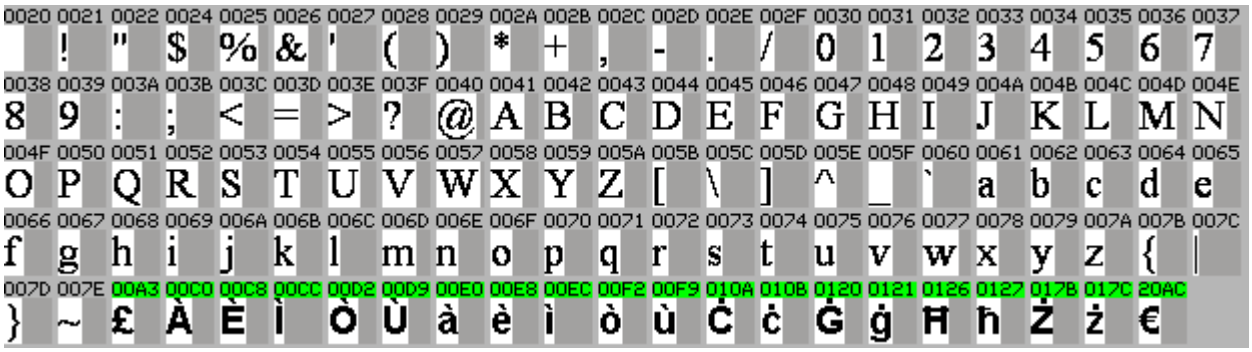
Select the nearest fit:



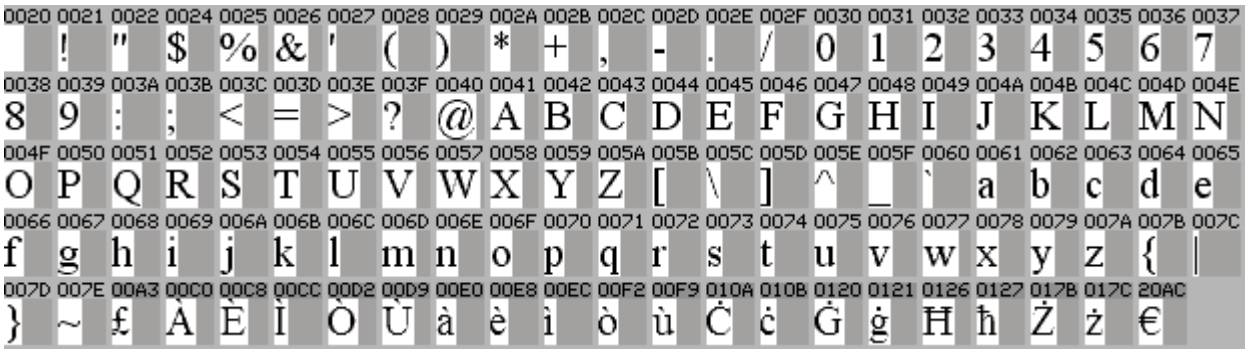


Press **OK**.

Mark characters from 0x00A3 to 0x20AC:



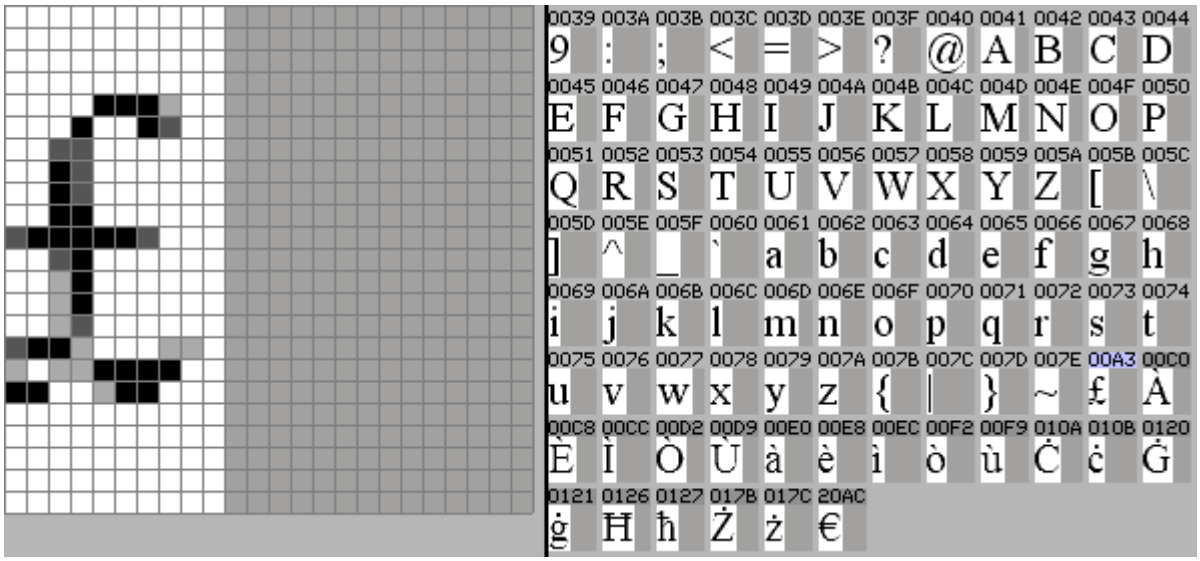
Use the ***Redraw Characters*** button to get new symbols from the new master font:



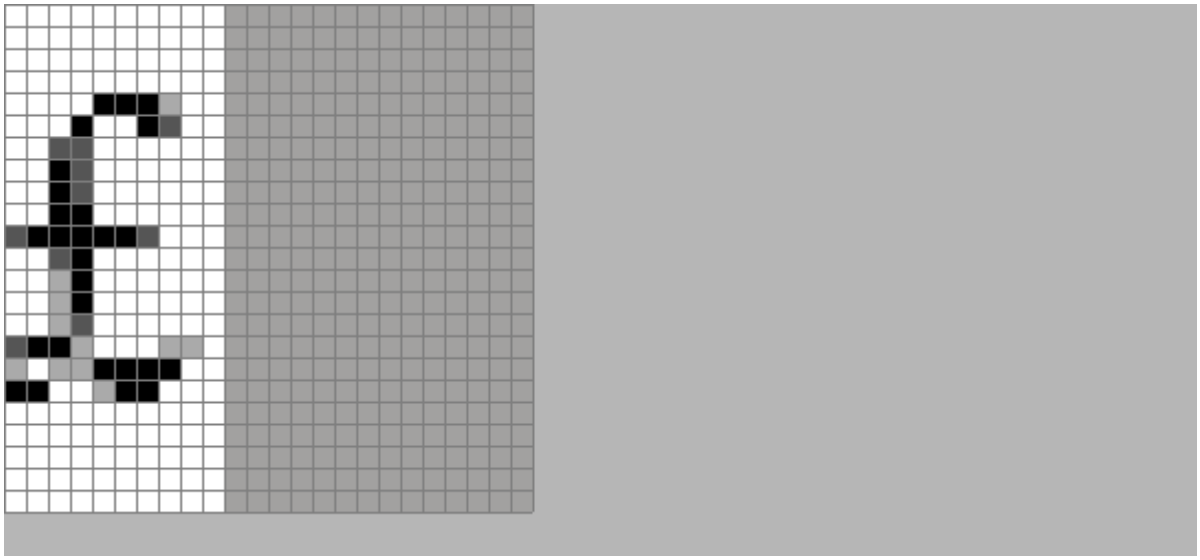
The new characters are marked in dark grey.

Any subsequent corrections should be done at pixel level in the character edit mode.

Right click a character to enter pixel edit mode for each new character:



The display of the font can be turned on and off with the ***Show Font or Group for Choice of Symbols*** button:



When all necessary corrections are done, the extracted language font is ready to save:

Press the ***Save All As…*** button in the Main tool bar to save the pixel data in the *Maltese.c* file in the proper directory. The Sym and Code Point Character List files are saved automatically.

E: How to Combine Two or More existing Fonts

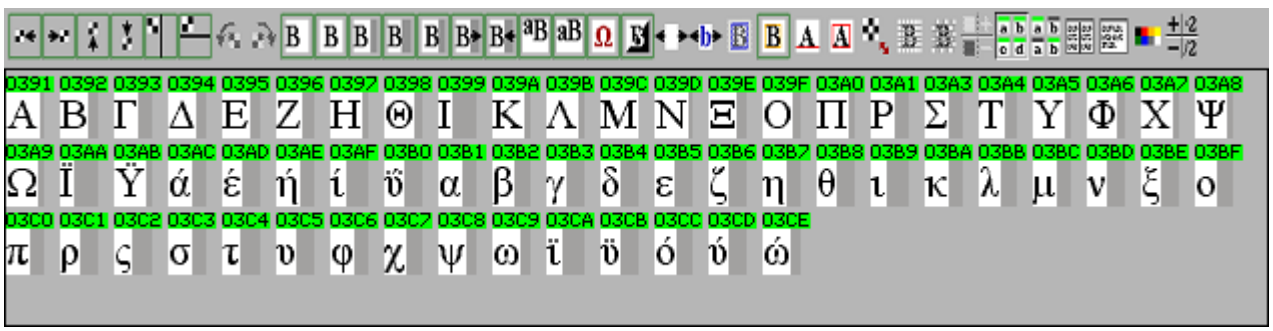
The Code Point Character List system is for adding or removing characters and keeping track of the used characters.

Press the ***File Open*** button in the Main tool bar:

Open two existing fonts, for example *LatinCyrillic.c* and *Greek.c*.

Press ***Split Main Window to Show 2 Datasets***:

Mark the whole font in *Greek.c* by pressing ***Mark All Symbols***:



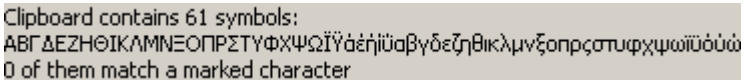
Use ***Copy to ClipBoard***:

Remove the split of the main window to show only one font:

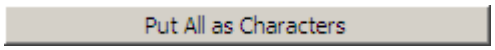
Select ***LatinCyrillic***.

Press ***Paste from ClipBoard*** :

to get this message:



The two fonts are the same height, so select:



The Greek font is fitted between Latin and Cyrillic:

Press the ***Save All As…*** button in the Main tool bar to save the pixel data in the *LatinGreekCyrillic.c* file in the proper directory. The Sym and Code Point Character List files are saved automatically.

F: How to Convert 8 bit Classic Fonts and Texts to 16 bit Unicode

Classic 8-bit fonts do not have a Code Point Character List system for removing unused characters and keeping track of used characters in a font. The character symbols are always numbered from 0 and up to 255 (0000 to 00FF). When the font is opened, a dummy Code Point Character List is automatically created to facilitate editing.

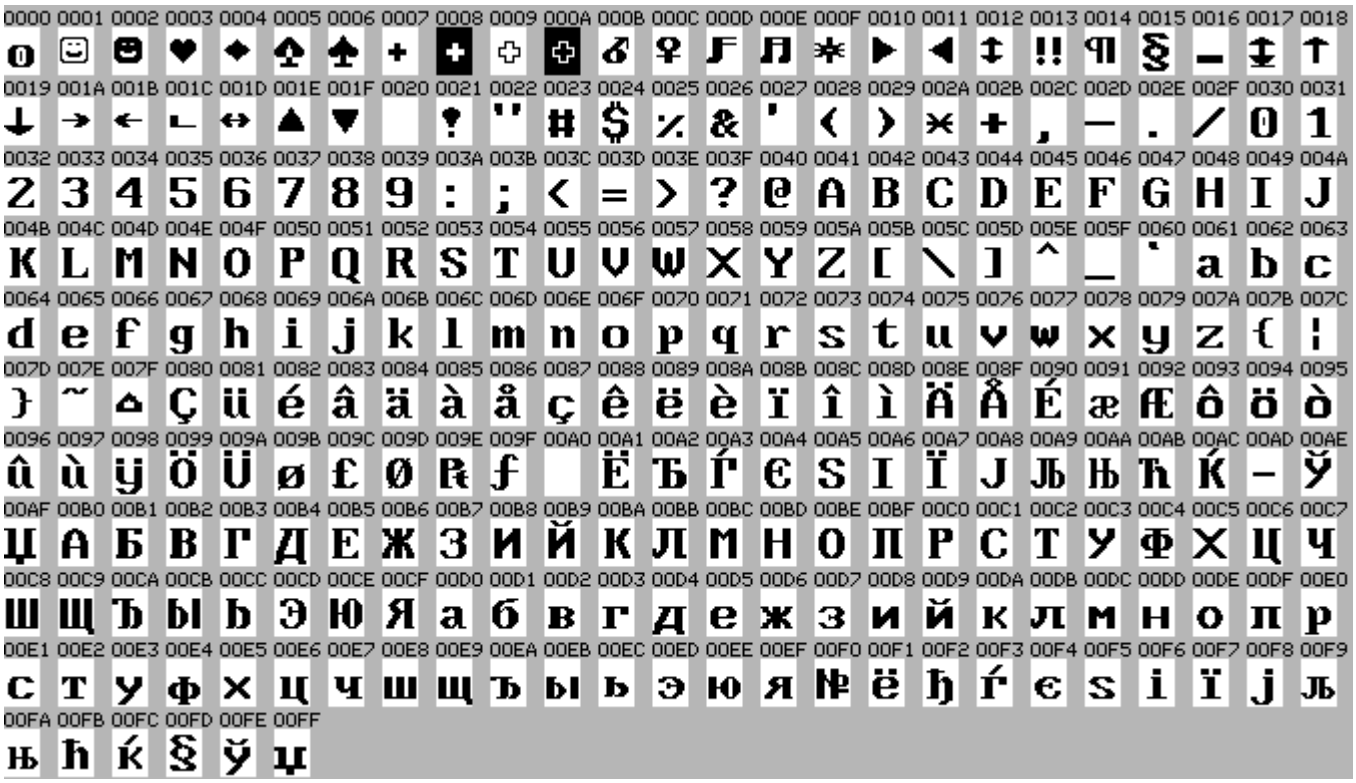
Classic fonts only cover a few languages whereas Unicode cover almost all living languages in the world. This means that once an application that uses classic fonts is converted to Unicode it is relatively simple to extend it to many different languages.

In this chapter we will convert a classic font and a classic text together to Unicode.

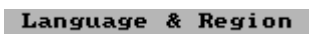
1: Convert Font from a Classic 8 bit Font Encoding to Unicode:

Press the ***File Open*** button in the Main tool bar:

Open the existing ISO8859-5 font *Cy18x24.c*:

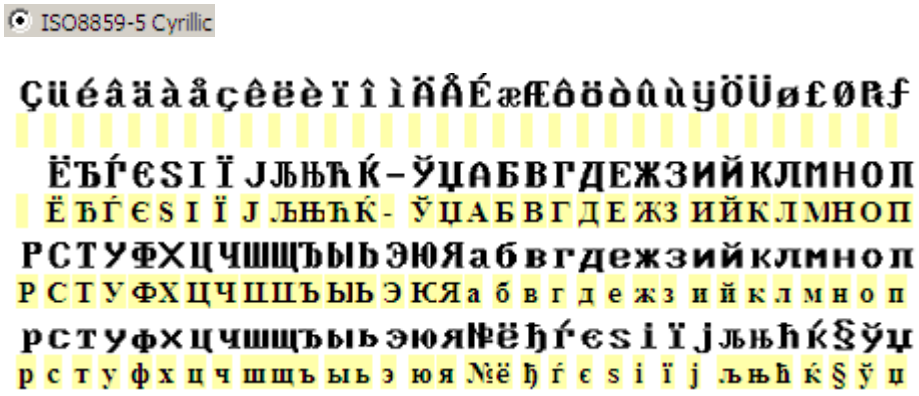


Change to ***Language & Region*** edit mode:



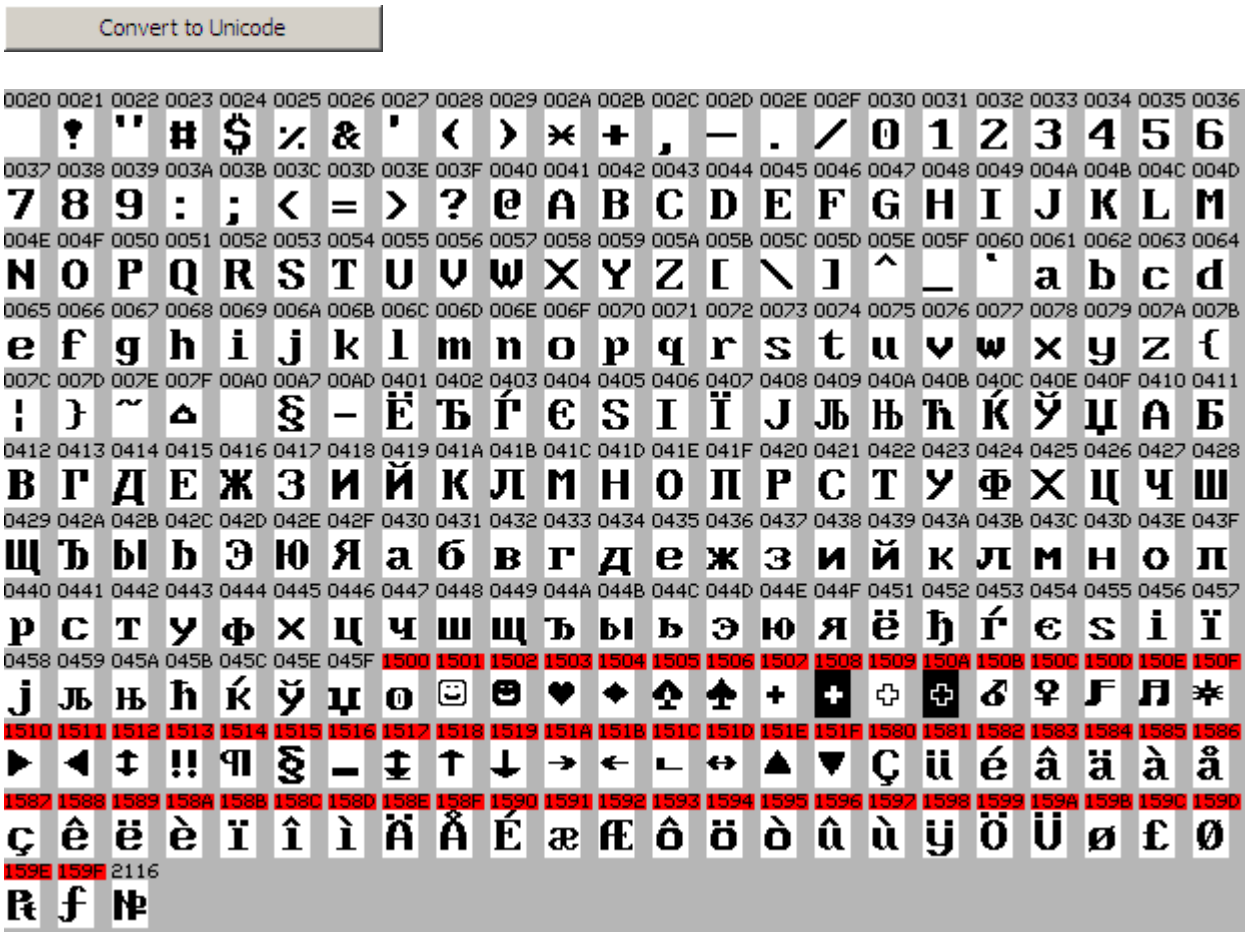
Start the ***Convert 8 bit Font*** dialog box:

Choose ISO8859-5 Cyrillic:

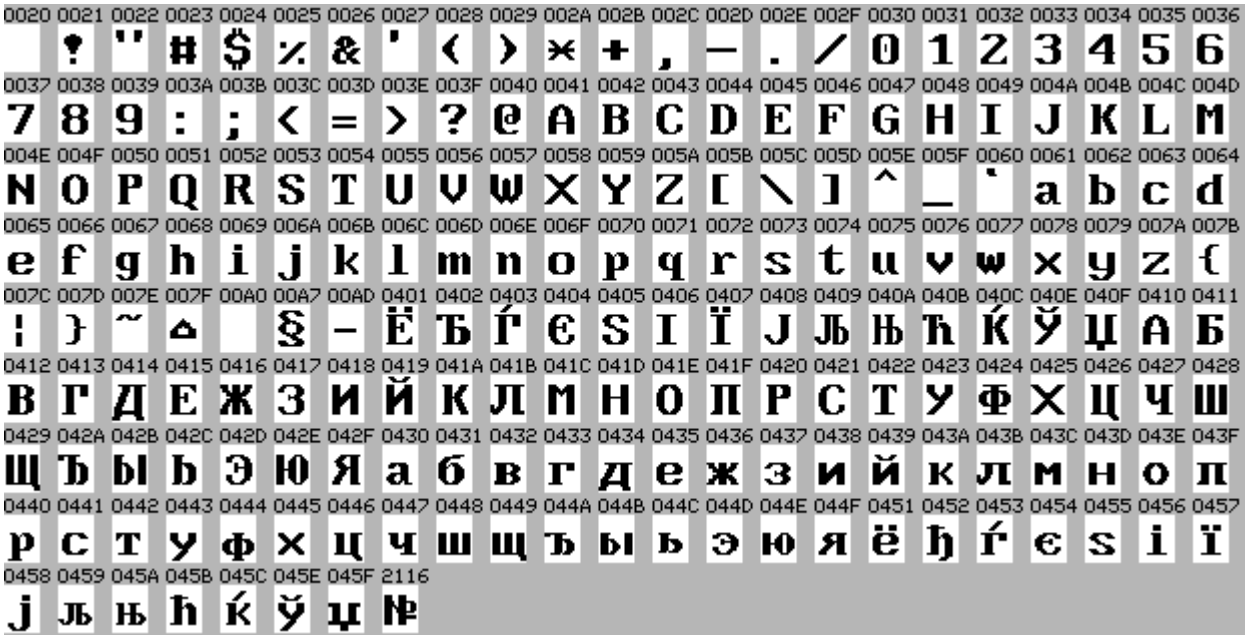


The white rows are the existing characters in the ISO font. The yellow rows are the Unicode characters the symbols will represent after conversion. The top pair of rows is not defined in the ISO standard, and will be moved to an unused area in Unicode and marked for delete during the conversion.

Convert:



The unused characters can be deleted with the local ***Delete Unused Characters*** function:



Your Font is ready for saving....

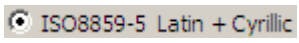
Press the ***Save All As…*** button in the Main tool bar to save the data under a new name in the proper directory. The Sym and Code Point Character List files are saved automatically.

2: Convert a Text with Classic 8 bit Font Encoding to Unicode:

Open an exiting Unicode text *Astronomy_8.c* as marks with ***Text -> Import Text or Text Catalogue to Mark or Create Characters***.

This opens the text converter dialog:

Choose

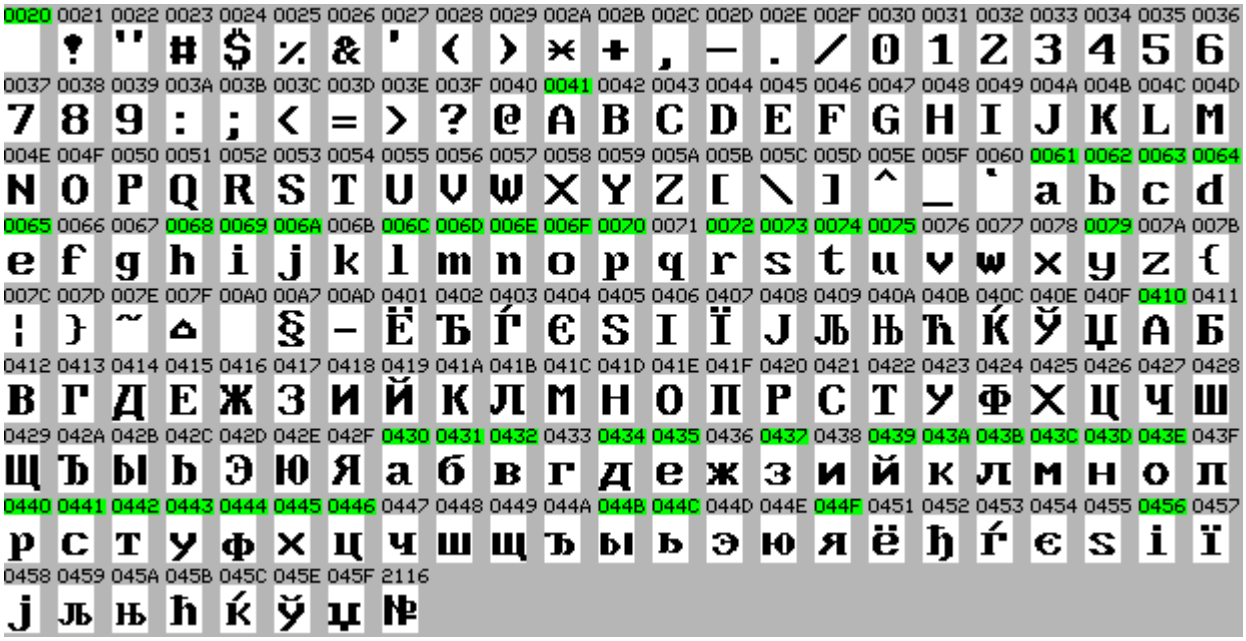


To see the correct text:

```
// Disclaimer: These texts are only for demonstration of the principle
// and may not make any sense to someone familiar with the language
#ifdef ENGLISH
char szA_00[]={"Astronomy is a natural science that"};
char szA_01[]={"studies celestial objects and phenomena"};
#elif BELARUSSIAN
char szA_00[]={"Астраномія  фундаментальная навука"};
char szA_01[]={"якая займаецца даследваннем нябесных"};
#endif
```

Press **OK**.

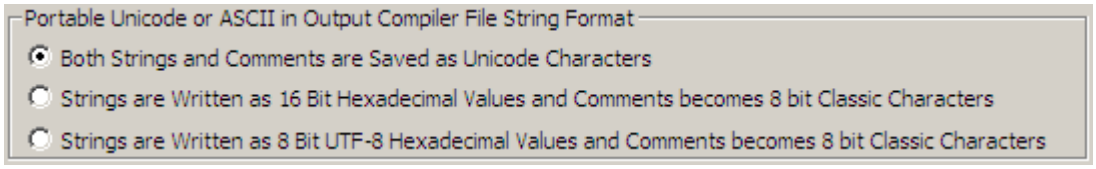
The characters used by the Unicode text are highlighted in green:



The text is automatically shown with the font:

```
#ifdef ENGLISH
char szA_00[]={"Astronomy is a natural science that"};
char szA_01[]={"studies celestial objects and phenomena"};
#elif BELARUSSIAN
char szA_00[]={"Астраномія  фундаментальная навука"};
char szA_01[]={"якая займаецца даследваннем нябесных"};
#endif
```

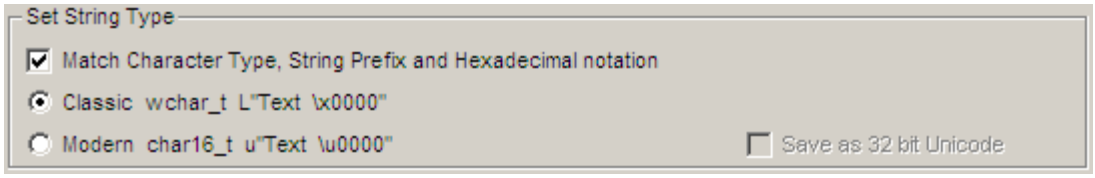
Click ***Show Exported Text*** to set options:



The text is now Unicode and **char** have to be changed to **wchar_t** by your editor or by the Set String Type before the text can be compiled:

```
#ifdef ENGLISH
char szA_00[]={"Astronomy is a natural science that"};
char szA_01[]={"studies celestial objects and phenomena"};
#elif BELARUSSIAN
char szA_00[]={"Астраномія  фундаментальная навука"};
char szA_01[]={"якая займаецца даследваннем нябесных"};
#endif
```

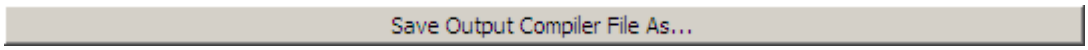
Option for updating the code:



If your editor or compiler cannot handle Unicode texts choose UTF-8 instead:

```
#ifdef ENGLISH
char szA_00[]={"Astronomy is a natural science that"};
char szA_01[]={"studies celestial objects and phenomena"};
#elif BELARUSSIAN
char szA_00[]={"\xD0\x90\xD1\x81\xD1\x82\xD1\x80\xD0\xB0\xD0\xBD\xD0\xBE\xD0\xBC\xD1\x9
char szA_01[]={"\xD1\x8F\xD0\xBA\xD0\xB0\xD1\x8F \xD0\xB7\xD0\xB0\xD0\xB9\xD0\xBC\xD0\x
#endif
```

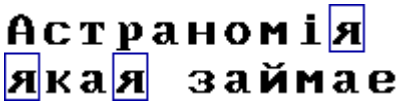
To save the text as *Astronomy_8__c.c* press:



Press **OK**.

Use blue highlighting to identify the characters:

Press **OK**, and click any character:




```
\xD0\x90\xD1\x81\xD1\x82\xD1\x80\xD0\xB0\xD0\xBD\xD0\xBE\xD0\xBC\xD1\x96\xD1\x8F \x
\xD1\x8F\xD0\xBA\xD0\xB0\xD1\x8F \xD0\xB7\xD0\xB0\xD0\xB9\xD0\xBC\xD0\xB0\xD0\xB5\x
```

G: How to Convert 16 bit Unicode Font or Text to 8 bit Classic Encoding

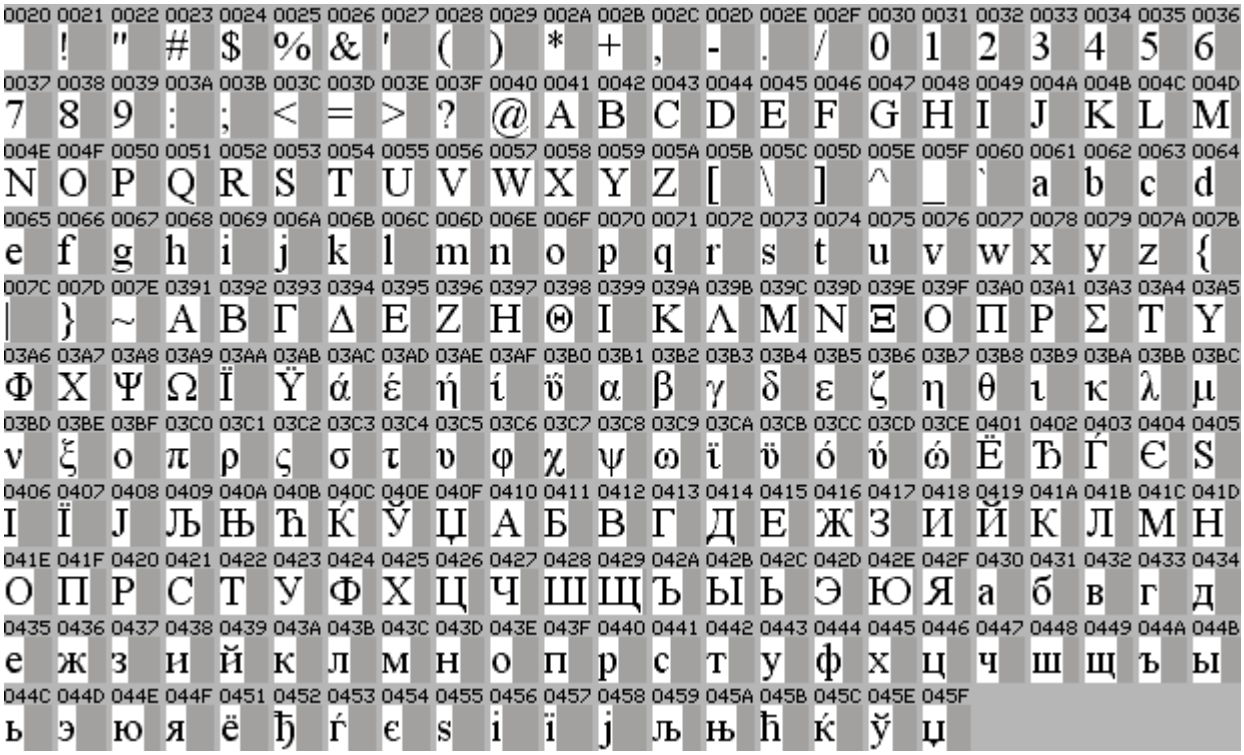
Unicode with 16 bit Code Points define about 56000 characters for almost all living languages in the world, whereas 8 bit classic fonts such as DOS, Windows and ISO8859 have 128 ASCII Latin and control characters followed by 128 language or region specific characters.

In many cases it would be convenient to be able to convert a 16 bit Unicode font and text to fit an 8 bit Classic font. In this chapter we will do both.

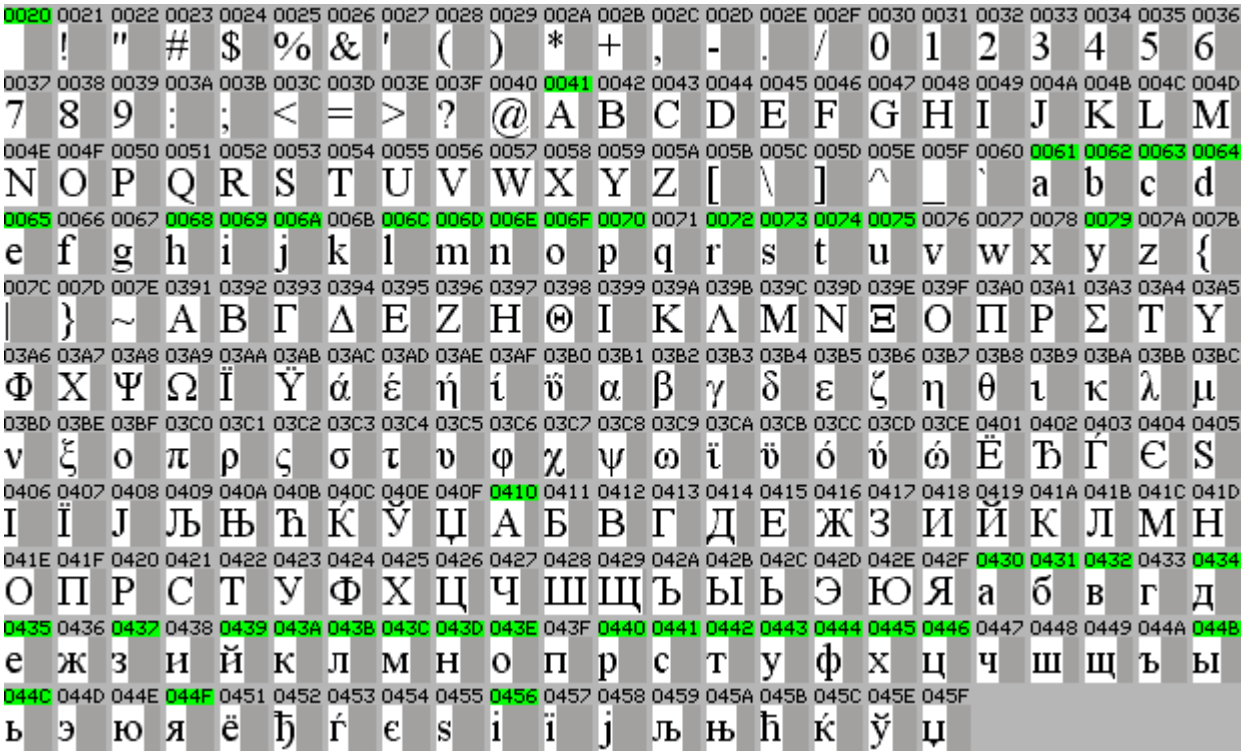
1: Convert a Unicode Font to a Classic 8 bit Font Encoding:

Press the ***File Open*** button in the Main tool bar:

Open an existing Unicode font, for example *LatinGreekCyrillic.c*.



Open an exiting Unicode text *Astronomy_U.c* as marks with ***Text -> Import Text or Text Catalogue to Mark or Create Characters***. The characters used by the text are highlighted in green:

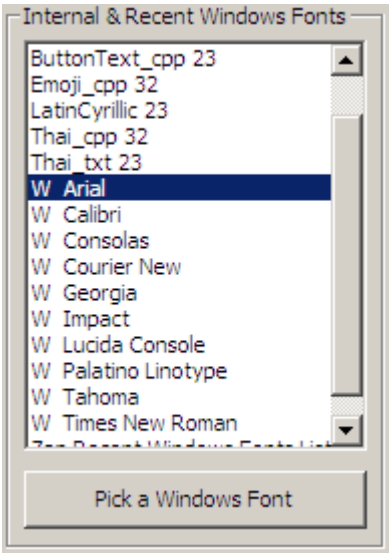


The text is shown automatically with the font:

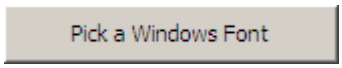
```
#ifdef ENGLISH
wchar_t szA_00[]={L"Astronomy is a natural science that"};
wchar_t szA_01[]={L"studies celestial objects and phenomena"};
#elif BELARUSSIAN
wchar_t szA_00[]={L"Астраномія фундаментальная навука"};
wchar_t szA_01[]={L"якая займаецца даследваннем нябесных"};
#endif
```

During conversion to a classic font, any characters missing in *LatinGreekCyrillic.c* will be taken from the Master Font, so it has to look as much as possible like the original font. Open the compare dialog:

First select a Windows Unicode font:



If the recent fonts are not ideal, pick another Windows font as this example shows:



This opens the Windows font picker, the appearance of which is depends greatly on the Windows version, Windows native language, what kind of foreign language support is installed, and any additional non-Windows fonts added later.

The original font looks a lot like Palatino Linotype or Times New Roman, so fill in the 3 top fields to choose for example Times New Roman, Regular and 24. The last field is the height of the whole character including top and bottom white space. This is the height of the each symbol in pixels.

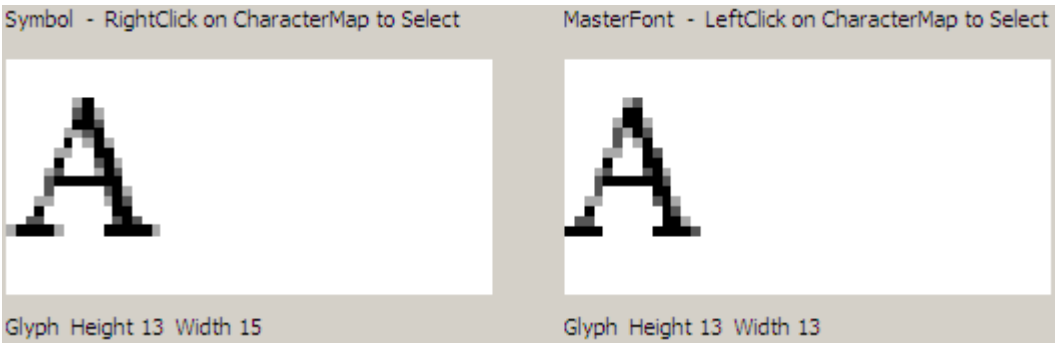
Times New Roman Regular 23

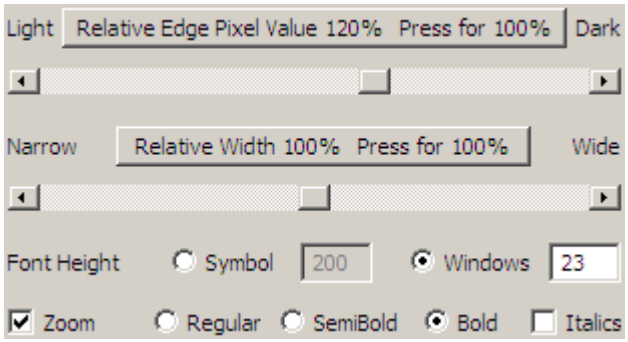
Press OK.

This is basically a grey tone vector font, and the conversion to grey levels has to be fine trimmed.

Adjust the thickness of the character until the connectivity is OK and the characters look as much as possible like the LatinGreekCyrillic.

Characters with diagonal lines like / @ A W are usually the most critical.





Press **OK**.

Check for correct Unicode values in the original font:

2: How to Check for Correct Unicode Values Before Converting

Press ***Compare Symbols: Show Symbols in Actual Size***:

And then again as ***Compare Symbols: Show Symbols and Master Font Together***:

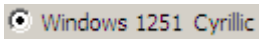
Scroll through the whole font to check that characters are alike:

The top row is the newly opened font and the lower is the chosen Master Font.

Change to ***Language & Region*** edit mode:

Start ***Convert***:

Choose Cyrillic:



The black on white symbols on grey background are the original font, and black on pink symbols on white background are going to be supplied from the Master Font as normal black on white characters.

Press the convert button:

Convert to 8 bit Font

The Font now contains 256 Characters.

There are now 2 options, Classic Font and Code Point Character List Font:

3: Save as a Classic 8 bit Font:

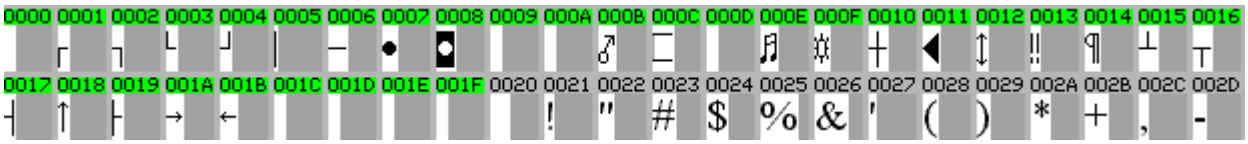
Press the ***Save All As…*** button in the Main tool bar to save the pixel data as *Cy24x23.c* in the proper directory. The Sym file is saved automatically. The corresponding Code Point *Cy24x23.cp* is redundant, and is not saved.

4: Save as an 8 bit Memory Optimized Font:

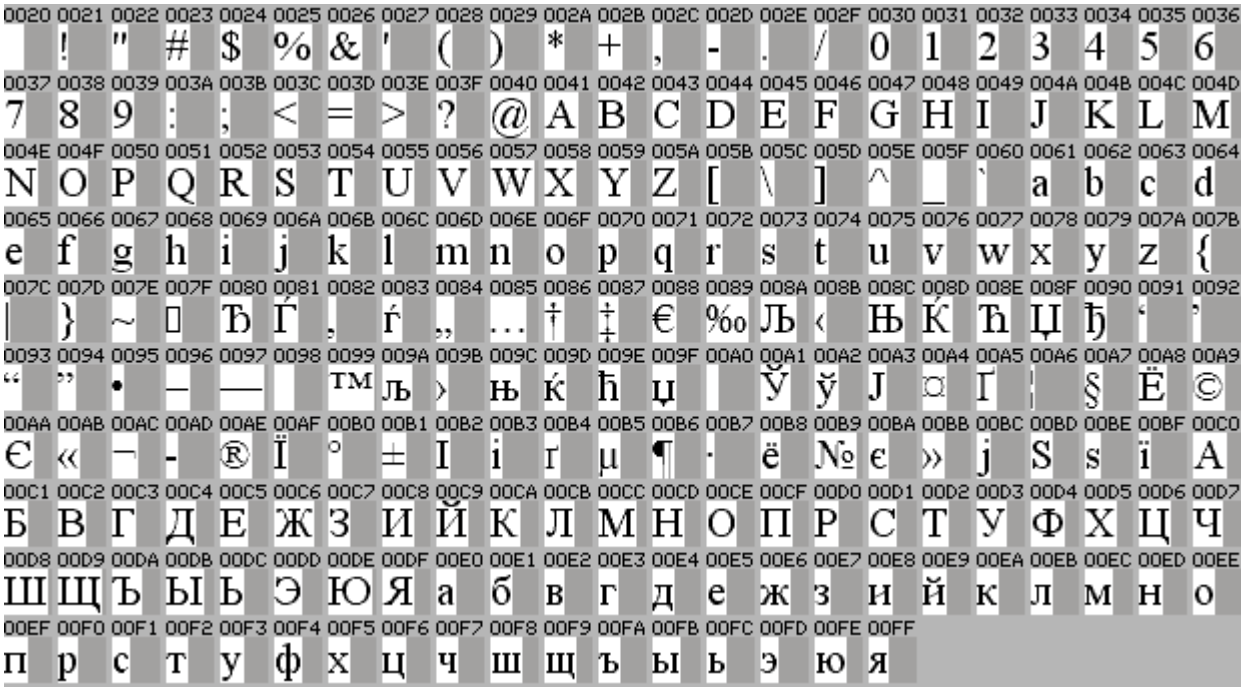
The control characters 0000 to 001F are not displayable, they only take up unnecessary memory space and could be deleted.

Change to **Font Edit** mode:

Mark the unnecessary symbols with the Mouse and Ctrl+Mouse:



Press ***Delete***:



The Font now contains 224 valid Characters.

Press the ***Save All As…*** button in the Main tool bar to save the pixel data as *Cy24x23.c* in the proper directory. The corresponding Sym and Code Point Character List files are saved automatically.

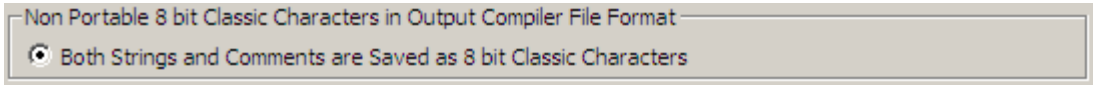
5: Convert a Unicode Text to a Classic 8 bit Font Encoding:

The Text and the Font do not match anymore, as indicated with red highlighting of the Cyrillic text:

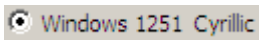
```
#ifdef ENGLISH
wchar_t szA_00[]={L"Astronomy is a natural science that"};
wchar_t szA_01[]={L"studies celestial objects and phenomena"};
#elif BELARUSSIAN
wchar_t szA_00[]={L"Астраномія фундаментальная навука"};
wchar_t szA_01[]={L"якая займаецца даследваннем нябесных"};
#endif
```

Click ***Show Exported Text*** to set options:

Select ***Classic Characters***:



Press ***Output String and Comment Encoding***, and select ***Windows Cyrillic***:



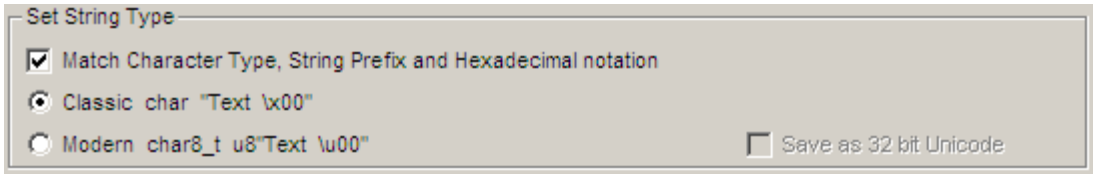
Press **OK**. The classic text shown with the classic font is now correct:

```
#ifdef ENGLISH
wchar_t szA_00[]={L"Astronomy is a natural science that"};
wchar_t szA_01[]={L"studies celestial objects and phenomena"};
#elif BELARUSSIAN
wchar_t szA_00[]={L"Астраномія фундаментальная навука"};
wchar_t szA_01[]={L"якая займаецца даследваннем нябесных"};
#endif
```

But the text is no longer portable and will be shown garbled in most editors:

```
#ifdef ENGLISH
wchar_t szA_00[]={L"Astronomy is a natural science that"};
wchar_t szA_01[]={L"studies celestial objects and phenomena"};
#elif BELARUSSIAN
wchar_t szA_00[]={L"Àñòðàíîì³ÿ  ôóíäàìåíòàëüíàÿ íàâóêà"};
wchar_t szA_01[]={L"ÿêàÿ çàéìàåööà äàñëåäâàííåì íÿáåñíûõ"};
#endif
```

Before the text can be compiled **wchar_t** has to be changed to **char** and **L** should be removed. This could be done manually after the text is saved or automatically with the **Set String Type** option before saving:



Press ***Save Modified Text File as*** to save the classic text as *Astronomy_U__c.c*.

H: How to Convert a Group of Symbols to Characters in a Font

The Code Point Character List system is for removing unused characters and keeping track of used characters in a font. Groups do not have a Code Point Character List. The symbols in a Group are always numbered from 0 and up to N-1. If a group of symbols is to be used as Characters in a Font, the symbols should be moved to their proper character position.

Press the ***File Open*** button in the Main tool bar:

Open the existing group *ABC123.c*.

To change to a font press the ***Modify Symbol Group*** button in the Main Toolbar:

This opens the modify dialog box:

Choose how symbols should be organized:

Press OK.

The symbols are now characters 0000 to 0006.

Character symbols can only be moved to vacant Code Points, so to avoid conflict, start with the symbol that should have the highest Code Point.

If the symbols are so intermixed that starting from the top is impractical, a large number of symbols can be parked temporarily in the Unicode user area from E000 to F8FF by Cut to Clipboard and Paste: Put in Unicode Private Use Area.

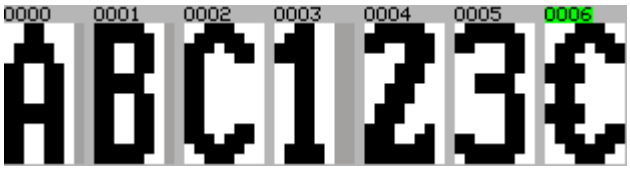
Please note that the Unicode Private Use Area is also used by the Non Unicode standard Hong Kong Supplementary Character Set for the Chinese language.

1: General ClipBoard Method

This method can be used for moving characters and symbols inside or between fonts and groups.

Start from the top:

Mark the Euro sign **€**:

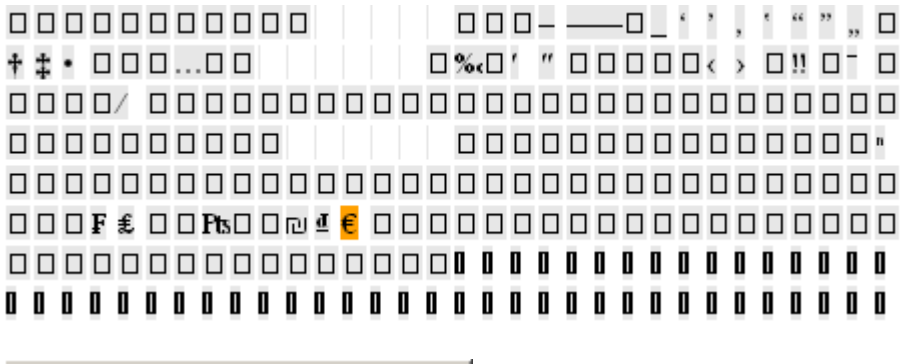


Cut to Clipboard:

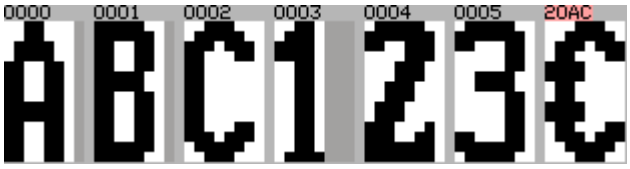
Paste to a new position:

This opens the Paste to Font dialog box:

The Unicode value for the Euro sign **€** is 20AC, mark it with the mouse:



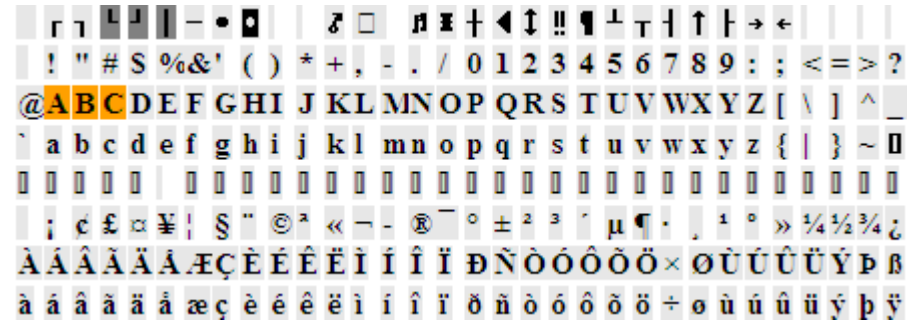
The Euro sign **€** is marked in pink to indicate that it has been recently pasted:



Repeat with ABC and 123:

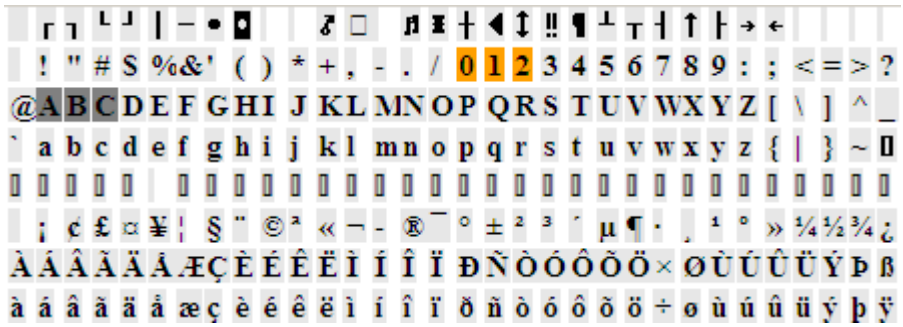
Mark ABC and ***Cut***:

Paste at unused values:



Mark 123 and ***Cut***:

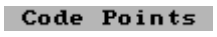
Paste at unused values. Characters ABC are marked in grey to show that these values are already occupied:



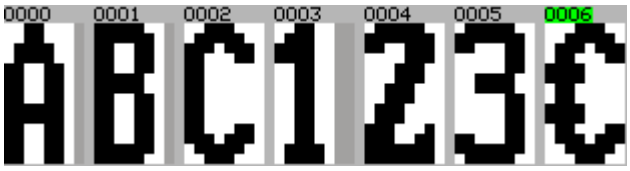
2: Fast Code Point Edit Method

This method can be used for moving characters inside a font.

Change to ***Code Point Edit***:

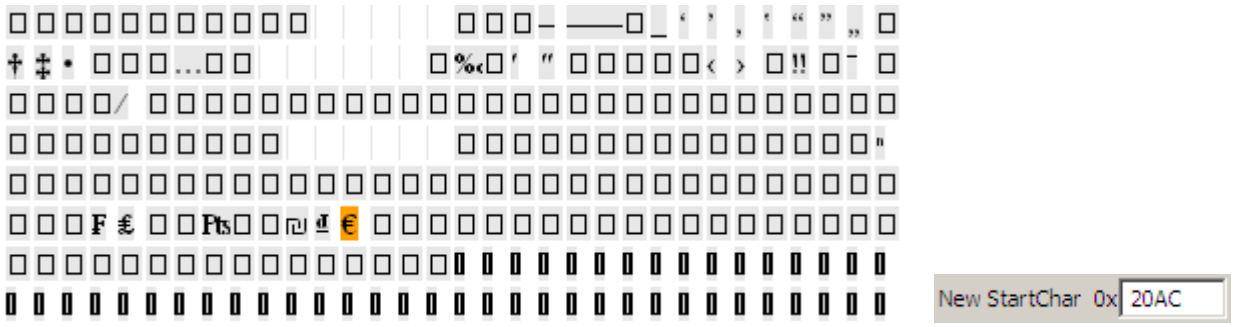


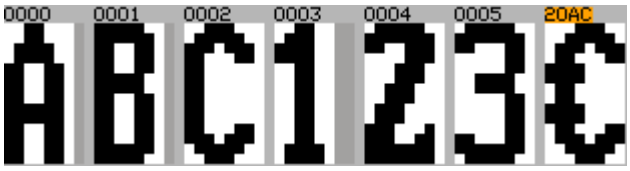
Mark the Euro sign **€**:



Press ***Move Selected Symbols***:

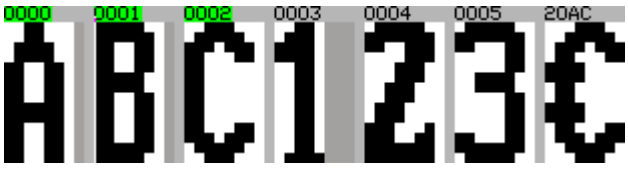
The Unicode value for the Euro sign € is 20AC, mark it with the mouse or just write the hexadecimal code:

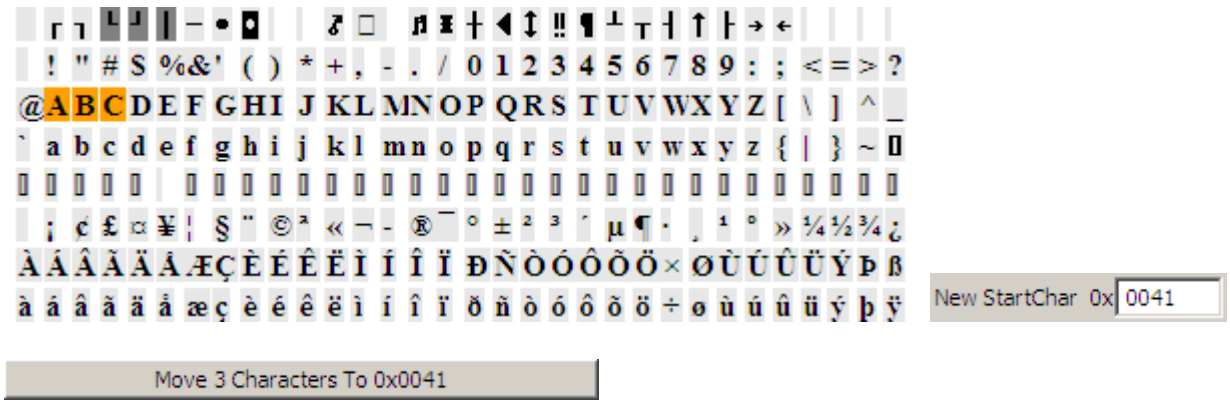




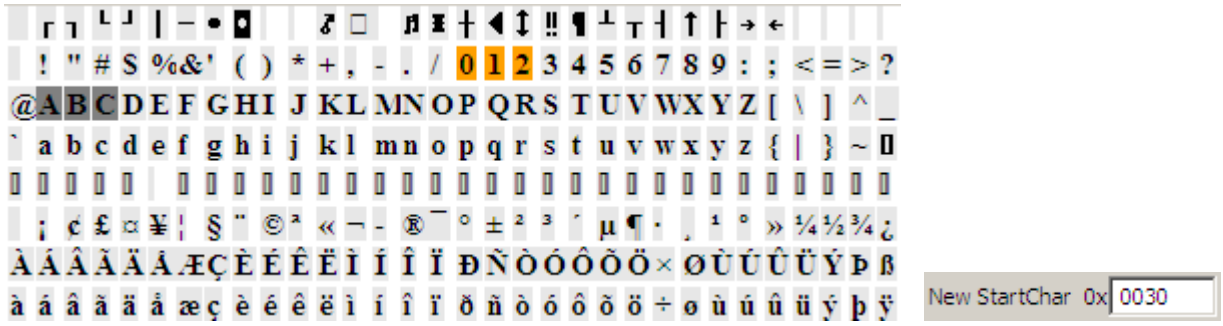
Repeat with ABC and 123:

Mark ABC and press ***Move Selected Symbols***:





Mark 123 and press ***Move Selected Symbols***:



3: Check for correct Unicode values after the conversion

Press ***Compare Symbols: Show Symbols in Actual Size***:

And then again as ***Compare Symbols: Show Symbols and Master Font Together***:

The match is apparent:

Your Font is ready for saving....

Press the ***Save All As…*** button in the Main tool bar to save the data in the proper directory.

4: How to Save Characters or Symbols as a BitMap

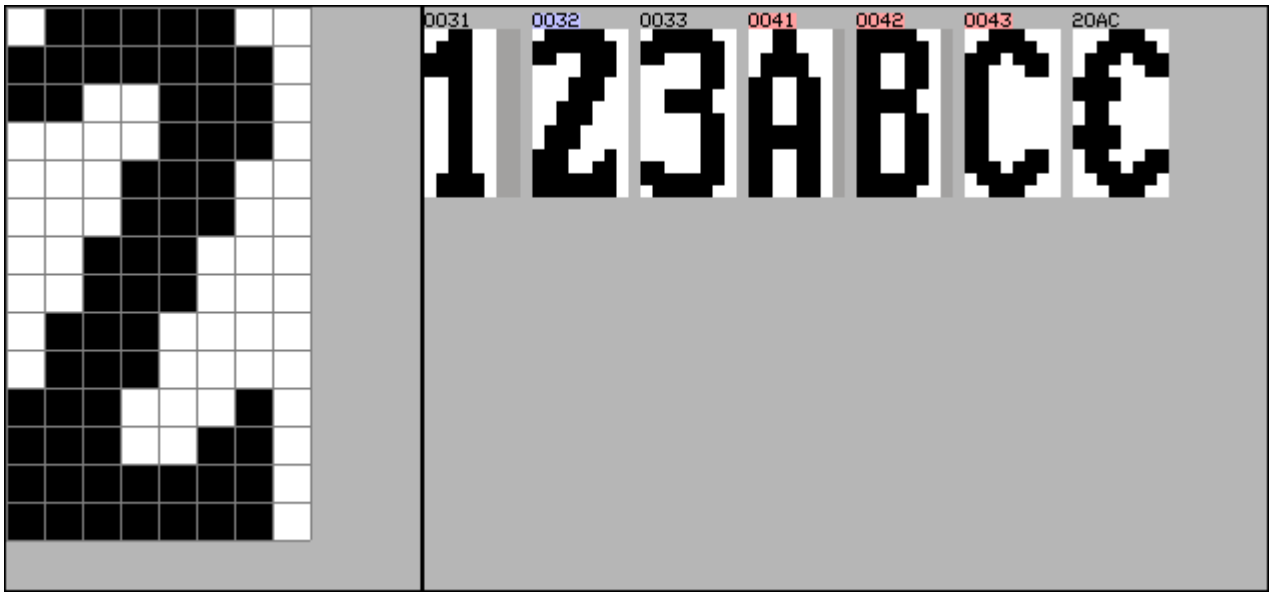
This is a continuation of the previous example.

Any character, symbol, font or group can be saved as a Windows bitmap file. IconEdit can save data as bitmaps in 2 different ways:

1. The character or symbol in the character or symbol edit window can be saved with the zoom factor 1:1.

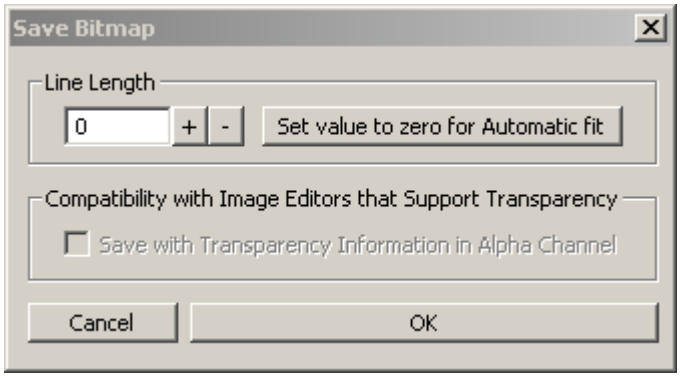
2. The whole data set can be saved as one image in the size it is designed i.e. the saved pixel size and zoom factor is 1:1.

To save bitmaps from the small font in the previous example assume the edit windows look like this:



Press ***Image -> Save Character or Symbol Edit Window as BMP File…*** in the menu to save the bitmap file in the proper directory. The bitmap file can be opened by any image editor that support bitmaps, and will look like this in Paint:

Press ***Image -> Save the Whole Font or Group as One BMP File…*** in the menu to get the Save Bitmap dialog box:



Set Line Length to Automatic fit and save the bitmap file in the proper directory. The transparency information for 32 Bit ARGB color mode can be saved for use by editors that support transparency, for all other color modes this option is irrelevant. The bitmap file can be opened by any image editor that support bitmaps, and will look like this in Paint:

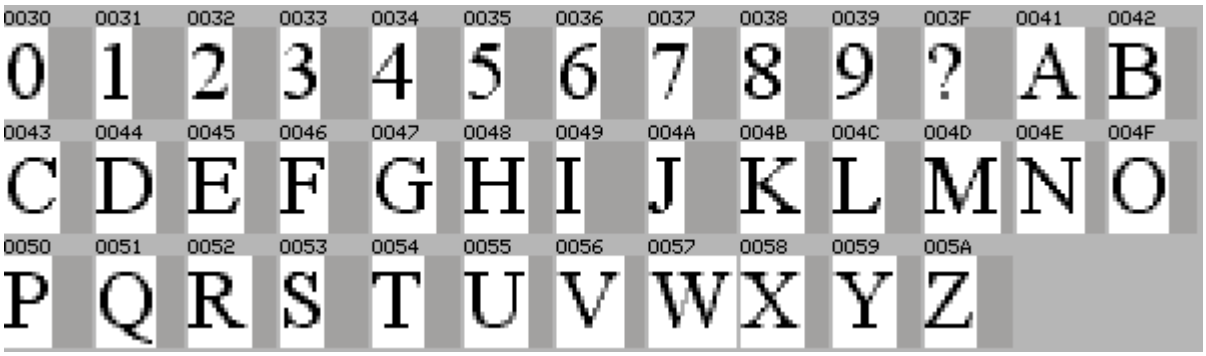
Please note that IconEdit supports a much larger number of color formats than the Windows Bitmap format, so when the Bitmap is read back into IconEdit the symbols now have the nearest equivalent to a Windows Bitmap format, but no symbol information is lost in the process.

I: How to Include Private Symbols in a Unicode Font

The Unicode has a large range of undefined Code Points that is meant for private use. The values stretch from 0xE000 to 0xF8FF.

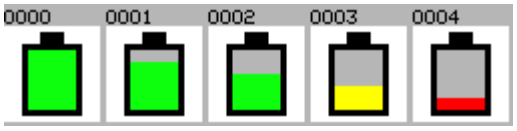
Warning: The *HongKong Supplementary Character Set* and part of the *Simplified Chinese Character Set* is placed in the middle of the private area from 0xE600 to 0xE8FF.

To add private symbols to an existing font such as AZ09:



Press the ***File Open*** button in the Main tool bar:

Open the existing file *battery.c*.



Mark them all with **Ctrl A** and copy to the clipboard with **Ctrl C.**

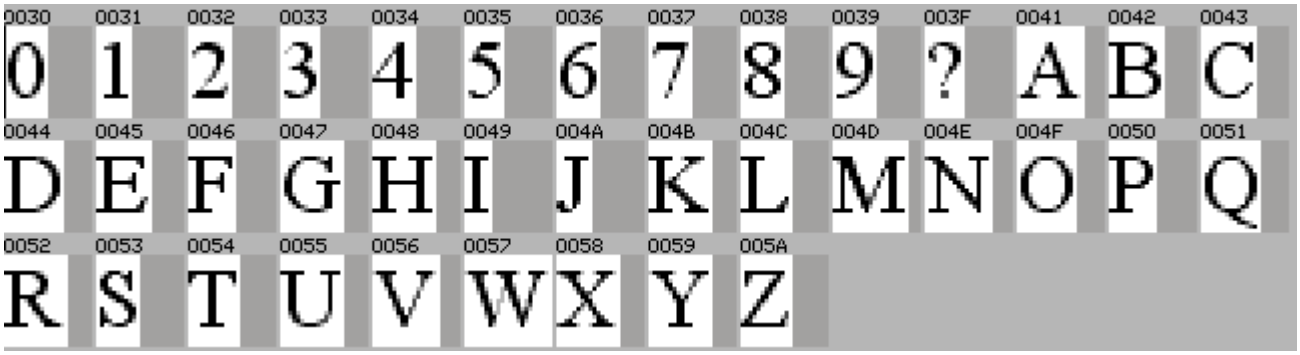
The battery indicators are in color, but the font is in greytone, so the font color format for AZ09 must be changed.

Press the color format button:

This opens the modify dialog box:

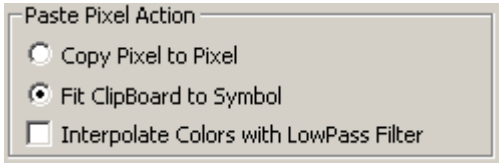
Choose a new color format, here 4 Bit Color Palette to match that of the battery indicators. This option is only available in the color version of IconEdit:

Press OK.



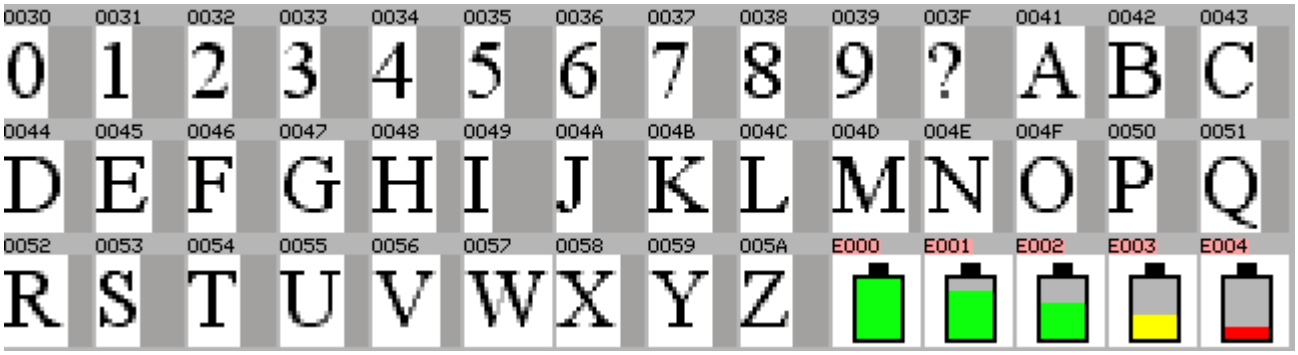
Paste the battery indicators with Ctrl V, this opens the paste dialog box:

Choose resizing:



And where to put the new characters:

The font now looks like this:



Your Font is ready for saving....

Press the ***Save All As…*** button in the Main tool bar to save the pixel data in the *AZ09_Battery.c* file in the proper directory. The Sym and Code Point Character List files are saved automatically.

J: How to Upscale, Downscale or Reshape a Font or Group

The ***Resize*** tool can be used to stretch or squeeze characters and symbols in the x and y direction separately.

The resizing can be done **Pixel to Pixel** this does not change the shape and size of the glyph and is meant for removing unused space in fonts.

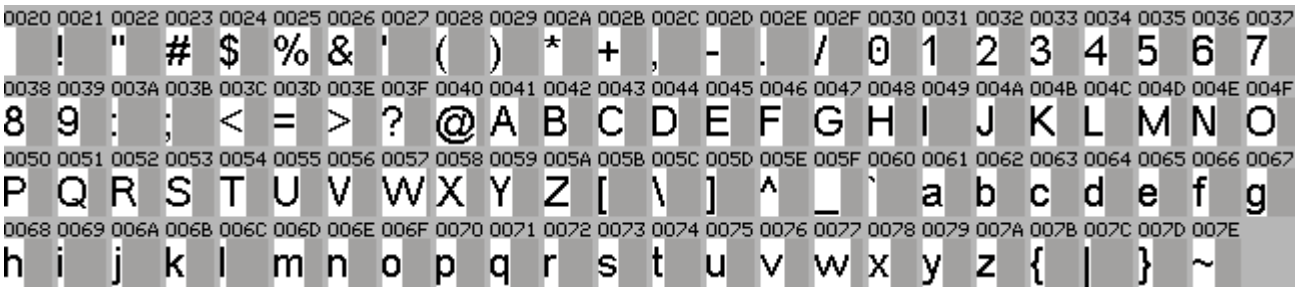
The resizing can be done **Pixel to Pixel but Squeezing Oversize Characters** (instead of just truncating the right hand side of the glyph) this does not change the shape and size of the glyphs that fit inside the new size but squeezes glyphs that would otherwise be truncated usually **M** and **W**. This is meant for reducing the memory consumption of fonts without changing them significally.

The resizing can be done by **Stretch or Squeeze**. This is primarily intended for fitting various images to a desired size but can also be used for upscaling or downscaling of fonts. The **LowPass Filter** can be used to smooth the images by producing intermediate colors.

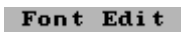
To reshape all the characters in an existing font such as Aa_24x24:

Press the ***File Open*** button in the Main tool bar:

Open the existing file *Aa_24x24.c*.

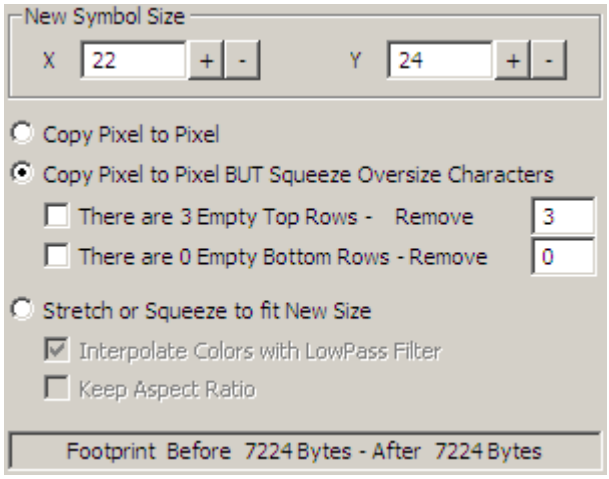


Change to Font Edit if it is not already there:

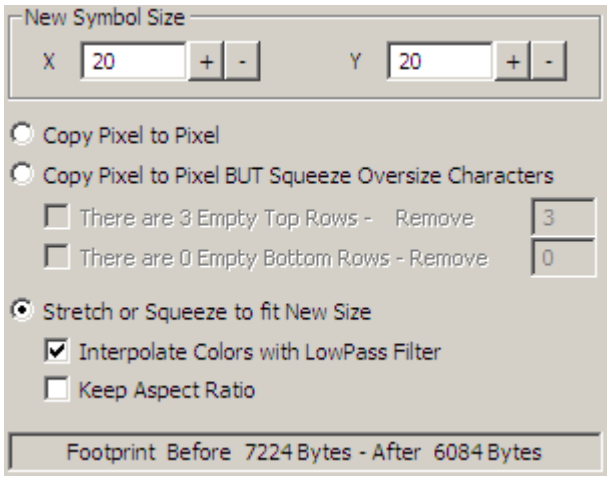


Press the ***Resize*** button:

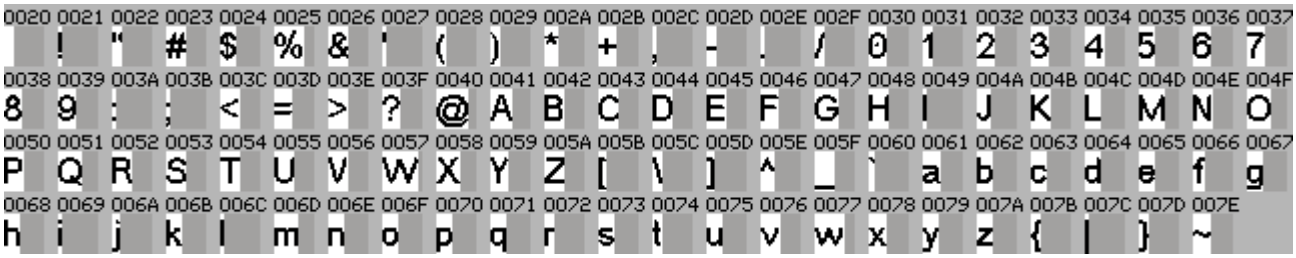
The resize dialog box shows the recommended new size for normal resizing:



Input a new size, choose ***Stretch or Squeeze***, and activate the ***LowPass Filter***:



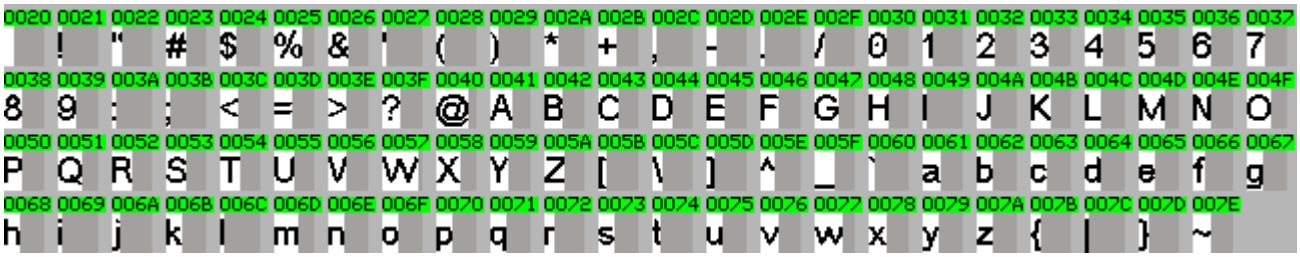
Press OK:



The font is now 20x20.

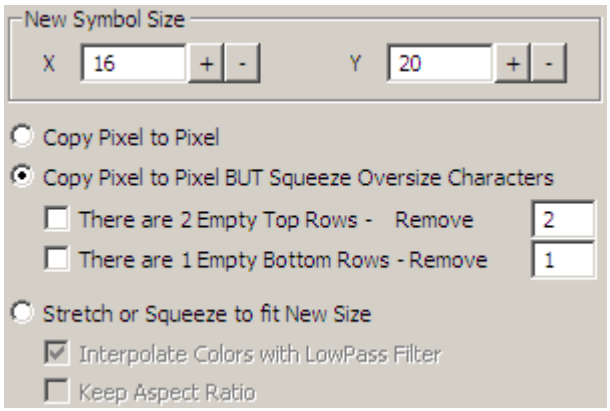
If the widest characters were squeezed to 20x16 the memory consumption could be reduced by a third.

Mark all characters with ***Mark All Symbols*** and remove any white space on the left and right side of the characters with ***Fit Symbol to Character***:

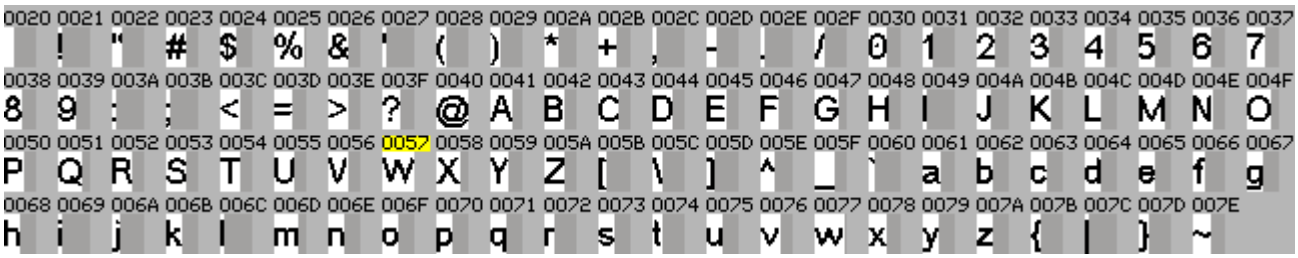


Press the ***Resize*** button:

Input a new size, and choose ***Copy But Squeeze***:



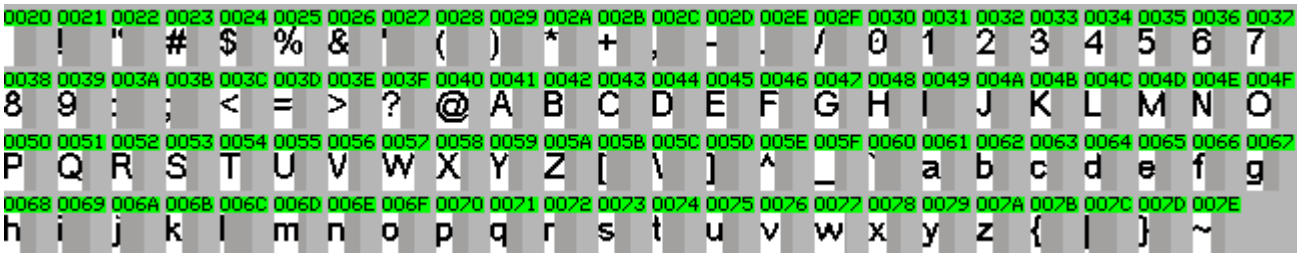
Press OK:



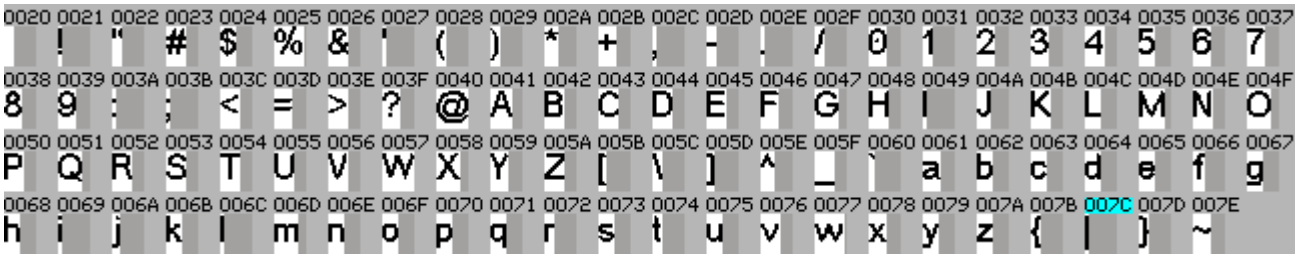
The font is now 16x20. The squeezed characters, in this case only W, are marked with yellow.

The characters have some white space in top and bottom that might not be needed, to remove it press ***Mark all Symbols***:

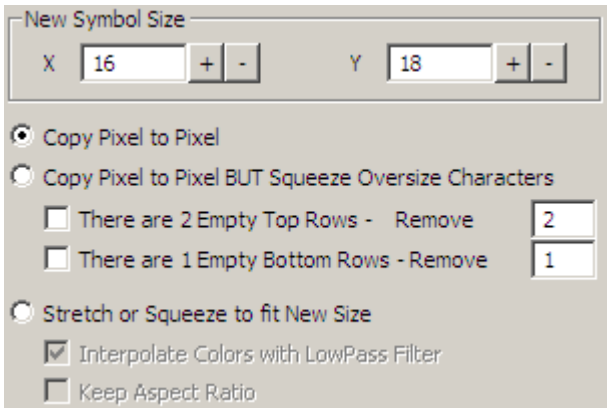
Move the characters to top with ***Move Marked Characters***:



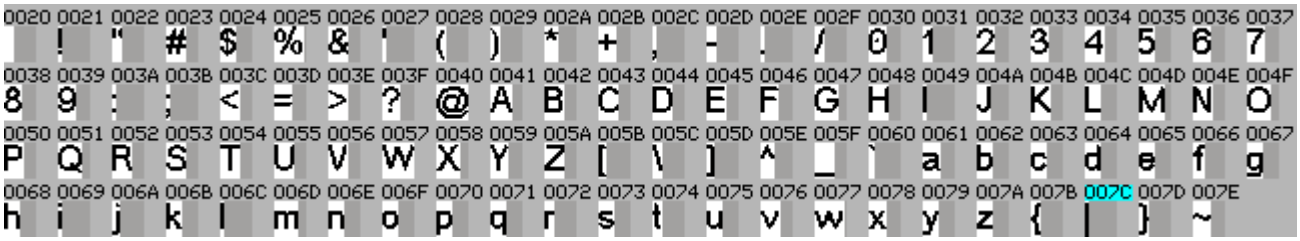
Use ***Get Biggest Character*** to find out how much can be cut off at the bottom:



The “lowest” character ends at 17 so the font can then safely be reduced to 16x18:

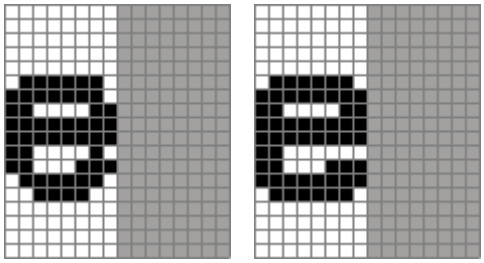


Press OK:



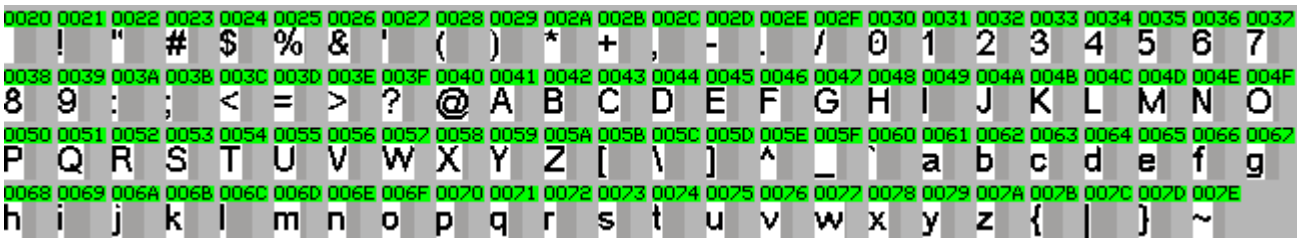
After these manipulations the look of the characters should be checked for unwanted side effects such as misplaced pixels or uneven line thickness. Any character that needs improvement can be edited in ***Character Edit*** mode by a right click on the character:

Before and after improvement:



When all characters have been updated character spacing should be added after each character.

Change to ***Font Edit*** and ***Mark All Symbols*** then use ***Leftset Character*** to add character spacing:



Your Font is ready for saving....

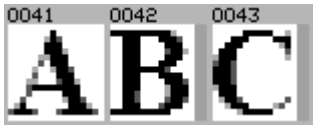
Press the ***Save All As…*** button in the Main tool bar to save the pixel data in the *Aa_16x18.c* file in the proper directory. The Sym and Code Point Character List files are saved automatically.

K: How to Add Characters to a Reshaped Font

Fonts based on an existing Windows font have a large amount of white space above and below the basic characters to make room for diacritics. Often much of this space is not necessary and the ***Scroll*** and ***Resize*** tools are used to squeeze characters in the y direction.

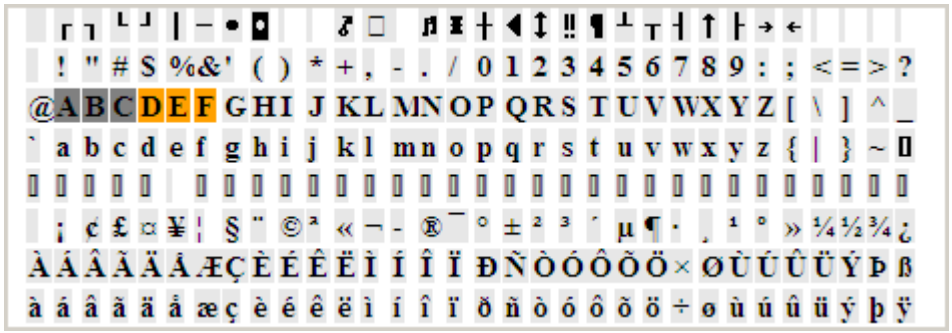
Before new Windows characters can be added to a squeezed font it has to be brought back to the original height, otherwise the added characters will be too small.

Assume an existing 16x16 font with the characters ABC:

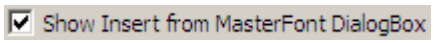


Press the ***Insert New Characters*** button:

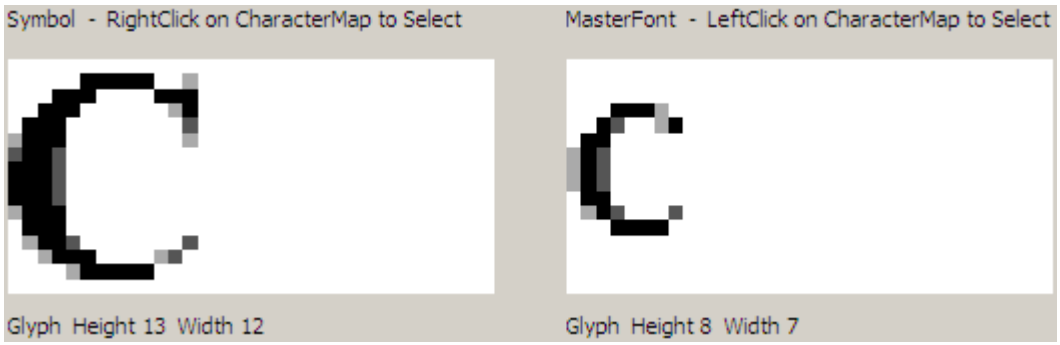
Select the new characters with the mouse:



Enable the ***Insert from MasterFont*** dialog:

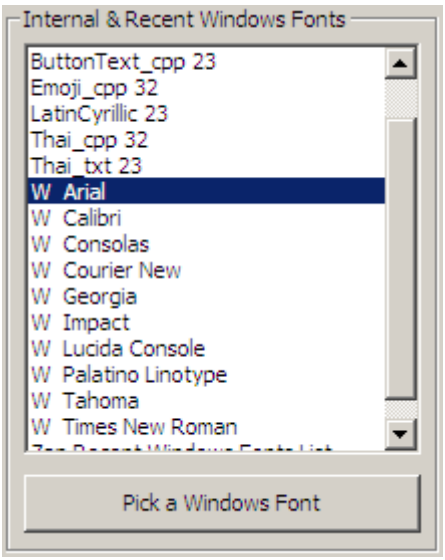


Press ***Insert*** to open the ***Insert from MasterFont*** dialog:



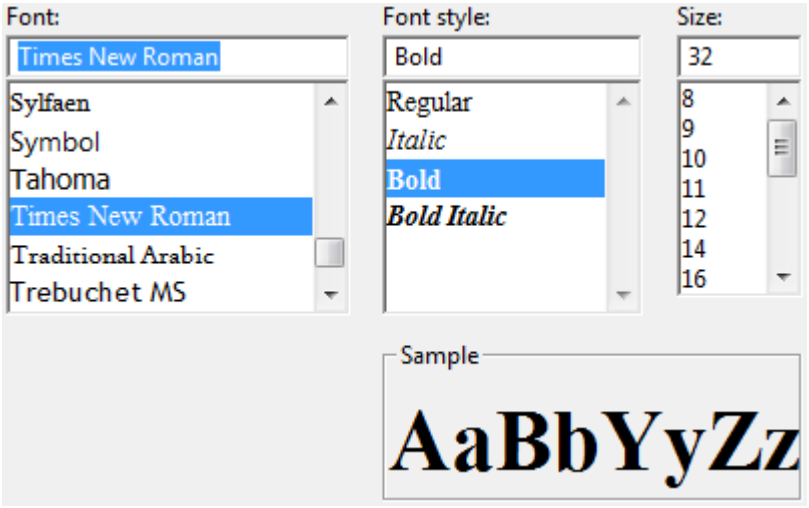
The Master Font has normal white-space and no seriffs, but the characters are 14 pixels high and with seriffs, so a proper Windows master font has to be found.

First try the recent fonts:

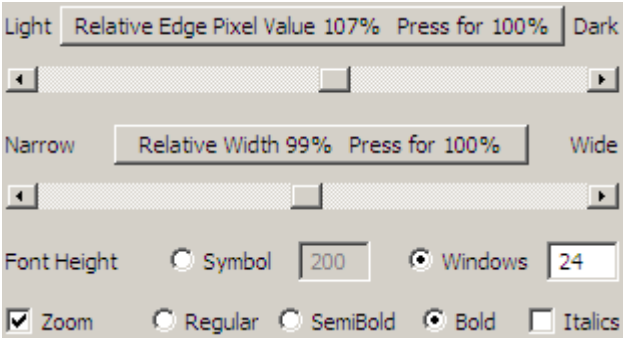


If they are not really enough press the ***Pick a Windows Font*** button to open the Windows font selector.

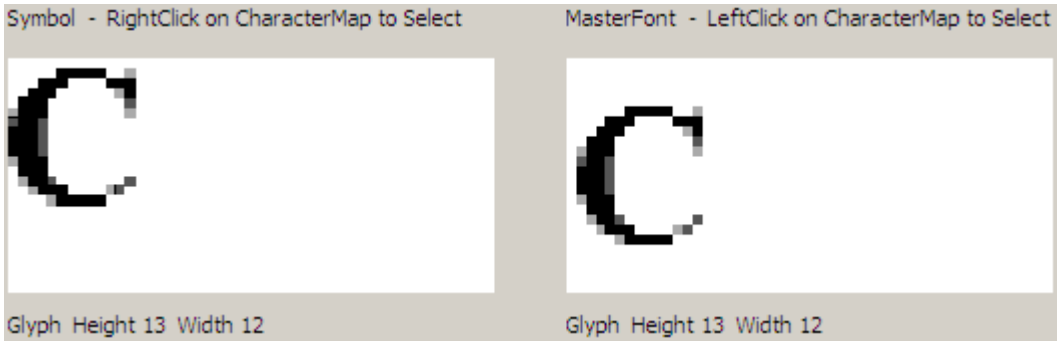
Search for a font with the same look as the original font, in this case Times New Roman seems like the best fit:



Press ***OK*** and experiment with the settings for the closest match with the original font. The original font was probably 24 pixels high. Press the ***Use TT Height*** button and choose 24:



The Master Font characters are offset by normal white-space, but otherwise look a lot like the original:

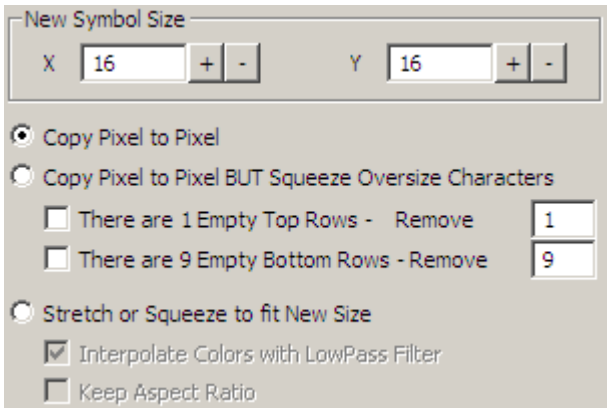


Press ***OK***.

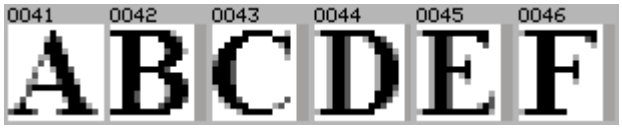
Choose the new characters with the mouse:

Scroll DEF up to the same position as ABC:

The original font was 16 pixels high. Press the ***Resize*** button and choose **Y** = 16:



Press ***OK***.



The original characters are unchanged and the new characters have the same size and look as the original ones.

06: How to Use IconEdit Tools and Make Palette Colors

A: How to Change Tool and Palette Colors

This example applies only to the color version of IconEdit.

IconEdit has a palette of various sizes of 2+1, 4, 16 or 256 colors. The palette shows the available drawing tool colors, and is always visible in Character or Symbol edit mode. In Font or Group mode it can be turned on or off by the ***Show Palette*** button:

Tool color is chosen by left click on one of the color buttons.

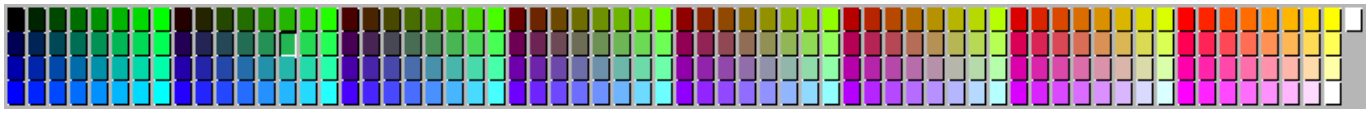
Tool color can also be chosen with the color picker:

Clicking on a pixel picks a new tool color. If the color is already present in the palette, the new color is selected. If the color is not in the palette, the already selected tool color is changed to the new color.

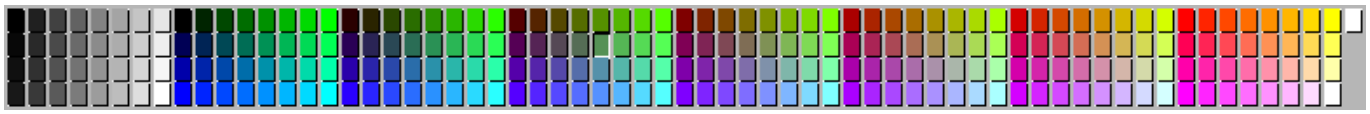
Background color can also be picked, right click on the tool color picker to get the background picker:

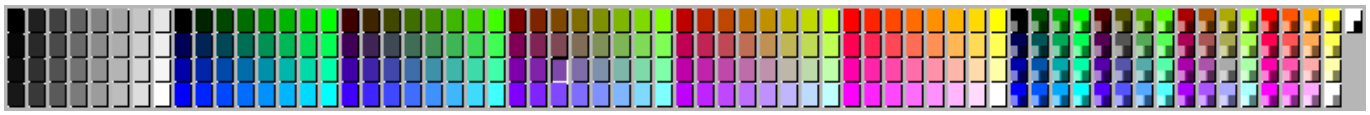
The background for the tool turns white to indicate the background picker. Clicking on a pixel picks a new background color. The new color can be seen in the status bar.

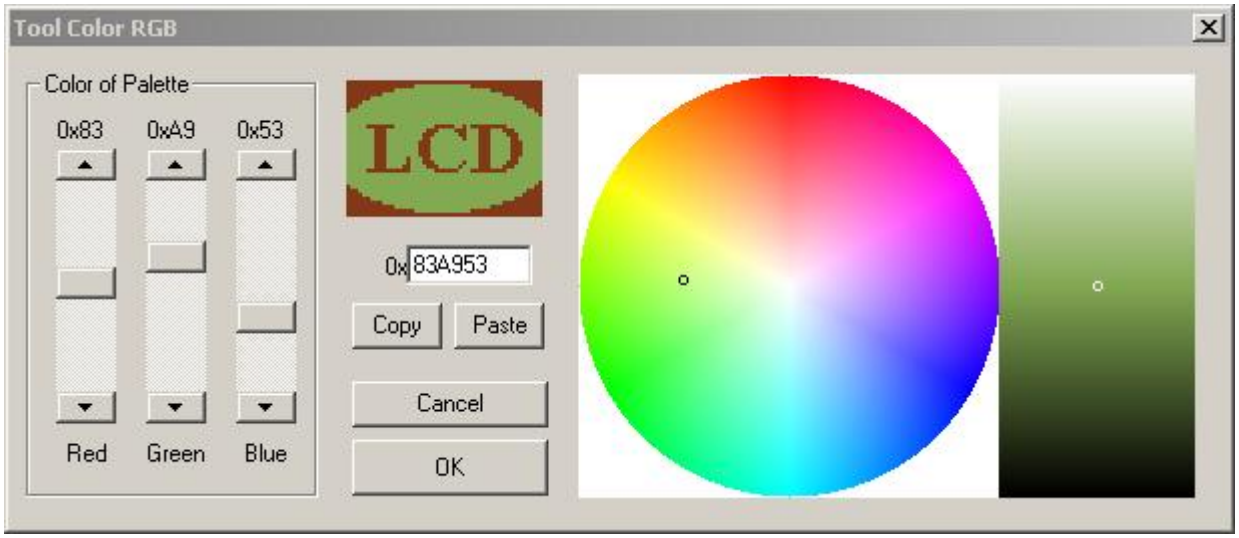
In Black & White, all the intensity levels, all grey tones and 8-bit RGB color mode the palette contains all the possible colors for that particular color mode, and can not be changed.



In 16 and 24-bit RGB and 32-bit ARGB color mode all the colors can be changed using a right click on the color button to start the tool color dialog box.



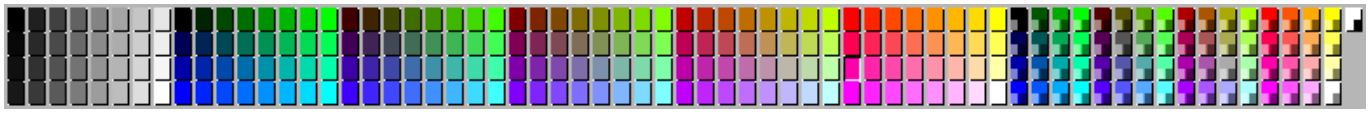




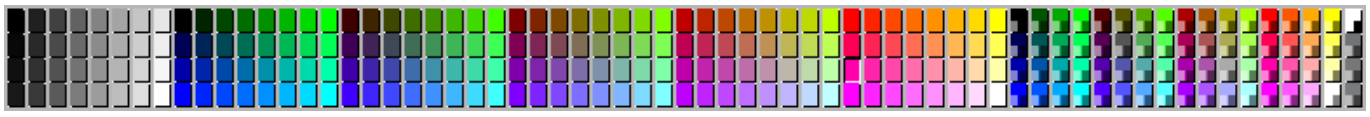
The control picture shows the new color as an ellipse, and the old color as text and surroundings.

Colors can also be changed by entering a 24-bit hexadecimal value, or by copy and paste a 6 digit hexadecimal number.

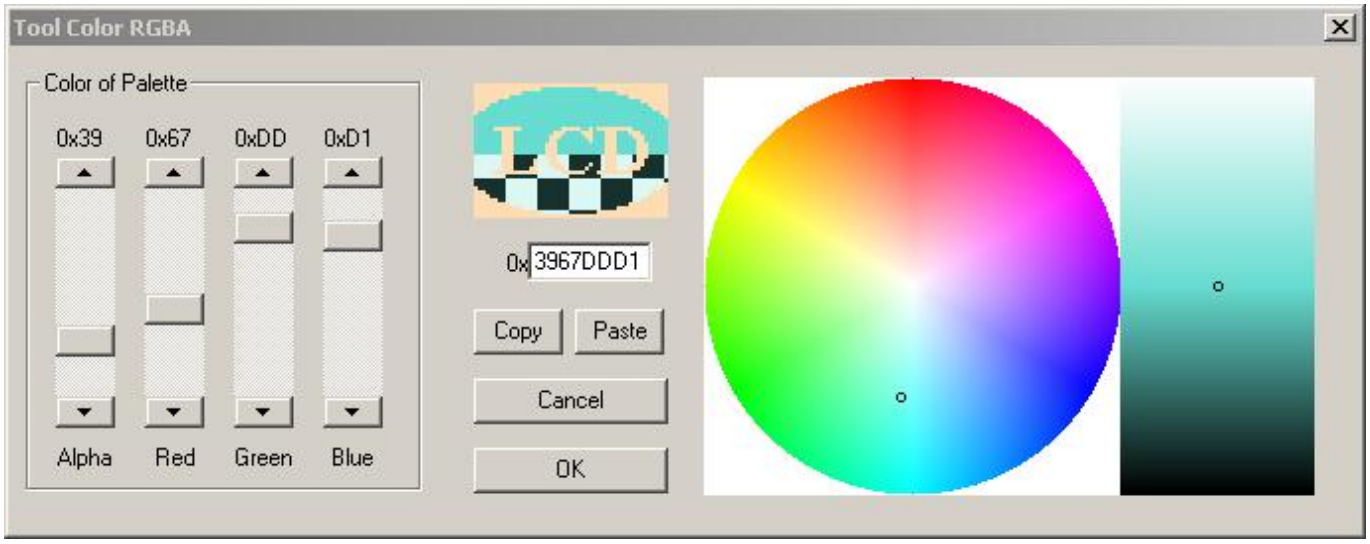
In the 32 bit ARGB color mode, the lower part of the palette button indicates the degree of transparency of that color by blending it with white and black. The background color may not be among the 256 colors in the working palette, so it has its own button at the far right:



If color blending is activated the pseudo colors for modifying the transparency are shown to the far right under the background color button:



The 32-bit ARGB the tool color dialog box has an additional scrollbar for the alpha value:

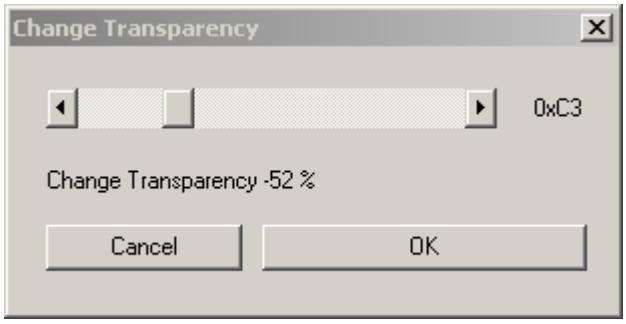


The control picture shows the new color as an ellipse, and the old color as text and surroundings. The upper half of the ellipse shows the basic color, and the lower half shows the basic color on black and white checkerboard background.

Colors can also be changed by entering a 32-bit hexadecimal value, or by copy and paste an 8 digit hexadecimal number.

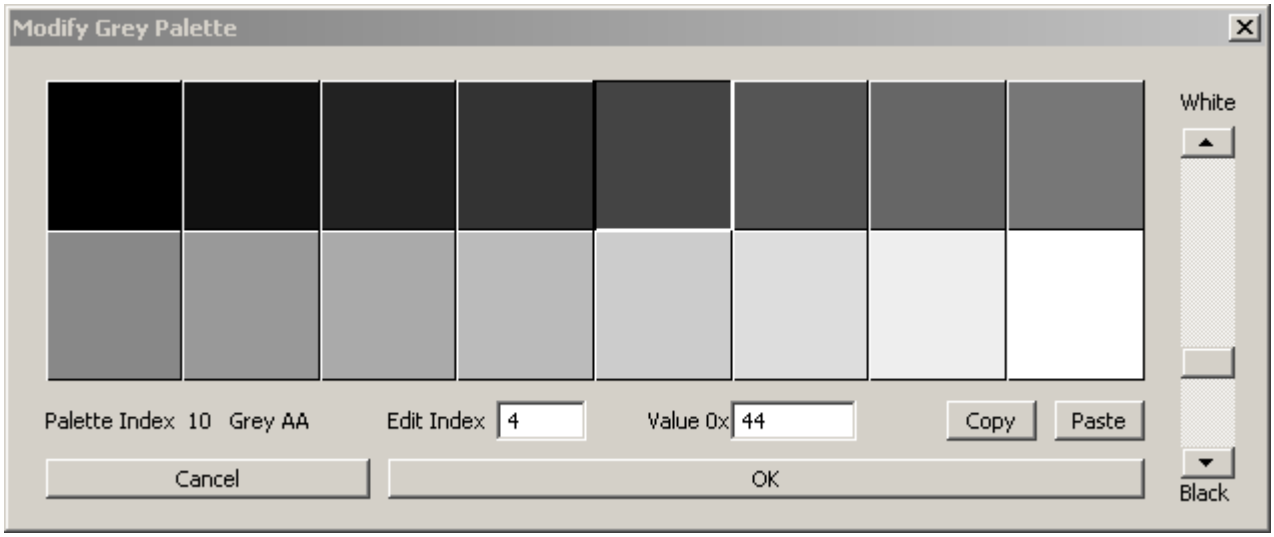
The change transparency pseudo colors is only available when Blend Colors is active:

The change transparency function does not have a basic color, only a change value:



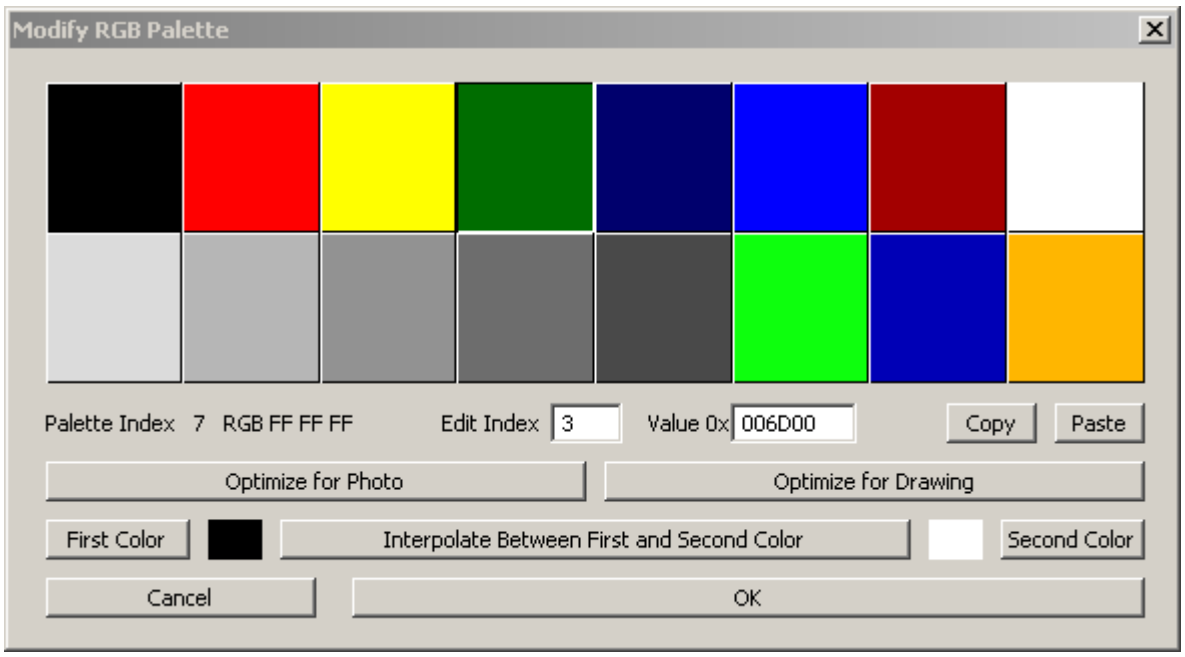
In all palette data format modes, palette sizes depend on the number of bits used by each pixel to index the palette colors. The palette colors should normally only be changed by accessing the palette directly in the modify dialog box:

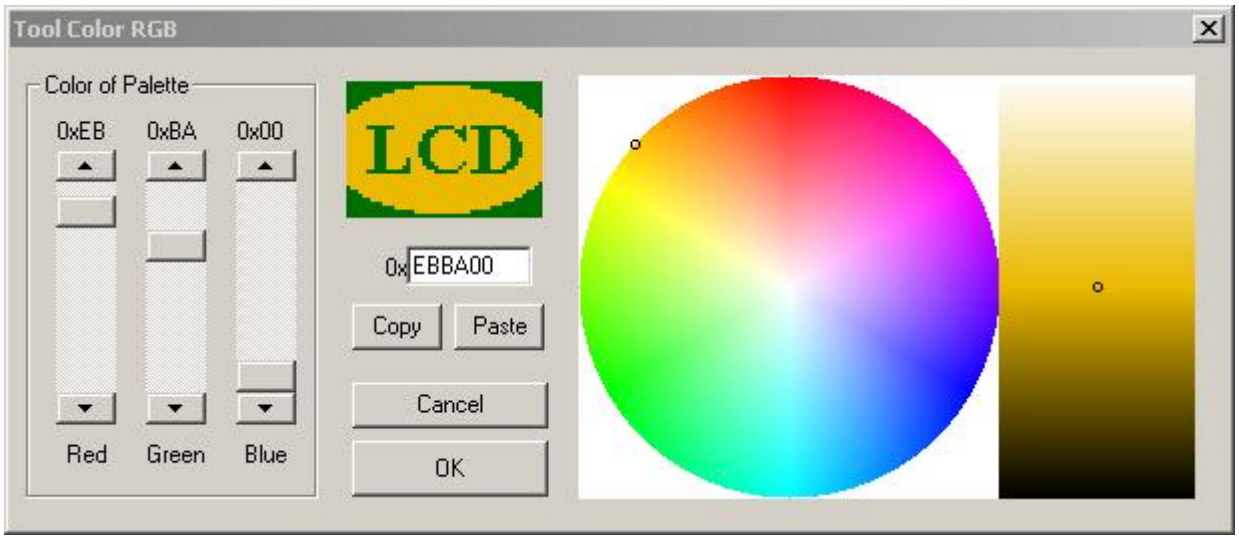
For grey palettes the colors are changed using the scrollbar.



Colors can also be changed by entering an 8-bit hexadecimal value, or by copy and paste a 2 digit hexadecimal number.

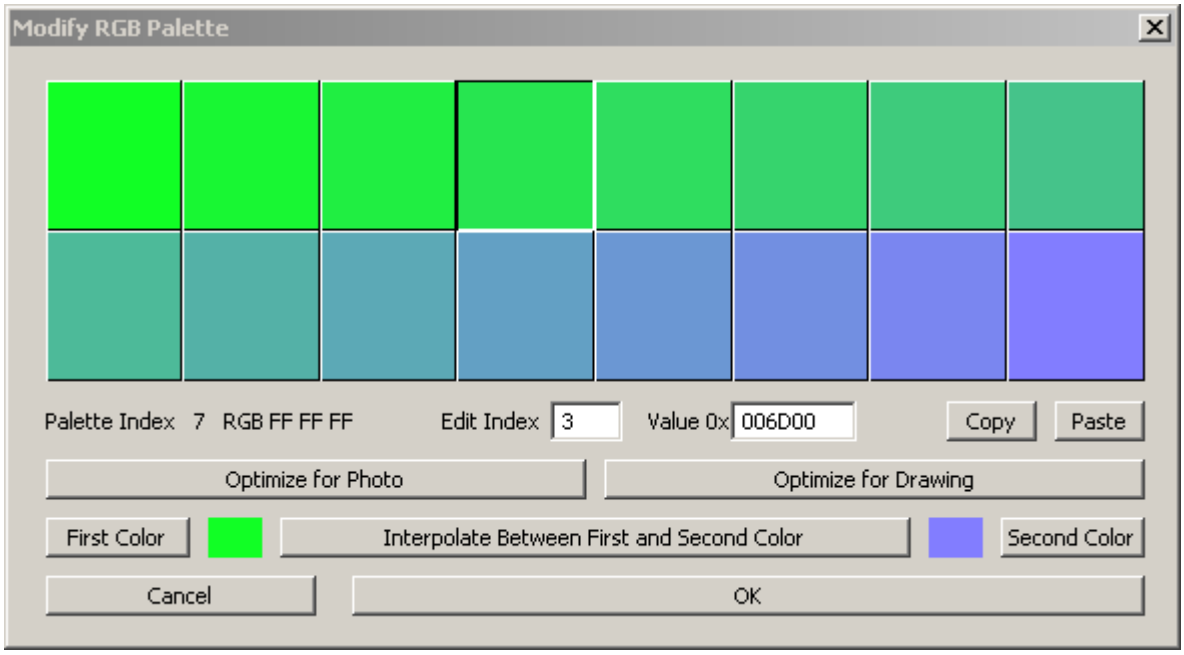
For color palettes colors are changed using a right click on the color to start the tool color dialog box.



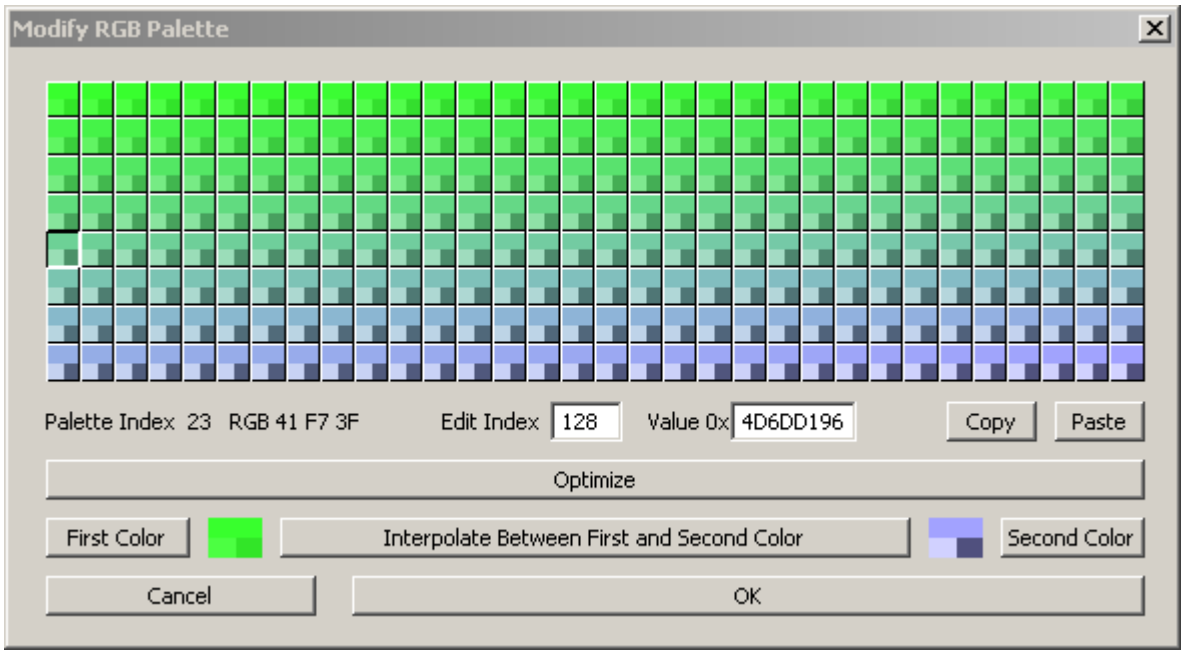


The control picture shows the new color as an ellipse, and the old color as text and surroundings.

A set of color shades can be created by interpolation between two colors, set first and second color to the endpoints and pres interpolate.

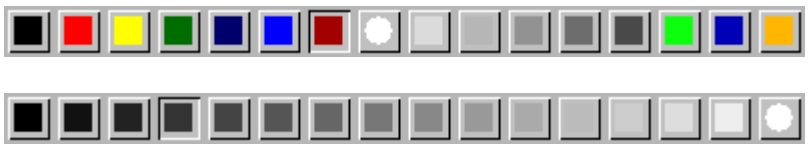


This also works for semi transparent colors in 32 bit ARGB color mode.



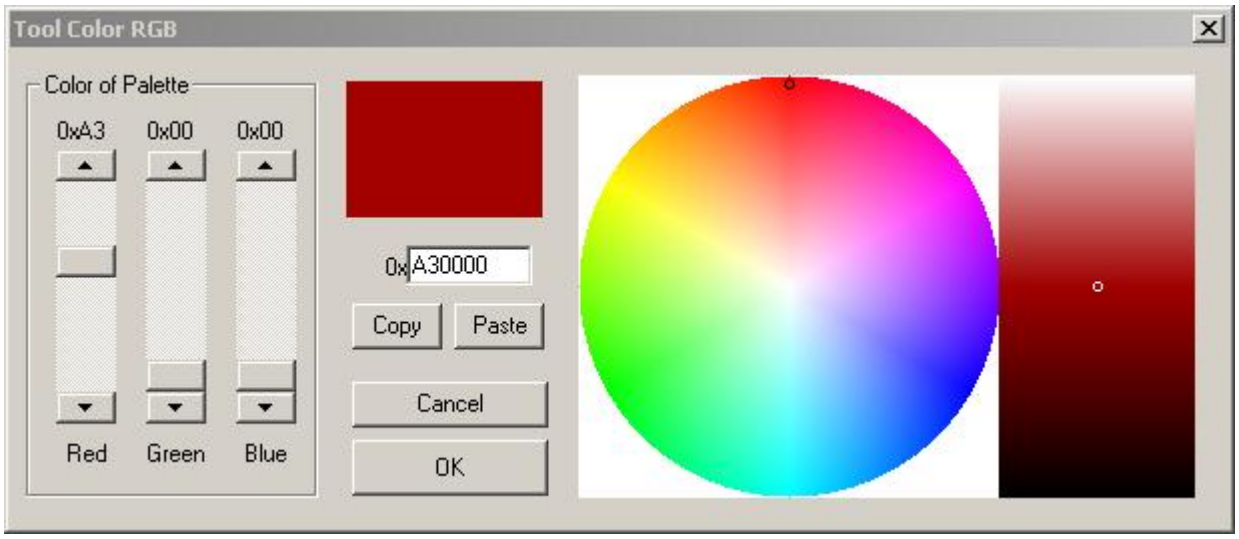
Colors can also be changed by entering a 24- or 32-bit hexadecimal value, or by copy and paste a 6 or 8 digit hexadecimal number.

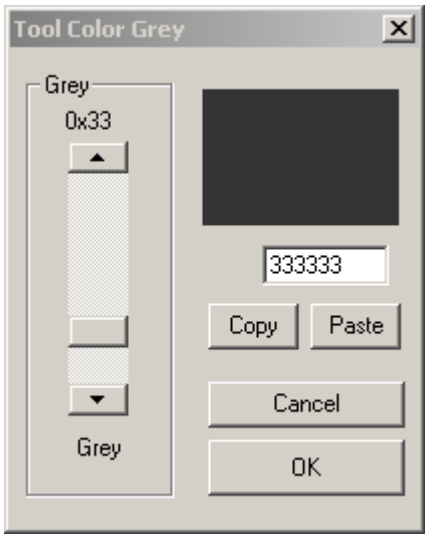
If only a single color should be changed right click on the color button to start the tool color dialog box.



In palette color modes this will generate a warning:

If you accept this the Tool Color dialog box will appear, either Tool Color RGB or Tool Color Grey depending on the color mode:



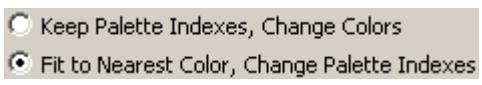


Colors can also be changed by entering a 24-bit hexadecimal value, or by copy and paste a 6 digit hexadecimal number. If colors are entered or pasted in the Tool Color Grey dialog box they are automatically converted to grey,

Palette colors are auto-generated every time palette mode is changed, so changing between grey and RGB or changing the number of bits per pixel generates a new palette.

To make it possible to use the same palette for a number of fonts or groups, existing palettes can be imported into the symbols:

If one palette mode is changed to another palette mode either colors or indexes have to be matched:



Please note that if the palette is common for a number of fonts or groups, modifying the palette for this data set will affect all symbols and data sets that share the palette, including unopened data sets, because they will use the modified palette the next time they are opened.

B: How to Use the various Flood Fill functions

IconEdit has 4 different flood fill functions for the tool color:

A **surface filler with tool color** that can fill large areas and does not leak through one pixel thick diagonal lines or borders. If it is used for filling thin diagonal lines or curves every line segment has to be filled separately.

A **line filler with tool color** that can follow a one pixel thick diagonal line around curves and corners. If used for surfaces it will leak through one pixel thick diagonal border lines or curves to adjacent areas and fill them too.

A **local substitute with tool color** that changes all occurrences of a color in a single symbol to the tool color.

A **global substitute with tool color** that changes all occurrences of a color in all the symbols in a font or group to the tool color. This filler is primarily intended for changing background colors in whole fonts or groups.

If you need to remove a figure, pick the background color as tool color before flood filling.

To change between the flood fill functions, right click the flood fill button in the Tool Box:

 Surface filler with tool color change to Line filler:

 Line filler with tool color change to Local substitute:

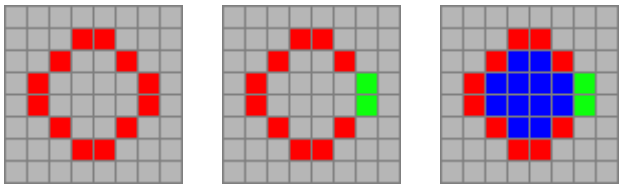
 Local substitute with tool color change to Global substitute:

 Global substitute with tool color change back to Surface filler.

The cursor will change to indicate the type of flood filler.

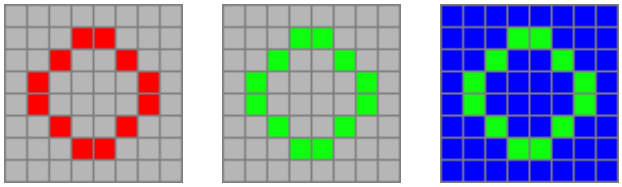
The difference between Surface and Line fillers lies in their ability to follow or be stopped by thin lines.

Using a Surface filler to try to fill the red circle with green and then its center with blue:



The Surface filler cannot fill the thin line of the circle, but is stopped by it.

Using a Line filler to try to fill the red circle with green and then its center with blue:



The Line filler can fill the thin line of the circle, but is not stopped by it.

In 32-bit ARGB transparency mode the surface and line flood fillers use the basic color regardless of its degree of transparency, and will therefore overwrite a color with the same basic color but with a different transparency. They will not normally change a fully transparent area to a new color because the transition from a visible pixel to a fully transparent pixel is also considered a border even if the basic color is the same for both pixels. Fully transparent pixels are considered borders for flood filling; this means that filling a fully transparent background will not affect the visible characters or figures. The substitute fillers use both basic color and transparency. They will even change a fully transparent area to a new color, and can also be used to separate a fully transparent background from a glyph of the same basic color.

Example: Six identical zeroes drawn in black with 25% transparency and filled with the surface filler at an entry point at the blue crosses. The first column is not filled; it only shows the 2 different entry points. The second is filled with red with 50% transparency and the third is filled with fully opaque green.

Warning: *The flood fillers use the basic color as search criterion, so trying to fill an area with the same basic color as the tool has no effect even though the alpha level may be different. The basic colors have to be different.*

.

C: How to use smoothing of the edges of Characters and Lines

This example applies only to the color version of IconEdit.

IconEdit can smooth edges by applying intermediate grey or color tones to mimic partly drawn pixels. This is only possible in color modes with a sufficiently large number of color shades between the tool color and the “background” that the tool color should be blended with.

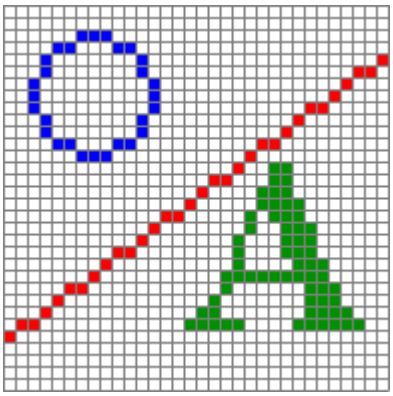
Please note that if the tool or the “background” is a color marked as a single transparency color the blending will produce fully opaque color shades between the transparent color and the tool color. If the blended colors should have a varying degree of transparency one of the four transparent data formats are recommended.

For lines, ellipses, rounded rectangles, triangles, arcs and text, right click on the button in the tool box:

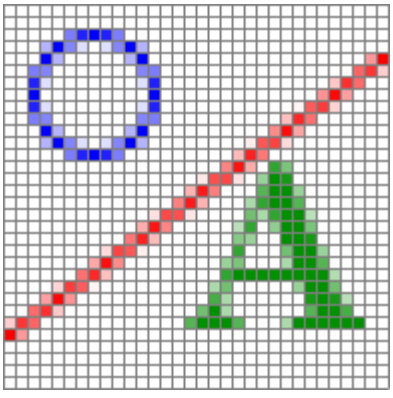
This gives a white figure indicating that smoothing is active:

Drawing example:

Without smoothing:



With smoothing:

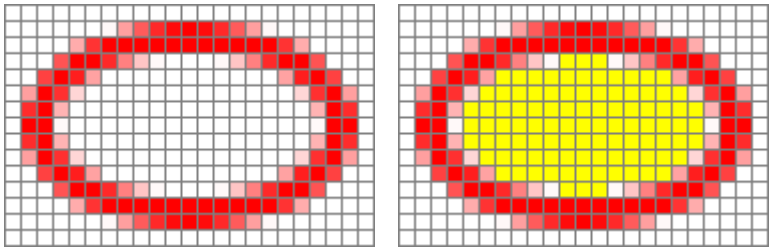


In actual size: Without smoothing:  With smoothing: 

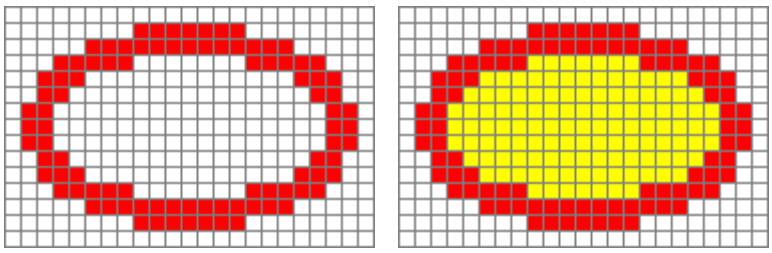
Note: The small arrows on some of the buttons indicate that the tool has extra mouse functions, but these have nothing to do with the smoothing.

D: How to Combine Smoothing with Flood Fill

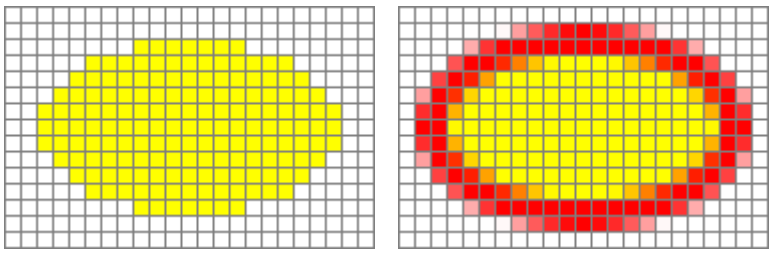
The flood fill functions change pixels with the same color as the start pixel, this means that it is stopped at the edge of a smoothed line or ellipse because the smoothed edge is not the same color as the “main” ellipse:



There are two alternatives, either draw without smoothing:



Or draw a yellow ellipse first, and then the ring around it:



In 32-bit ARGB transparency mode the whole ellipse including the smoothing is the same basic color, so the flood fillers can change the color of the ellipse but filling the centre is stopped by the smoothed edges.

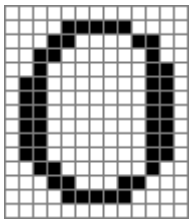
An alternative method is drawing at oversize and then squeezing, see chapter **04.C.2 A range of faces in high plane with anti alias by squeezing**.

E: How to Use Frames for making symmetrical figures

IconEdit can operate on parts of one or more characters or symbols by means of a blue frame. The content of the frame can be copied, moved, scrolled, mirrored and pasted.

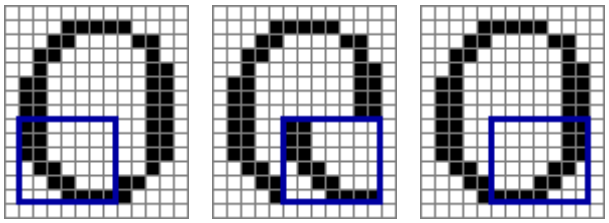
This example uses frames at the pixel level in symbol or character edit mode:

This is a capital O as generated in monochrome from Calibri with 24 cell height:

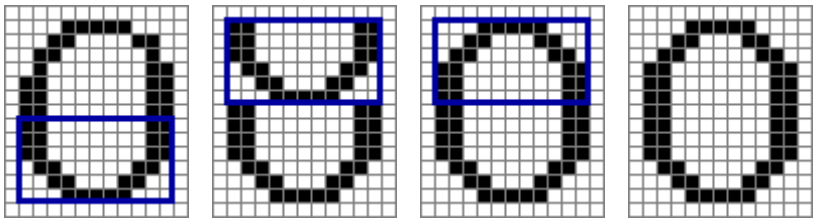


It can be made fully symmetrical by moving and mirroring:

Frame the lower left corner, move it to the right, and mirror it.



Then frame the whole bottom, move it to the top and mirror it.



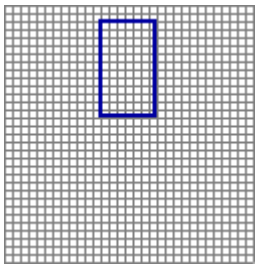
F: How to Use Frames for making Sub- and Superscripts

IconEdit can operate on parts of one or more characters or symbols by means of a blue frame. The content of the frame can be copied, moved, scrolled, mirrored and pasted.

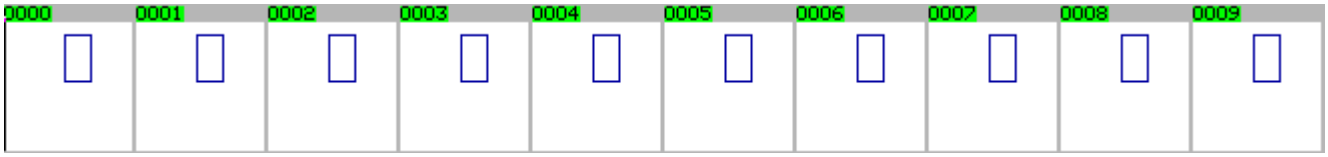
This example uses frames at the symbol level in Font or Group edit mode:

These characters are all 7x12 numbers in a 12x15 font. To make them superscripts in a 32x32 symbol group, copy to Clipboard with Ctrl C.

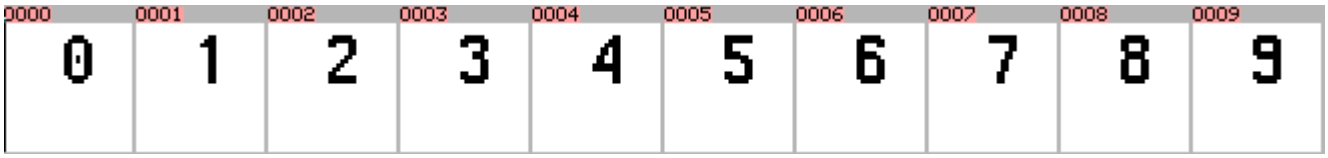
Make a 7x12 frame in one of the target symbols in ***Symbol or Character*** edit mode:



Change to ***Group or Font*** edit mode, select target symbols or characters for the numbers 0…9, and activate ***Paste Inside Blue Frame***:



Paste with Ctrl V, select **Copy Pixel to Pixel Inside Frame** and **Overwrite Frame in Marked Symbols**:



G: How to Use the Text in Frame tool

The tool can be invoked in two different ways one way is activating one of the *Write Text Line* buttons on the toolbar:

Change between the 4 types **opaque anti-alias**, **opaque normal**, **semi transparent anti-alias**, and **semi transparent normal** with right click.

Activating gives an empty frame with input from the keyboard:

The other way of activating is pasting a text from the clipboard to a symbol or character:

This gives a frame with as much of the text there is room for on the symbol or character:

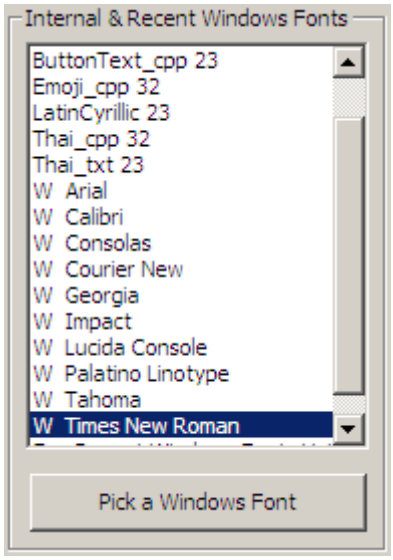
In both cases the frame can be moved around with the mouse, and a click outside the frame start a new writing session at the click point:

The text is treated as a single object, so changing color, smoothing, font or size affects the whole text:

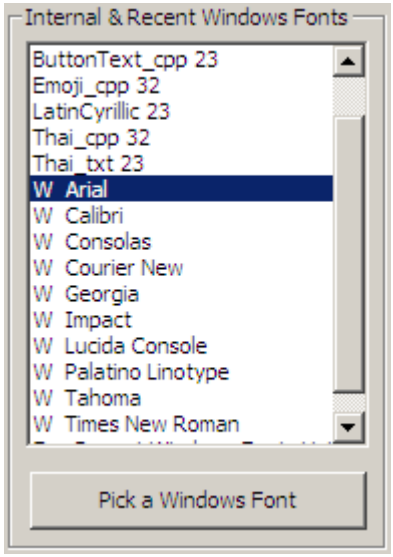
Color:

Smoothing:

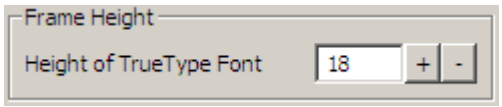
Font is changed with the virtual keyboard:



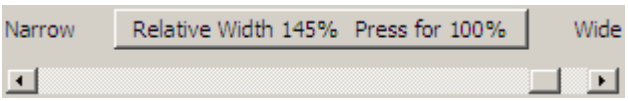
Change:



Height is changed with the virtual keyboard:

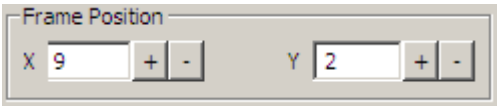


Width is changed with the virtual keyboard:



The text tool always writes a single line even if there may be <CR> or <LF> control characters in the text string. If the text should be multi line or have different colors or fonts each separate text object have to be finished by a mouse click outside the frame before the next is made. Pasting a new text to the text tool finishes the present text and start a new frame with the pasted text.

The text frame can be moved to a new position with the mouse or with the virtual keyboard.



The background for the text is normally transparent, but can be any valid color:

When this function is active any color chosen in the palette will be the new Background color, and the Tool color remains unchanged.

The text tool can be turned off by selecting any of the other mouse based tools.

H: How to Write Text in Foreign Languages

In this example, the aim is to write characters or text with characters that are not implemented on the keyboard, and the text is not available elsewhere for copy and paste.

The Master Font Selector can be used as a virtual keyboard for Unicode characters when text input mode is selected.

Left click the ***Write Text Line from Keyboard*** button:

This automatically changes the ***Master Font Selector*** to the ***Virtual Keyboard***:

For charge of smoothing or semi transparency right click the ***Write Text Line from Keyboard*** button:

It changes to white yellow or green to indicate anti-aliasing and/or transparency if the color mode supports this.

Text can now be entered at the keyboard or - if the characters are not available - from the ***Virtual Keyboard***.

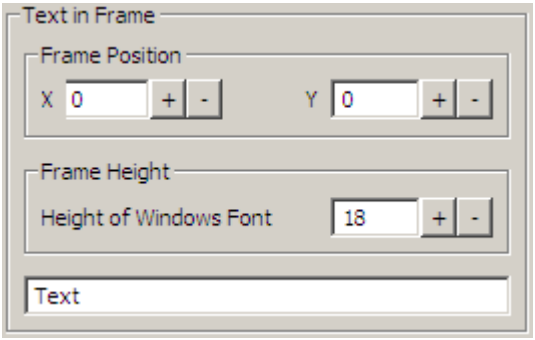
This opens the Font Selection dialog box as ***Virtual Keyboard***. Most fonts in Windows support a large number of different languages as Unicode on the Basic Multilingual Plane. The site www.unicode.org contains the official information about how and where the characters for a specific script are encoded.

In this example, we will use Cyrillic positioned at 0x0400….0x04FF.

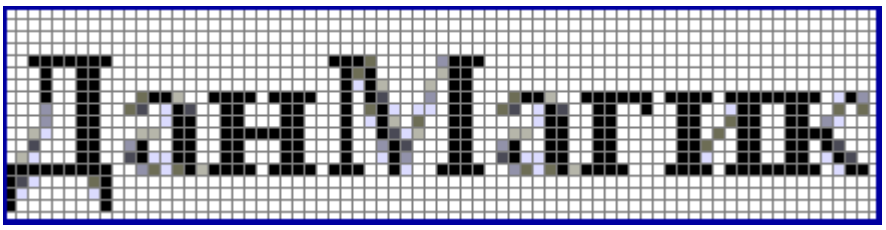
Scroll down to 0x0400 to get the Cyrillic page:

The rectangular characters and characters highlighted in pink are characters not defined in the presently selected Master Font. In this case, the missing characters are obsolete characters, and not used in modern “Cyrillic” languages such as Belarusian, Russian or Ukrainian.

To make the virtual keyboard and the physical keyboard active simultaneously write in the text window. Please note that the two keyboards compete about placement of the caret:



This is DanMagic transliterated to Cyrillic and written with smoothing in 8-bit RGB:



The pixels used for smoothing may appear slightly colored when magnified on the PC screen, but this is invisible both on the target display and on the PC screen in natural size:

ДанМагик

I: How to Make and Save Semi-transparent Colors

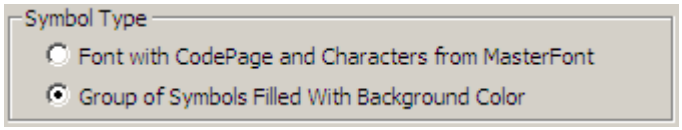
This example applies only to the 32 bit ARGB color mode in the color version of IconEdit.

To make shades like this:

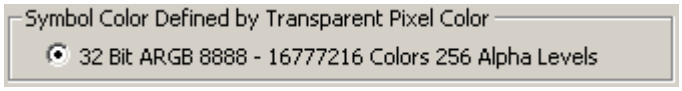
Press ***New Font or Symbol Group***:

This opens the create dialog box:

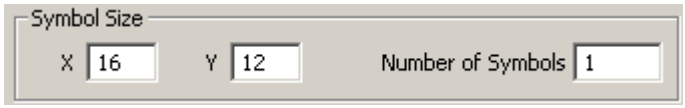
Choose how characters or symbols should be organized and shown initially:



Choose a color format:

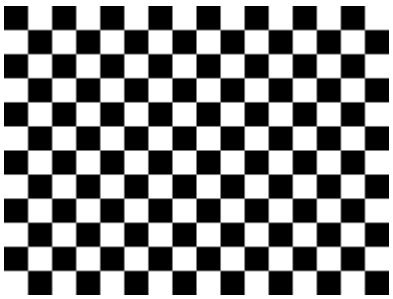


One symbol:



Press OK.

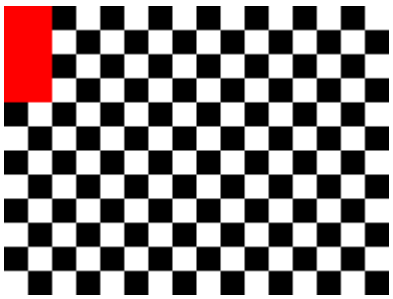
The background is transparent, to make this more visible press the  on the ***Color and Size of Transparent Background*** button and change the checkerboard color to black and white with the and  buttons:



Left click the bright red in the palette window to select it as drawing tool:

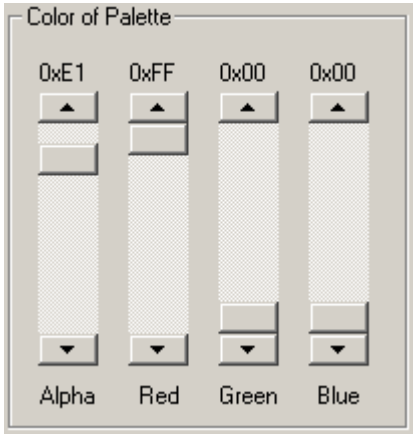
Press ***Solid Rectangle*** in the tool bar:

Draw 2x4 a rectangle



Right click the next button on the Palette get the RGBA tool color dialog box:

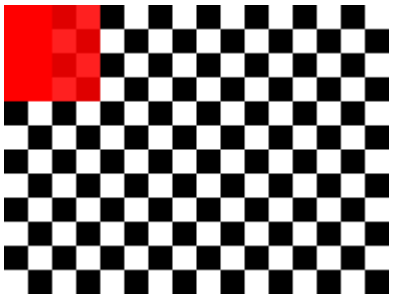
Change the color to bright red and Alpha to 0xE1



In the tool color dialog box the chosen transparency is displayed on checker board background

Press OK,and the palette button shows the degree of transparency on white and black background:

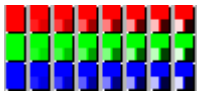
Draw the next 2x4 a rectangle.



Repeat the process with Alpha C1 A1 81 60 40 and 20 to get this:

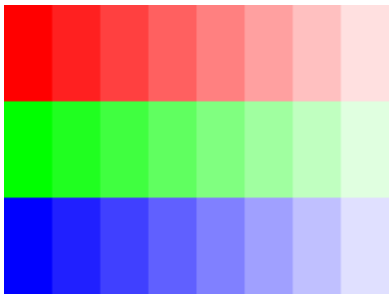
If the whole process is repeated with green and blue, it will produce this:

The Palette now contains all the transparent colors:

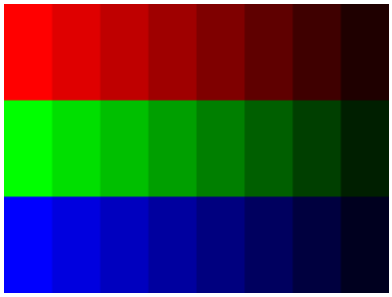


The appearance of the transparent area can be changed by the ***Color and Size of Transparent Background*** button, + and – change the size and white and black change the colors of the checkerboard:

White + white produces this:



Black + black produces this:



Although the transparent area is shown in different ways, it is still the Transparent Color as reported by the mouse movement shown here:

```
A RGB 60 FF0000
```

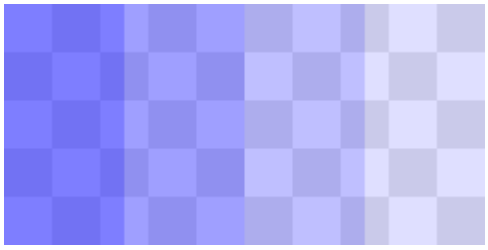
This value is the value saved in the picture symbol file regardless of how the background is visualized.

Press the ***Save All As…*** button in the main tool bar to save the pixel data in the *ColorShades.c* file in the proper directory. The Sym and Palette file are saved automatically.

J: How to Modify the Transparency and Make Shading

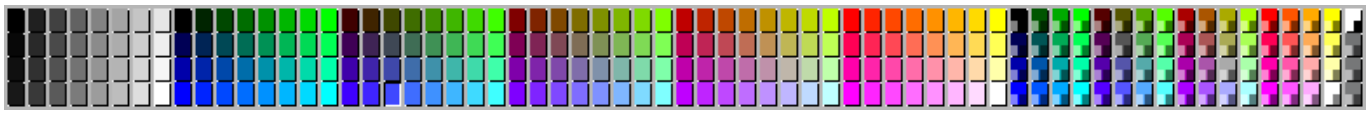
This example applies only to the 32 bit ARGB color mode in the color version of IconEdit.

Shading can be made with any of the drawing tools except floodfill, in this example we will write a text on a semi transparent background:



Activate blending to make shading possible:

This makes the transparency buttons visible to the far right:

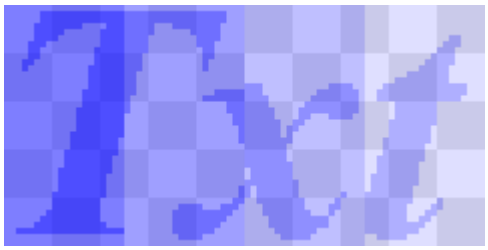


Right click the ***Change Alpha Level*** button and scroll to Change Alpha level 24 Steps:

Press OK.

Activate a drawing tool such as ***Write Text Line from Keyboard***:

Write “Txt” and move it in position:



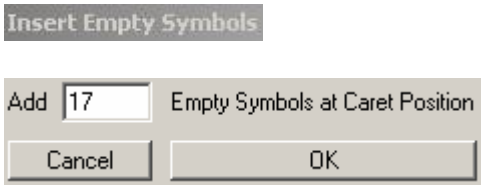
K: How to Add Symbols to a Group

Symbols can be added to a group in any number at any position within the maximum of 65536 symbols. The symbols are inserted in ***Group Edit*** mode at the caret position much like characters are written in a text editor. As there is no Code Point Character List has to be maintained a new symbol is just inserted at the caret position, moving the following symbols to higher group index values.

A mouse click anywhere on the Group Edit window puts the caret at the nearest boundary between two symbols.

Insert One Empty Symbol shifts all symbols after the caret position one position up before a symbol with background color is inserted.

Insert Multiple Empty Symbols shows a dialogbox:



Enter a number and press OK. This shifts all symbols after the caret position 17 positions up before 17 symbols with background color are inserted.

Warning: *The insert functions changes the symbol number of all symbols after the insert point.*

L: How to Add Characters to a Font

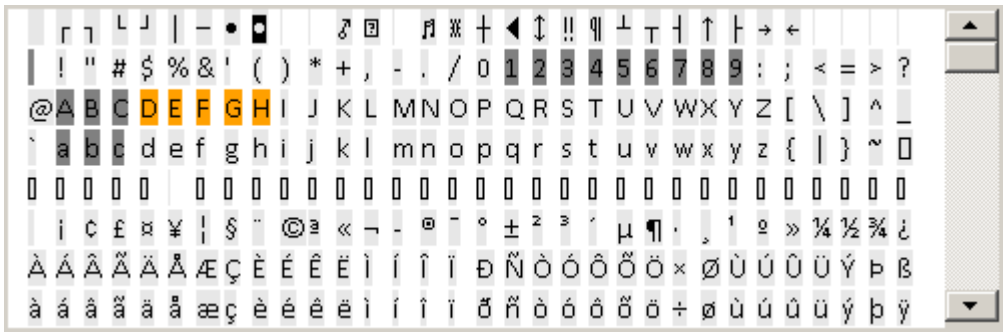
Characters can be added to a font in any number at empty positions within the maximum of 65536 character symbols. The characters are inserted in ***Font Edit*** mode either by pasting a text or by specifying Code Points.

See also chapter **05.D.1 How to Fit New Characters to an Existing Font** and **06.O How to Make Narrow Characters and Fonts** about methods for matching the look of the master font and the target font.

1: Insert Empty Symbols at Code Points

Insert Anywhere lets you choose Code Points anywhere in the 65535 character space of Unicode plane zero:

Insert Empty Symbols shows a dialog box where already occupied Code Points are marked in grey:



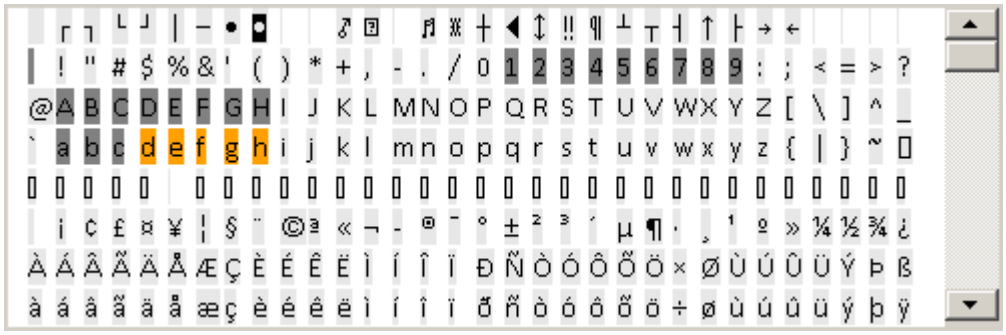
New characters can be marked with the mouse, and are highlighted in orange. Press:

This will add 5 empty characters.

Already existing characters keep their Code Point regardless of how many characters are inserted.

2: Insert From Master Font at Code Points

Insert New Characters shows a dialog box where already occupied Code Points are marked in grey:



New characters can be marked with the mouse, and are highlighted in orange. Press:

This will add 5 characters based on the Master Font.

Already existing characters keep their Code Point regardless of how many characters are inserted.

3: Modify Symbol at Code Point

If you want to insert a new or different character symbol at an existing character position, you have 3 options:

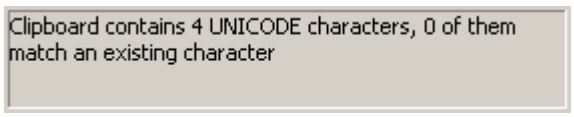
A: Update the character symbol content in ***Character Edit*** mode.

B: Paste the character symbol content form another Font or Group in ***Character Edit*** mode.

C: Delete the symbol and then insert a new or empty symbol at the same position in ***Font Edit*** mode.

4: Paste New Characters from Master Font at Code Points

Paste from ClipBoard shows a dialog box with the text and how many of characters already exist in the target font:



Press:

This will add all 4 characters based on the Master Font.

Already existing characters keep their Code Point regardless of how many characters are inserted.

M: How to Modify Several Characters or Symbols Simultaneously

Selected characters and symbols can be modified together in Font or Group Edit Mode.

If no symbols are selected the modify tools are greyed:

All can be selected with the ***Mark all Symbols*** tool:

Selection of chosen symbols can be done with the mouse-click, Shift+mouse-click and mouse-swipe much the same way as is common for text editors. To select several subgroups of symbols use Ctrl+mouse, this makes the mouse operations act as toggle instead of normal select.

Once some or all symbols are selected the tools that can be used for modification are shown with a green frame to indicate that they operate on the selected characters or symbols only:

If the operation is not possible for some of the tools, the tool stays grey.

N: How to Convert between Proportional and Mono-Spaced Characters

Characters can be mono-spaced or proportional, and proportional characters can be left-set, right-set or fit to the glyph.

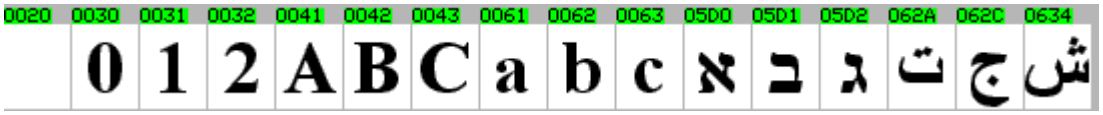
This example will demonstrate how to convert a font to be a mono-spaced or proportional font. Assume this font example with SPACE, 3 Numbers, 3+3 Latin, 3 Hebrew and 3 Arabic characters:

1: Convert to Mono-Space

The mono-space function fills out the symbol with background and moves the glyph to the center of the symbol, so be sure to make any ***Resizing*** first:

To make a mono-space font select all characters with the ***Mark All Symbols*** button:

Then mono-space all characters with the ***Monospace and Center Character*** button:

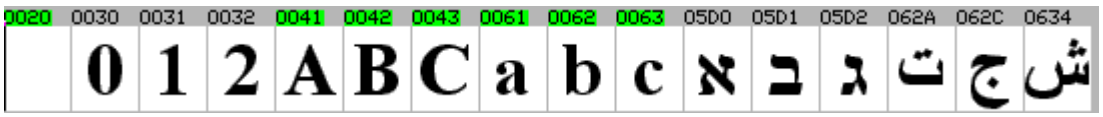


The mono-spaced font is ready.

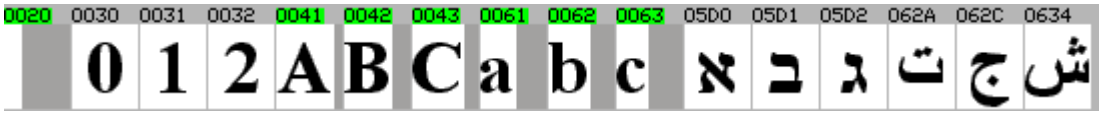
2: Convert to Proportional

To convert a mono-spaced font to proportional the needs of different scripts should be considered. Numbers should have the same width with a small space between each character, so they must be mono-spaced separately. Latin is written from left to right with a small space between each character, so there should be a space on the right side of the glyph. Hebrew is written from right to left with a small space between each character, so there should be a space on the left side of the glyph. Arabic is written from right to left with connected characters, so there should be no space on either side of the glyph.

Mark SPACE and the Latin characters with the mouse:



Then left-set the characters with the ***Leftset Character*** button:



Mark the Hebrew characters with the mouse:

Then right-set the characters with the ***Rightset Character*** button:

Mark the Arabic characters with the mouse:

Then clean the characters with the ***Fit Symbol to Character*** button:

To make the Numbers look mono-spaced they should be treated separately as a new font, mark the Numbers with the mouse:

Move the Numbers to the ClipBoard with ***Cut to ClipBoard***:

Paste the Numbers to a new font with the ***Fit Symbol to Character*** button:

Press OK:

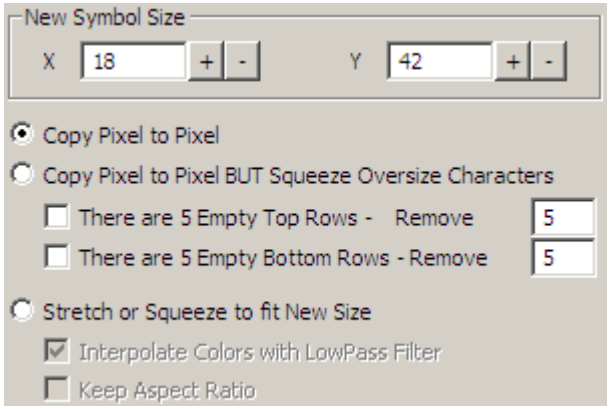
To make mono-space Numbers first select all characters with the ***Mark All Symbols*** button:

Then mono-space all characters with the ***Monospace and Center Character*** button:

Then clean the characters with the ***Fit Symbol to Character*** button:

Resize the font with the ***Resize All Symbols*** button:

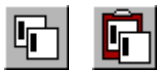
Resize suggests the largest character; add 1 pixel for internal space in the numbers:

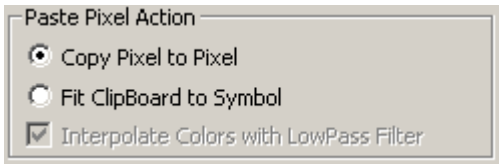


Press OK:

Then mono-space all characters with the ***Monospace and Center Character*** button:

Copy all characters to the Clipboard with ***Copy to ClipBoard*** and paste back into the original font with ***Paste from ClipBoard***:





The proportional font is ready.

O: How to Make Narrow Characters and Fonts

Characters can be made narrower in many different ways; this chapter demonstrates how to make narrow characters by squeezing one character at a time, by squeezing a whole font, or by generating narrow characters directly from a Windows Master Font.

1: Squeeze One Character at a Time

The character width can be modified in **Character Edit** mode with the ***Squeeze or Stretch*** function by mouse click anywhere on the character and movement left or right:

```
Character Edit
```

Before:

After squeeze:

After stretch:

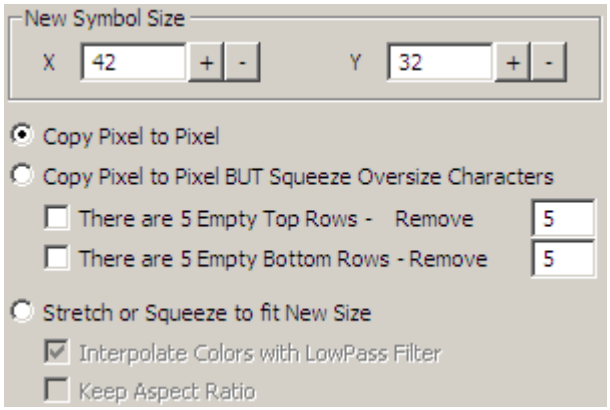
The character is ready.

2: Squeeze the Whole Font

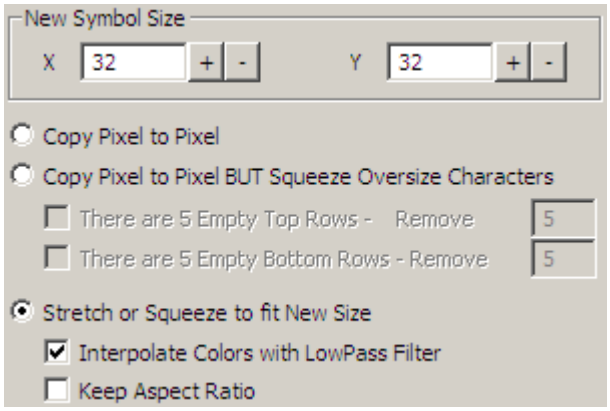
The character width can be modified in **Font Edit** mode with the ***Resize All Symbols***. This method is preferable for making a narrow proportional font from an existing font:

```
Font Edit
```

Press the button to get the resize dialog box:



Select a new width and Stretch or Squeeze:



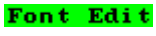
Before squeeze with symbol width 42:

After squeeze with symbol width 32:

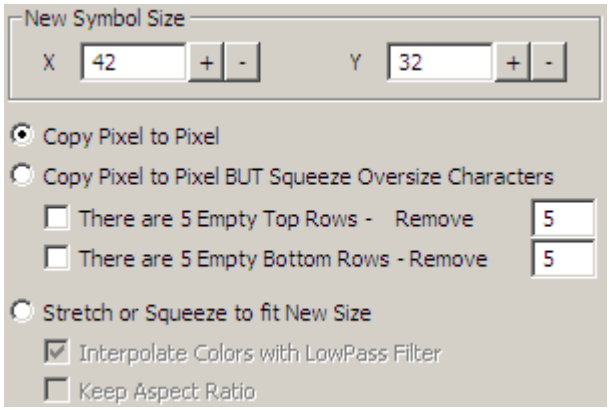
The font is ready.

3: Squeeze Part of the Font

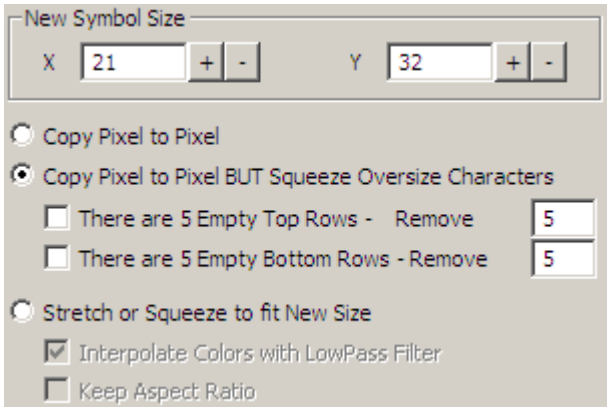
The character width can be modified in **Font Edit** mode with the ***Resize All Symbols***. This method is preferable for making a narrow mono spaced font from an existing font:



Press the button to get the resize dialog box:



Select a new width and Copy Pixel to Pixel but Squeeze Oversize:



Before squeeze with symbol width 42:

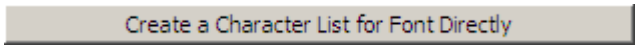
After squeeze with symbol width 21, the yellow highlight indicates characters where the glyph is squeezed:

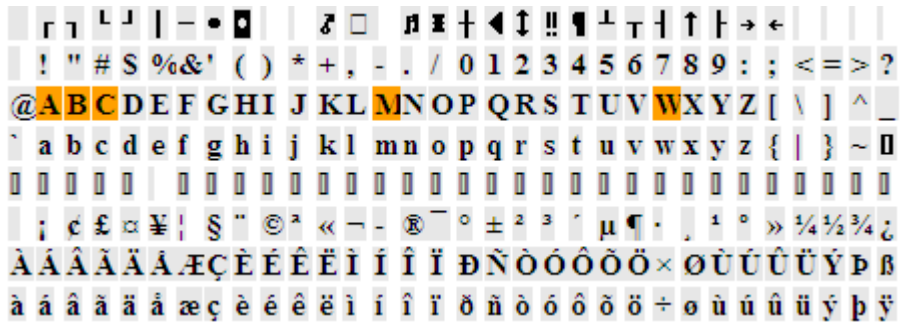
The font is ready.

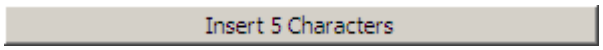
4: Generate Narrow or Wide Fonts from a Windows Master Font

This option works best for grey-tone fonts.

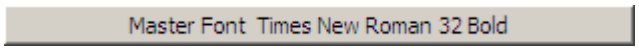
The character width can be modified in the Master Font setup with the Relative Width option. This method is preferable for making a ***New*** proportional font:



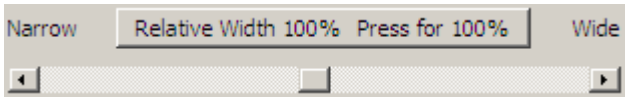


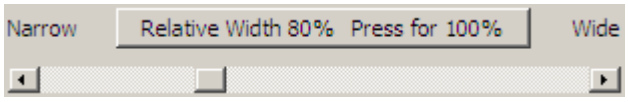


Go to MasterFont setup:



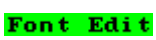
Choose a relative width:



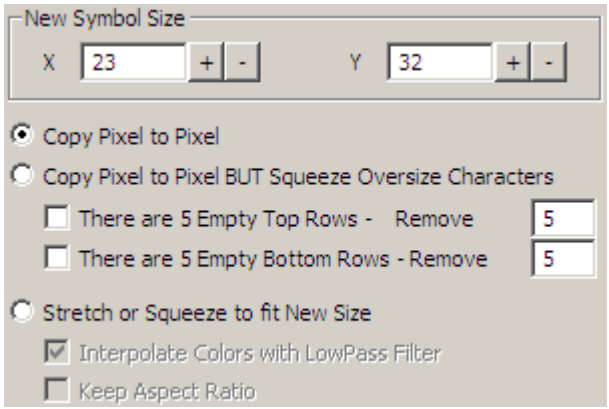


Press OK in Master Font setup.

Press OK in New:



Remove any unused space with the ***Resize All Symbols***. The best guess is already set:



Select Copy Pixel to Pixel and press OK:

The font is ready.

P: How to Add Smileys and Emojis to a Font.

Unicode has a number of emojis placed on Plane 1 at code-points between 1F300 and 1F6FF witch means that characters need 32 bit addresses. To access the higher planes Unicode has a system of high and low 16 bit surrogate code-points so the characters can be written directly by a 16 bit text with a combination of the two surrogates.

Notice: *The build-in Windows fonts in the different versions of Windows are usually according to the Unicode standard at the time the Windows version was created. This means that for higher Planes where the emojis are the support in Windows XP is non existing, Windows 7 have a few ancient scripts but no emojis, Windows 8.1 has many emojis, and Windows 10 is well supplied with both ancient scripts and emojis.*

1: Put Emojis in 16 bit Address Space

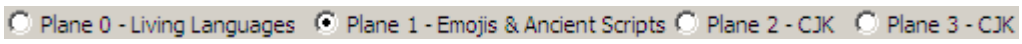
The IconEdit 16-bit version automatically puts characters from Plane 1 2 & 3 from the range 10000 to 3FFFF into the Private range E700 to F8FF as long as there is room. For the IconEdit 32-bit version high plane characters can be moved to and from Plane 0 with a simple command.

The insert process is as follows:

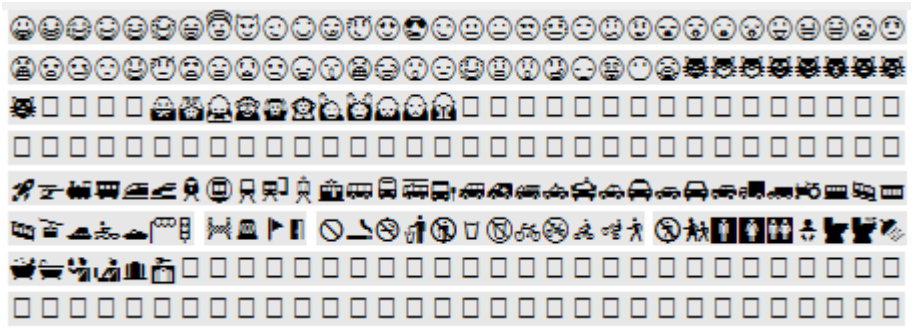
Assume a small font:

Press **Insert New Characters**:

This opens the Insert Character dialog box, choose Plane 1:



Scroll down to page F6:



With left mouse click mark a smiley and with control + left mouse click mark a cable car:

Press **Insert 2 Characters**:

The characters are placed automatically at E700 and E701 so that all characters in the font have 16 bit addresses by the 16-bit version.

For the 32 bit version switch to **Scripts & Symbols**

Press the **Move High Plane Symbols**.

The emoji now has a 16-bit address:

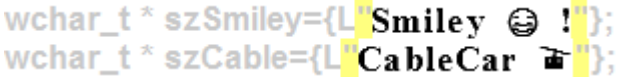
Any characters from higher planes that are moved to the private area in Plane 0 gain status as Combined Characters witch means that IconEdit knows where the character came from:

The new Code Point E700 can now be used instead of the original Unicode Code Point 1F603.

The original 32 bit text and the 16 bit font is combined:

```
wchar_t * szSmiley={L"Smiley \uE700 !"};
wchar_t * szCable={L"CableCar \uE701"};
```

This is how it will look at the target display:



2: Put Emojis in 8 bit ASCII Address Space

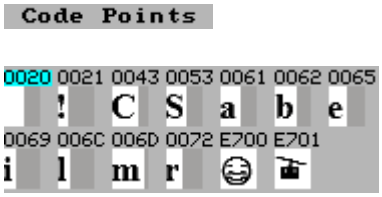
All the characters in the text strings have 16 bit Code Points, but for Latin characters a memory reduction factor of two can be achieved by converting the strings to UTF-8 format because Code Points 00 to 7F also called ASCII need only one byte in the UTF-8 format:

```
wchar_t * szSmiley={L"Smiley \uEE\u9C\u80 !"};
wchar_t * szCable={L"CableCar \uEE\u9C\u81"};
```

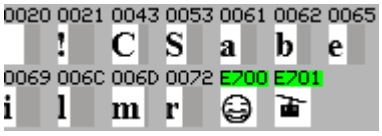
The catch is that all other characters need several bytes, in the case of the emojis 3 bytes each.

The ASCII standard has a range of rarely used modem control characters from 10 to 1F and placing the emojis there can save 2 bytes every time it is used.

Change to the **Code Point Edit** window:

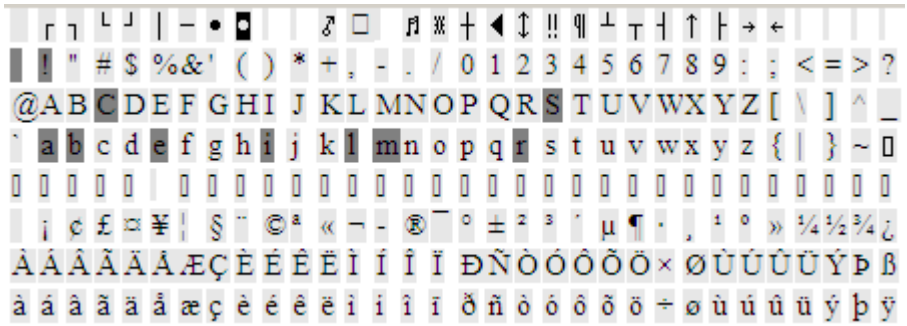


Mark the emojis with the mouse

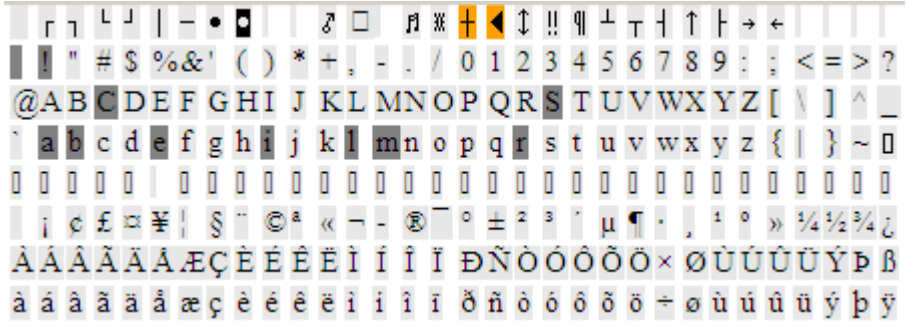


Press **Move Selected Symbols** to open the Move Symbols dialog box:

The Code Points already occupied by the font are highlighted in grey:

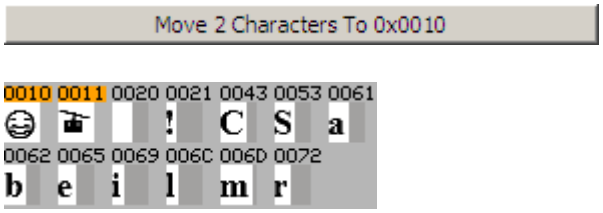


Select character 0x0010 with the mouse:



The future addresses of the symbols highlighted in green in the font window are highlighted in orange.

Press the **Move** button:



The smileys now have a new address, but the origin is still known:

The emojis are now represented by whatever glyphs the Windows font designer has placed at the first two modem control characters:

```
wchar_t * szSmiley={L"Smiley + !"};
wchar_t * szCable={L"CableCar ◀"};
```

But the output on the target display will still be the same:

```
wchar_t * szSmiley={L"Smiley  !"};
wchar_t * szCable={L"CableCar "};
```

Q: How to Use Dither for improved Image Quality.

Dithering is a method for making images look better on low resolution displays, or simply to save memory space by using a color mode with a smaller memory footprint.

Dithering can be used with 1 Bit Black & White, 4 Bit TRGB, 8 Bit RGB, 2 Bit Color Palette, 4 Bit Color Palette, and 8 Bit Color Palette.

In this example we will reduce a 24 bpp RGB image to a 2 or 4 bpp Palette image.

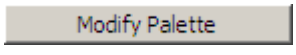
File -> Open falcon-24-bpp-rgb.png:

1: Photo with 4 bit per pixel

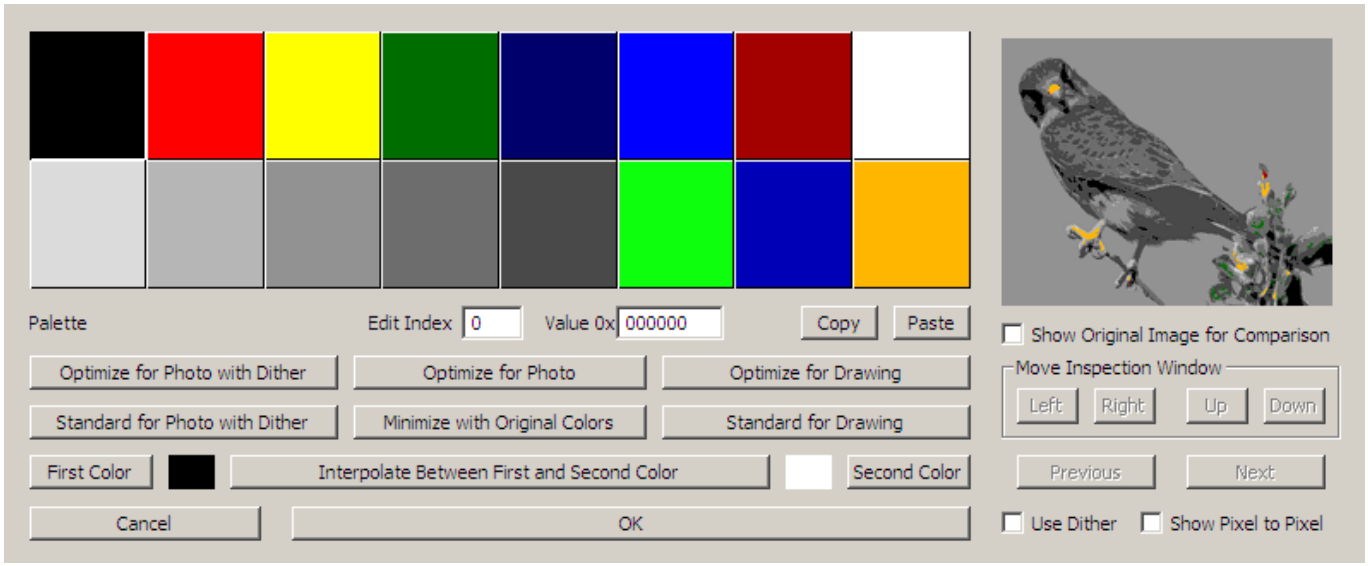
Press ***Modify Symbol***:

Select ***4 bit Color Palette***:

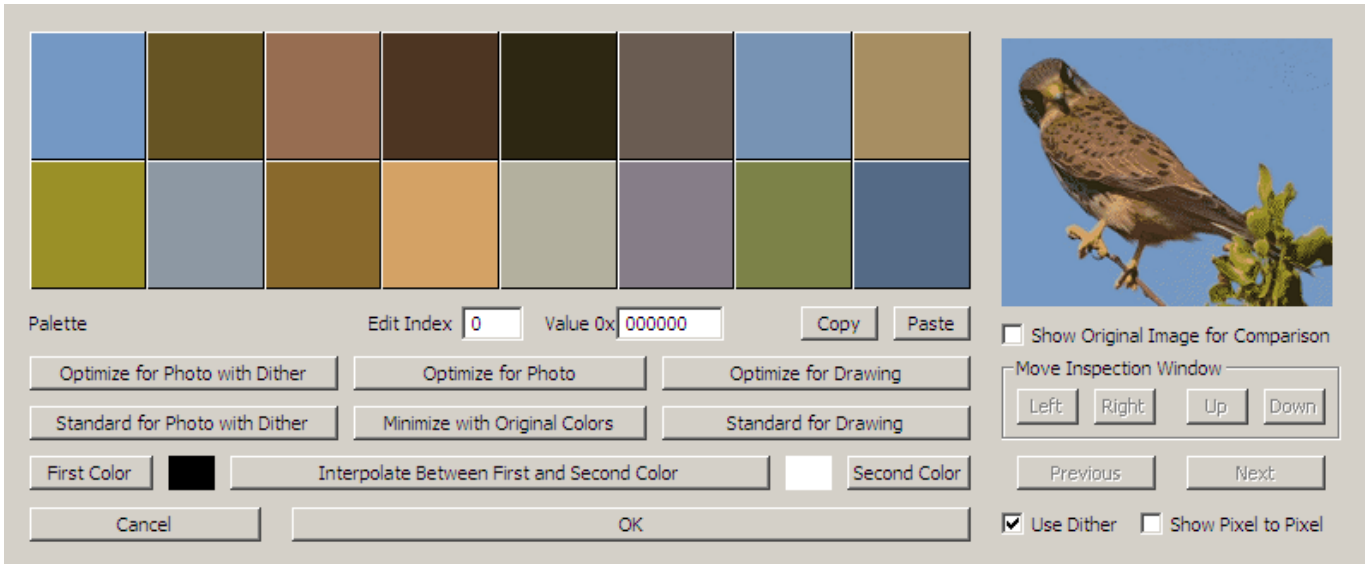
Press ***Modify Palette***:



The palette modifier opens with the standard 4 bit palette:



Press ***Optimize for Photo with Dither*** to change the palette:



Press ***OK*** to exit the palette optimizer.

Press ***OK*** to have an image with a new color mode:

Image footprint is now reduced by a factor 6 with only a small change in image quality.

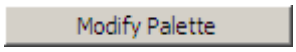
2: Photo with 2 bit per pixel

For even smaller footprint where image quality is less important than memory space use 2 Bit Color Palette:

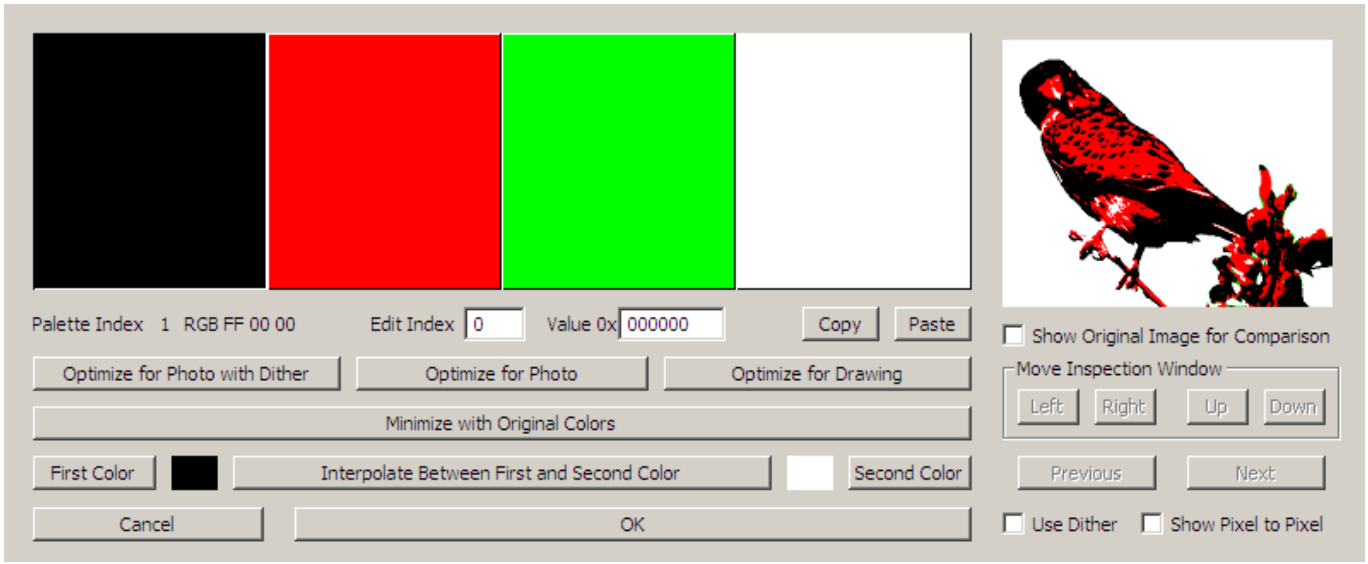
Press ***Modify Symbol***:

Select ***2 bit Color Palette***:

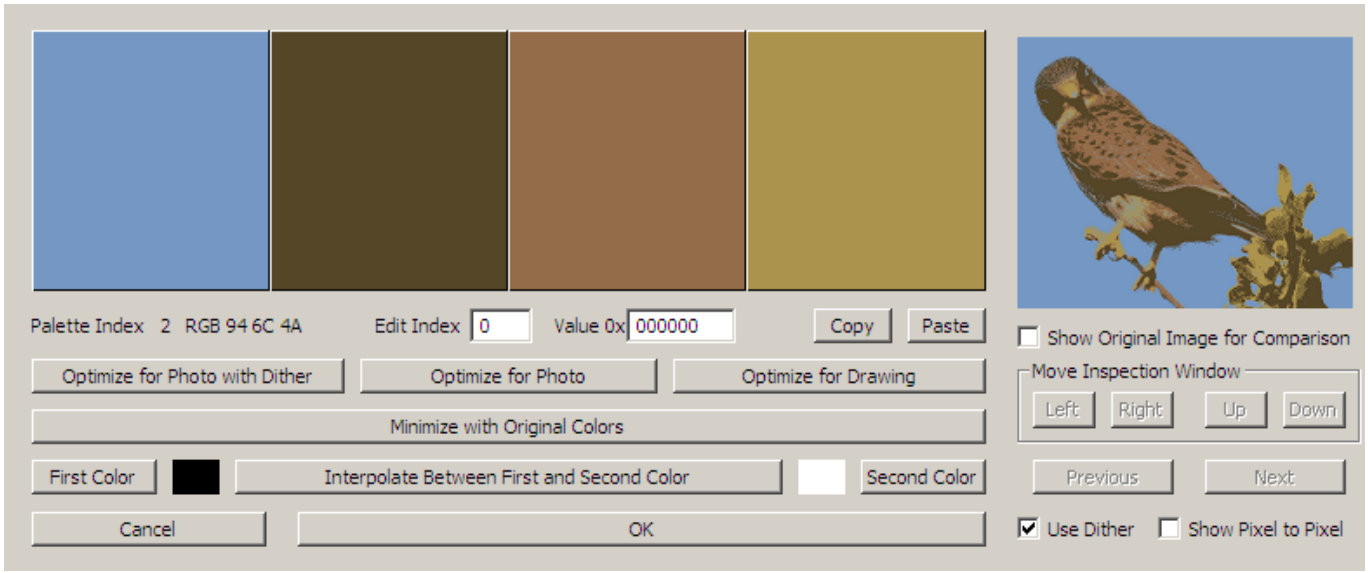
Press ***Modify Palette***:



The palette modifier opens with the standard 2 bit palette:



Press ***Optimize for Photo with Dither*** to change the palette:



Press ***OK*** to exit the palette optimizer.

Press ***OK*** to have an image with a new color mode:

R: How to Use the Text and Code Modifier

Open the file *TextTest.cpp*

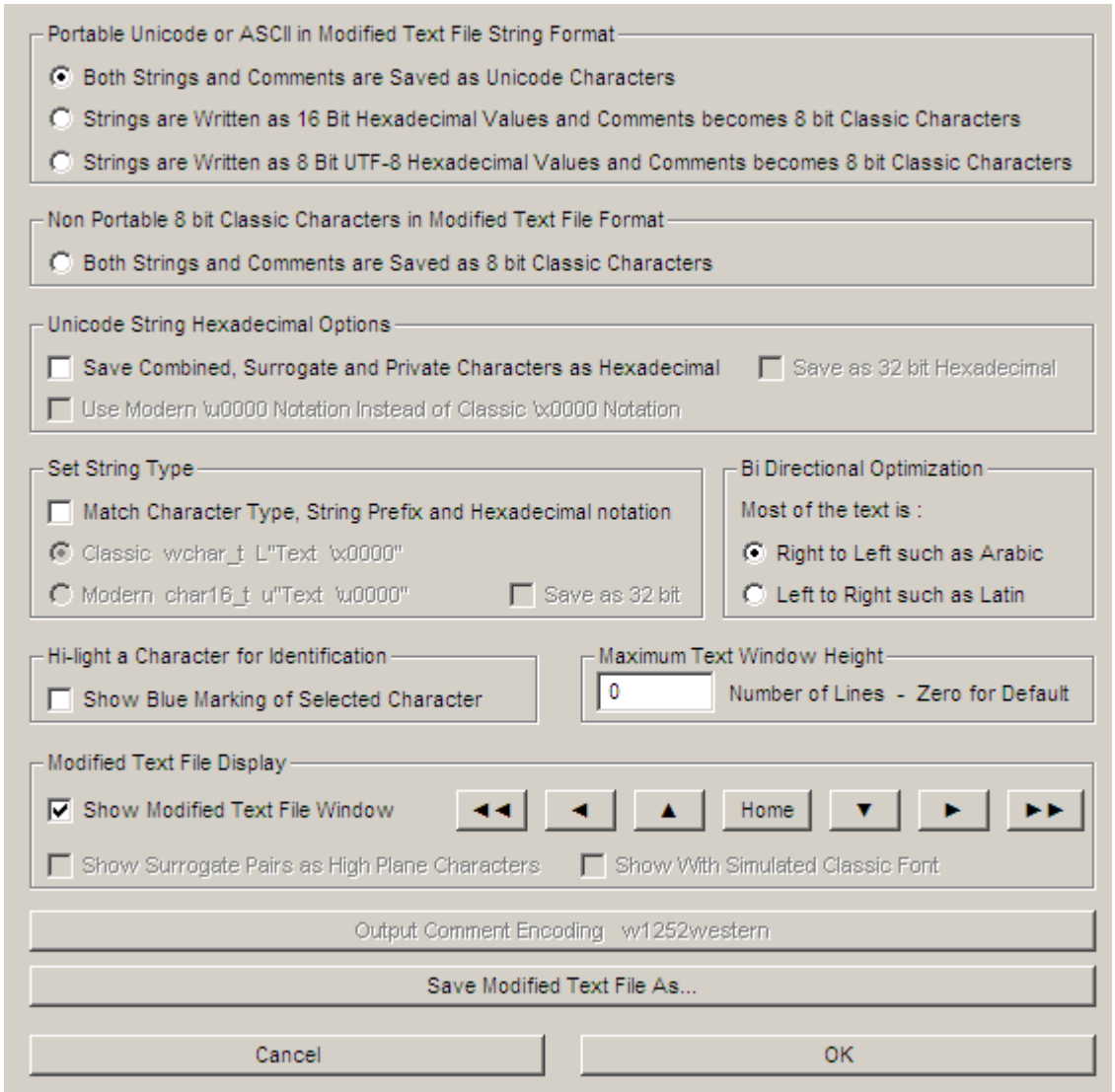
IconEdit makes a font and shows the strings with the font:

```
wchar_t * Greek={"Text Κείμενο"};
wchar_t * Marathi={"Text मजकूर"};
wchar_t * Arabic={"نص Text"};
```

The text modifier dialog box can be activated in two ways.

Use the ***Text->Save Modified Text with Autogenerated Presentation Characters as New File…***

Or press the ***Show Exported Modified Text*** button;

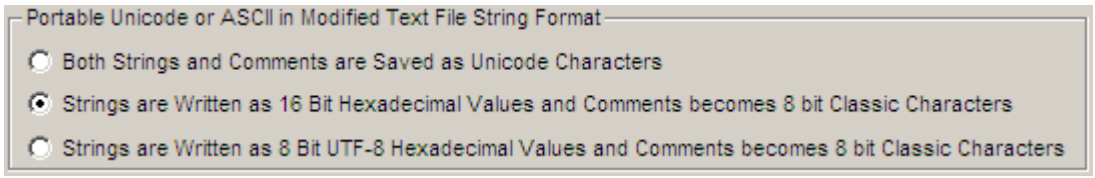


This opens the modified text display as a normal Unicode editor would show it:

```
wchar_t * Greek={"Text Κείμενο"};
wchar_t * Marathi={"Text मज□र"};
wchar_t * Arabic={"نص Text"};
```

For most modern compilers this is OK and the modified text can be saved as is.

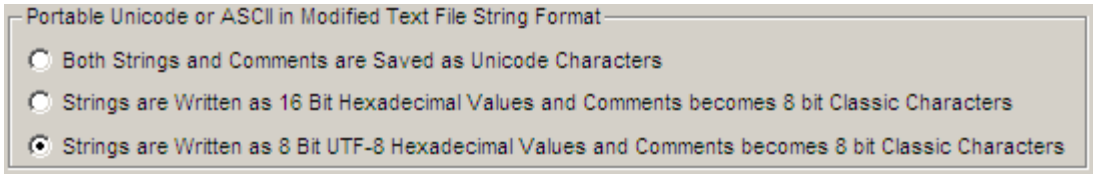
If the editor can only show 8-bit ASCII characters use the 16 bit Hexadecimal option:



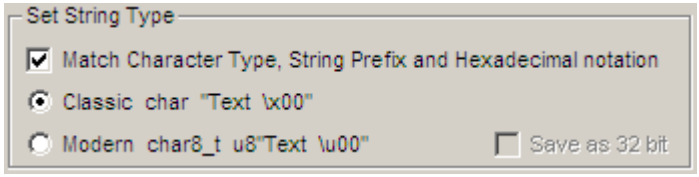

```
wchar_t * Greek={"Text \x039A\x03B5\x03AF\x03BC\x03B5\x03BD\x03BF"};
wchar_t * Marathi={"Text \x092E\x091C\xE700\x0930"};
wchar_t * Arabic={"\xFEBA\xFEE7 Text"};
```

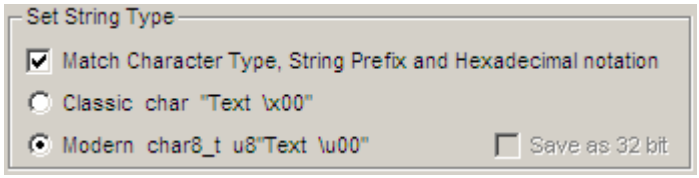
This makes the text displayable in an ASCII editor, but makes no difference to the compiler.

If the compiler can only accept 8-bit ASCII characters use the 8 bit UTF-8 Hexadecimal option:



This setting changes the string type, so activate type matching depending on the version of ‘C’ you are using:





This changes the string definitions according to the string and ‘C’ type:

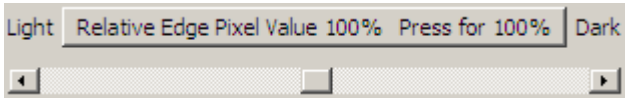
```
char * Greek={"Text \xCE\x9A\xCE\xB5\xCE\xAF\xCE\xBC\xCE\xB5\xCE\xBD\xCE\xBF"};
char * Marathi={"Text \xE0\xA4\xAE\xE0\xA4\x9C\xEE\x9C\x80\xE0\xA4\xB0"};
char * Arabic={"\xEF\xBA\xBA\xEF\xBB\xA7 Text"};

char8_t * Greek={u8"Text \xCE\x9A\xCE\xB5\xCE\xAF\xCE\xBC\xCE\xB5\xCE\xBD\xCE\xBF"};
char8_t * Marathi={u8"Text \xE0\xA4\xAE\xE0\xA4\x9C\xEE\x9C\x80\xE0\xA4\xB0"};
char8_t * Arabic={u8"\xEF\xBA\xBA\xEF\xBB\xA7 Text"};
```

S: How to Use the Edge Pixel Value Tool for Improved Quality of Fonts

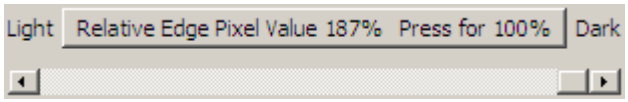
The edge pixel value tool is primarily for creating small anti-alias fonts and text from Windows fonts. The thin lines can so thin that they almost disappear and changing the edge value can improve connectivity of the glyph.

Open the Master Font Selector

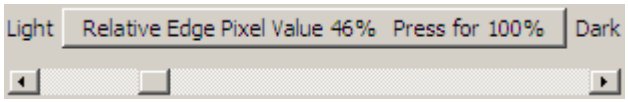
The relative edge pixel value is 100% meaning no modification:



If the thin lines tend to disappear in your fonts or tests darken the edges:

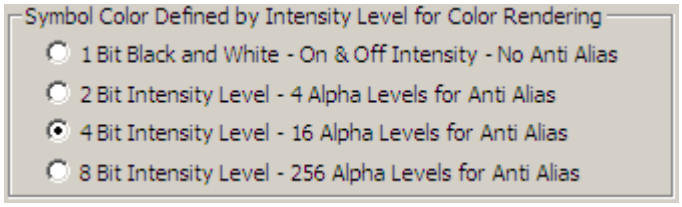


Or trim for a more extreme look, the glyph should still be fully connected:



T: How to Use Extra Smooth Anti Alias for Improved Quality of Fonts

The Extra Smooth Anti Alias is an addition to the normal Windows ClearType Anti Alias. It is primarily meant for alphabets with a handwritten look such as Arabic, Chinese, and Japanese in combination with 4 bit per pixel Intensity Level.



Normal Windows ClearType Anti Alias has problems with almost horizontal diagonal lines.

Extra Smooth Anti Alias generates a higher number of intermediate grey values, and even though they are finally reduced to 4 bit per pixel the glyph becomes smoother.

07: How to Work with Intensity Level Smoothed Characters

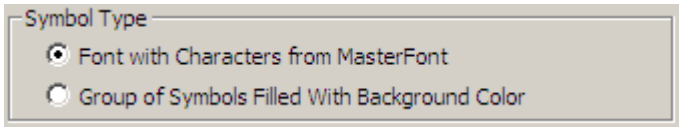
A: How to Draw Intensity Level Smoothed Characters on Symbols

This example applies only to the color version of Icon Edit.

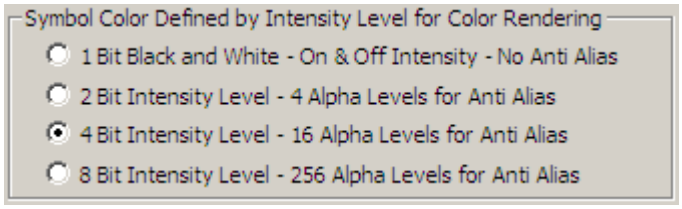
Press ***New Font or Symbol Group***:

This opens the create dialog box:

Choose how characters or symbols should be organized and shown initially:

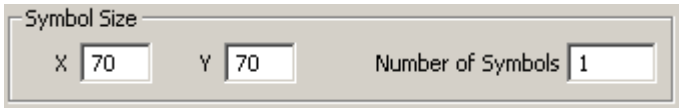


Choose a color format:



Choose the drawing colors:

One symbol:



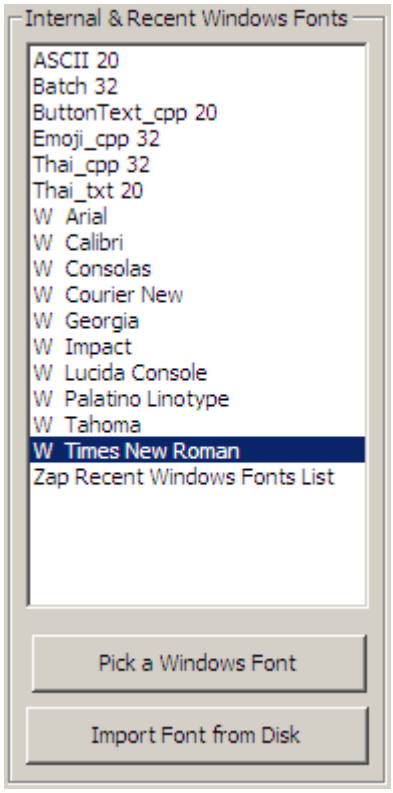
Press OK.

The palette shows 16 intensity levels, the first is the tool color marked by being pressed down, and the last is the background color marked with a circle:

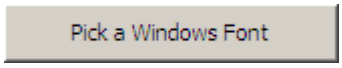
The intensity levels are also called alpha levels when the characters rendered on a background image. Black corresponds to the alpha value for fully opaque and white to fully transparent

To select a master font press ***Master Font*** in the main tool bar:

First, select a Windows font either from the Recent list or pick a new Windows font:



If the recent fonts are not ideal, pick another Windows font as this example shows:



This opens the Windows font picker, the appearance of which is depends greatly on the Windows version, Windows native language, what kind of foreign language support is installed, and any additional non-Windows fonts added later.

Fill in the 3 top fields to choose for example Times New Roman, Bold Italics and 32. The last field is height of the whole Character including top and bottom white space. This will be the height of the text frame in pixels.

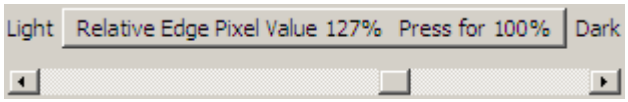
Times New Roman Bold Italics 32

Press OK.

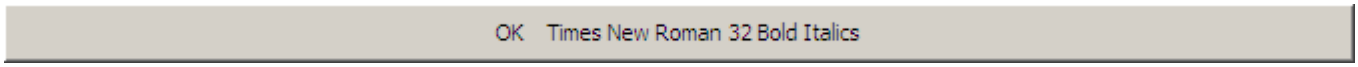
This is basically a grey tone vector font, and the conversion to black & white or to intensity level has to be fine trimmed.

Adjust the thickness of the edge of the character until the connectivity is OK.

Characters with diagonal lines like / @ A W are usually the most critical.



Press OK



Characters can be written as a text in a frame where the whole text frame can be moved around by the mouse.

To activate smooth characters right click the ***Write Text Line from Keyboard*** button:

It turns white to indicate anti alias smoothing:

The little arrow indicates that the frame can be moved by the mouse.

The whole text is considered one object, and it is redrawn each time the text is changed by adding or removing a character. Changing tool color or font affects the whole text next time it is redrawn after a change of the text, or if the frame is moved.

Write the text “Grey” on the keyboard:

Choose ***Draw Pixels*** or press ***Write Text Line from Keyboard*** again to turn off the ***Write Text Line from Keyboard*** tool:

The intensity level or alpha level formats are semitransparent by nature, and they are meant for rendering text with a color. Rendering with color on a checker board can be turned on by the ***Show CheckerBoard in Transparent Areas and Rendering Color for Characters*** button:

This changes the palette to show the intensity or alpha levels as the rendering color with transparency levels at the bottom of each button:

The checker board can be changed with the ***Color and Size of Transparent Background CheckerBoard*** button:

The rendering color can be changed by right click on the tool color button.

Warning: *Right or left click on any of the other palette buttons changes the intensity or alpha level of the whole text to the new tool color.*

B: How to Convert Intensity Level Smoothed Symbols to Semi Transparency

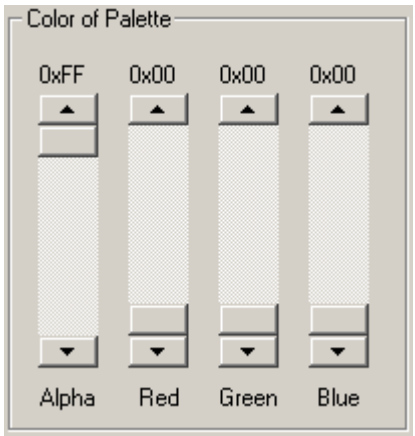
This example uses the symbol and settings from the previous example.

To convert the intensity level anti alias data format to semi transparency there are two possibilities either turn off rendering with color on a checker board by the ***Show CheckerBoard in Transparent Areas and Rendering Color for Characters*** button to get tool color characters or turn it on to get the rendering color from the intensity level color format:

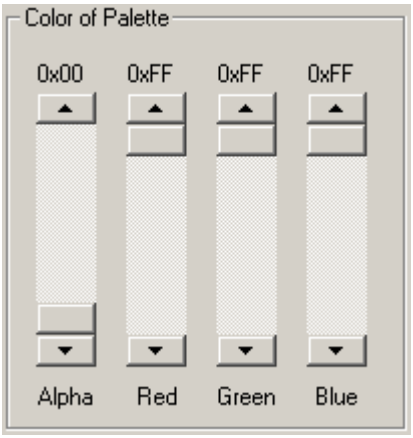
Then press the ***Modify*** button:

Change color format:

Set tool and background colors to opaque black and transparent white:



Press OK.



Press OK.

Press OK.

The text does not change, but it is now black with a transparent white background.

Use the + on the ***Color and Size of Transparent Background Checkerboard*** button to verify the transparency of the background:

The color of the characters can now be changed with ***Floodfill*** functions.

The appearance of the transparent area colors and checker size can be changed with the ***Transparency Background*** button:


Grep

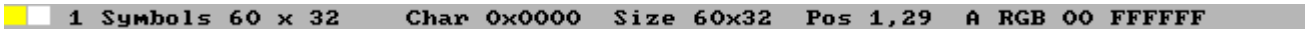
C: How to Convert Semi-transparent Symbols to Intensity Level Anti Alias

This example uses the settings and symbol from the previous example.

When colored glyphs are converted to grey tones it is normally done in accordance with to their luminosity:

But intensity level anti-alias is a transparency data format based on visible figures on a transparent background. This conversion is achieved by converting the transparency of the 32 bit ARGB symbol directly to the target intensity level or alpha levels without using the RGB basic color value. Fully opaque is converted to black and fully transparent is converted to white.

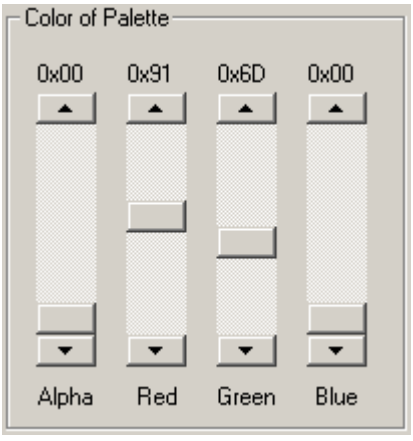
This means that the background for the ARGB figures must be fully transparent with an alpha value of 0 as shown:



A RGB 00 FFFFFF.

If this is not the case the background should be changed to a new fully transparent color:

Right click an unused color button in the palette and make it fully transparent:

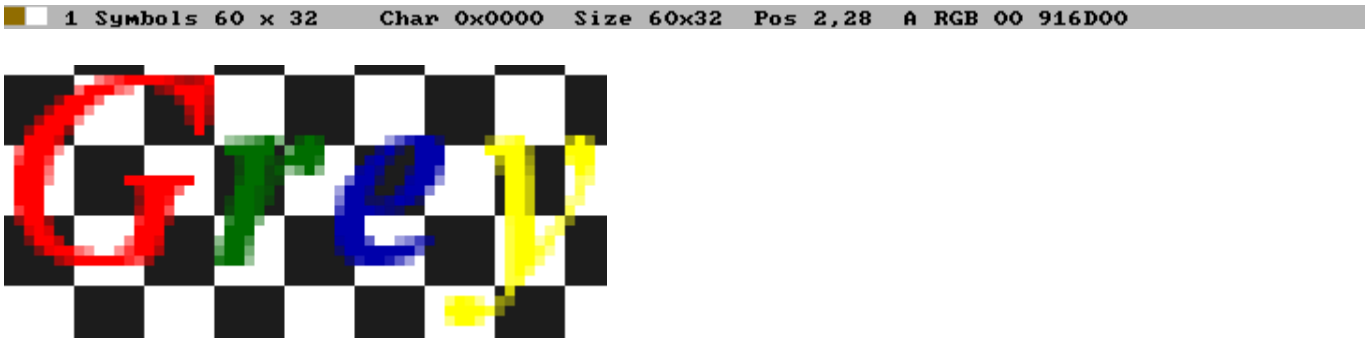


Press OK.

The lower part of the button shows the degree of transparency by mixing with white and black:

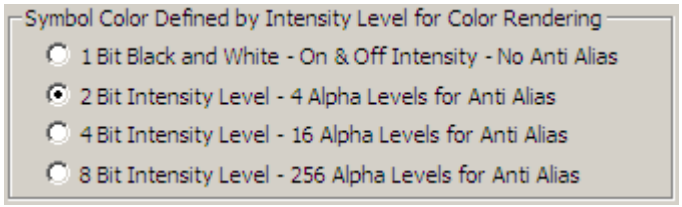
Right click the flood fill button twice to get the ***Local Substitute*** and then left click ***Local Substitute*** to activate it:

Substitute the background by clicking anywhere on the background color:

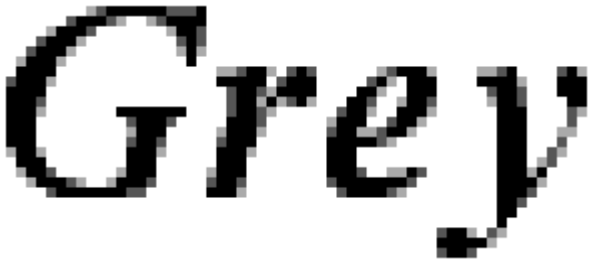


Press ***Modify Transparent Color***:

Select an Intensity Level 2-bit as this resolution is usually enough:



Press OK.



The symbol or text is now intensity level or alpha level for anti alias.

D: How to Convert Color Symbols to High Contrast Intensity Level Anti Alias

This example applies only to the color version of IconEdit.

This example will show how to get a high contrast image from a low contrast original design. This is obtained through a series of data format conversions that can be done either automatically or single-stepped for increased control.

Warning: It is important that the original design is either saved uncompressed such as a *Name.bmp* file or saved as a lossless compression such as a *Name.png* file to avoid artefacts such as multicoloured halos typical of lossy compressions such as *Name.jpg*:

Lossless compression *Name.png*, clear edges:

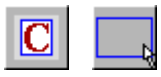
Lossy compression *Name.jpg*, unsharp edges and wrong colors:

Press the ***File Open*** button in the Main tool bar:

Open an existing image, for example *Logo.png*.

Mark the logo with the ***Edit Indside Blue Frame*** tool:

Press ***Crop the Picture*** to remove the status-bar and turn off the ***Edit Indside Blue Frame*** tool:



Set tool color to the text color with the ***Pick New Tool Color*** tool.

Right click the ***Pick New Tool Color*** tool to change to the ***Pick New Background Color*** tool.

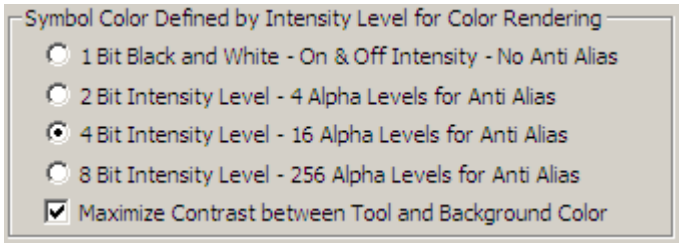
Set background color to the background color of the symbol with the ***Pick New Background Color*** tool.

1: Automatic conversion

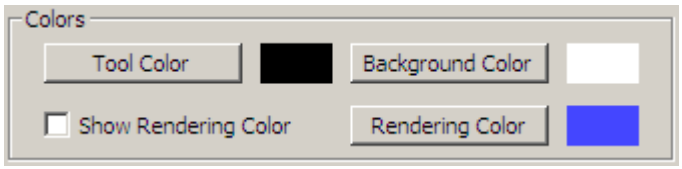
Change color mode in the ***Modify Color Symbol*** dialog box:

Change to intensity level.

Activate Maximum Contrast:



Make sure tool and background color are different after conversion to grey:



Press OK.

Your High Contrast Logo is ready for saving....

Press the ***Save All As…*** button in the Main tool bar to save the pixel data in the *Logo.c* file in the proper directory.

2: Single step conversion and rendering test

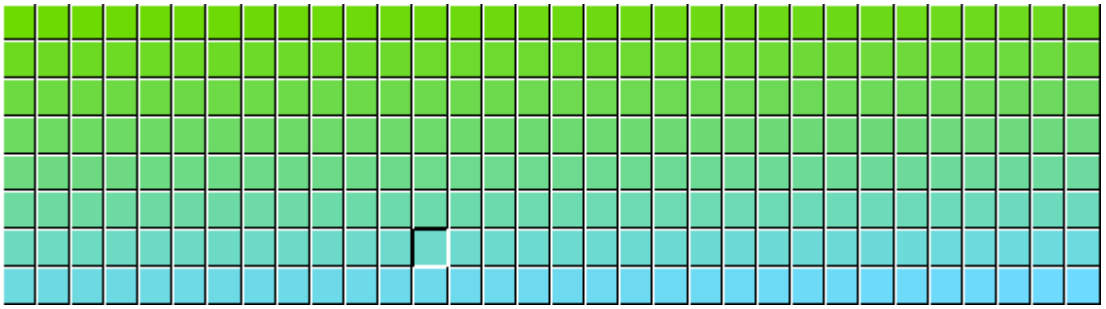
Change color mode in the ***Modify Color Symbol*** dialog box:

Change to color palette.

Change the palette.

The colors for interpolation are already set to **Tool Color** and **Background Color**:

Press ***Interpolate***:



Press Ok for palette.

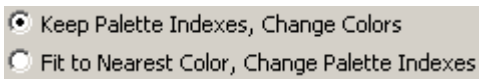
Press Ok for color mode.

The symbol looks the same but is now an 8 bit color symbol.

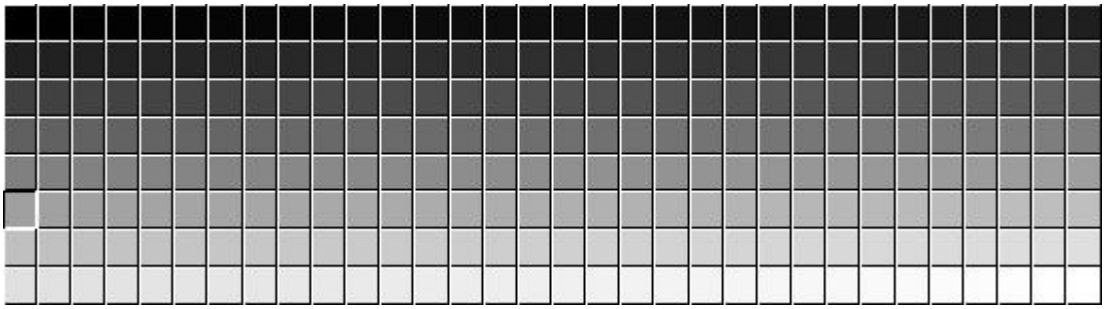
Change color mode in the ***Modify Color Palette Symbol*** dialog box:

Change to grey palette.

Set conversion type to keep indexes.



This is what generates the high contrast by substituting the green to blue palette with a black to white palette.

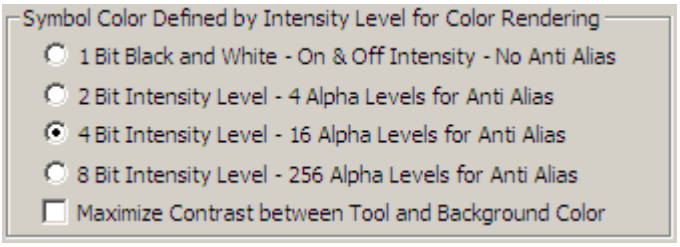


Press Ok for color mode.

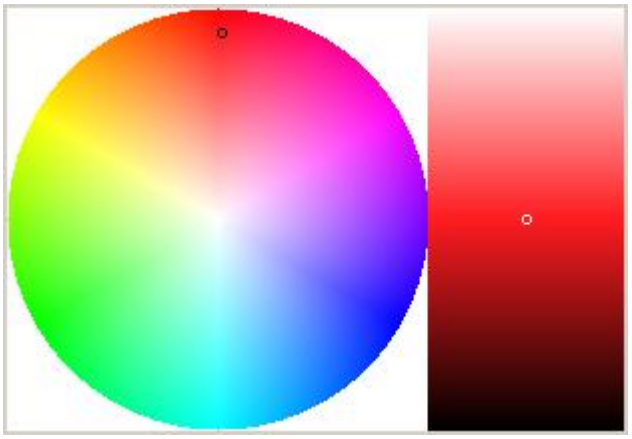
The symbol is now high contrast grey palette.

Change color mode in the ***Modify Grey Palette Symbol*** dialog box:

Change to intensity level.



Choose a rendering color.



Press Ok for color.

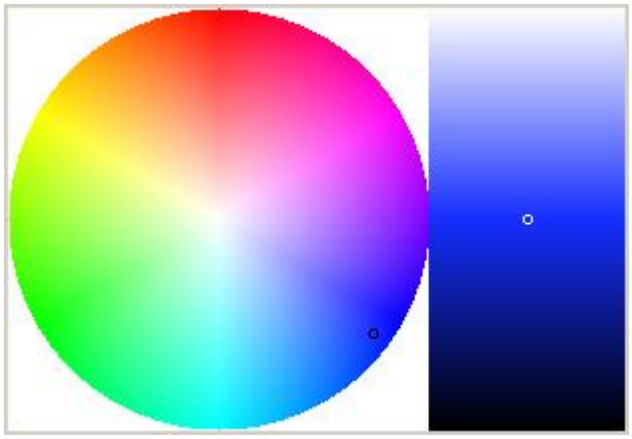
Press Ok for color mode.

The symbol is now high contrast intensity level.

DanMagic

And it can be rendered in any color with the ***Show CheckerBoard*** option.

The rendering color can be changed by right click on one of the palette buttons.



Your High Contrast Logo is ready for saving....

Press the ***Save All As…*** button in the Main tool bar to save the pixel data in the *Logo.c* file in the proper directory.

E: How to Extract Symbols from an Original Screen Design for Colored Animation

This example applies in full to the color version of IconEdit, extracting animated icons also applies to the black & white version.

This example will show how to get high contrast animated icons from a low contrast original design.

Warning: It is important that the original design for the screen has the same size as the target screen.

Press the ***File Open*** button in the Main tool bar:

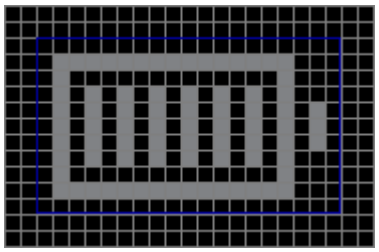
Open an existing image, for example *Logo.png*.

Make it bigger with the ***Change Zoom*** button:

Turn on ***Show Grey Pixel Grid***:

1: Symbol with sharp edges and only two colors

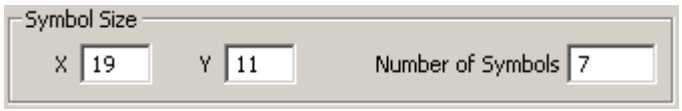
Mark the battery symbol with the ***Edit Indside Blue Frame*** tool:

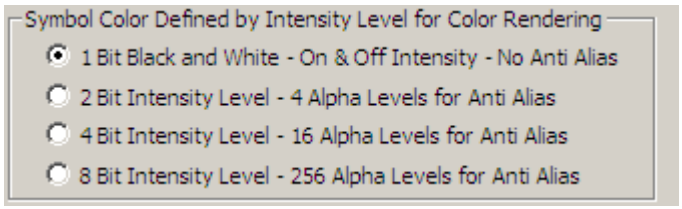


Copy to ClipBoard with.

Press ***Paste from ClipBoard as New Group***.

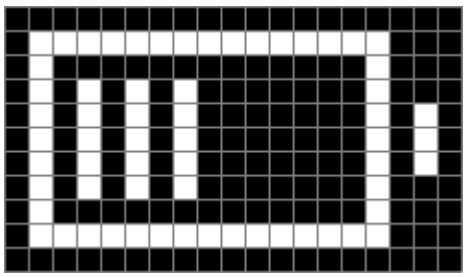
The battery indicator has 7 possible states and only 2 colors, so choose 7 black & white symbols:



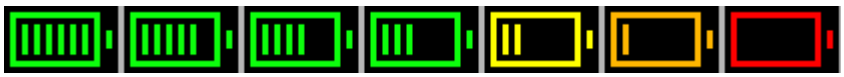


Press OK.

Right click one icon at the time to change the number of charge lines to black with the ***Surface Flood Fill*** tool:



Please note that even though it is called a black & white the format is actually a 2 color format and can be rendered on the target display in any color the display is capable of. This means that the deteriorating charge condition of the battery can be displayed as a series of icons with warning colors:

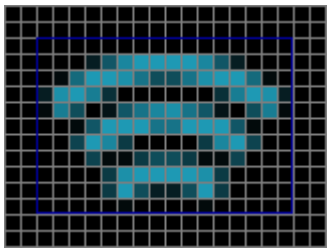


Your group is ready for saving....

Press the ***Save All As…*** button in the Main tool bar to save the pixel data in the *Battery_BW_7.c* file in the proper directory.

2: Symbol with smooth edges and anti aliasing by interpolating between two colors

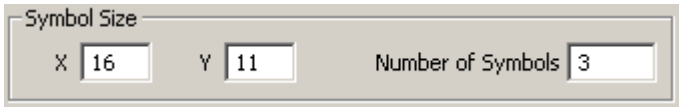
Mark the bluetooth symbol with the ***Edit Indside Blue Frame*** tool:



Copy to ClipBoard with

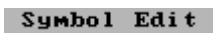
Press ***Paste from ClipBoard as New Group***.

The bluetooth indicator has 3 possible states and a range of interpolated colors, so choose 3 symbols and keep the color mode until the next step:



Press OK.

Change to Symbol Edit



Set tool color to the brightest color on the symbol with the ***Pick New Tool Color*** tool.

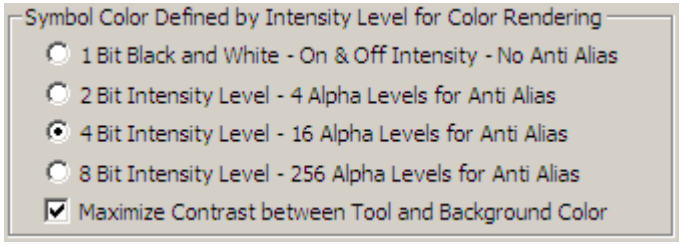
Right click the ***Pick New Tool Color*** tool to change to the ***Pick New Background Color*** tool.

Set background color to the background color of the symbol with the ***Pick New Background Color*** tool.

Change color mode in the ***Modify Color Symbol*** dialog box.

Change to intensity level.

Activate Maximum Contrast



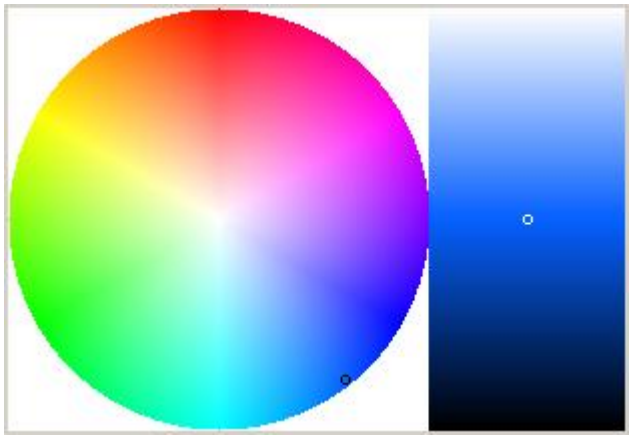
Press OK.

Remove the surplus arcs in some of the icons to show different signal strength.

Choose the white tool color button with **Alpha 0** and the ***Solid Rectangle*** tool to overwrite the surplus arcs.

The icons are now high contrast intensity levels and can be rendered in any color with the ***Show CheckerBoard*** option.

The rendering color can be changed by right click on one of the palette buttons.



Your group is ready for saving....

Press the ***Save All As…*** button in the Main tool bar to save the pixel data in the *Bluetooth_g4_3.c* file in the proper directory.

08: How to Draw Symbols

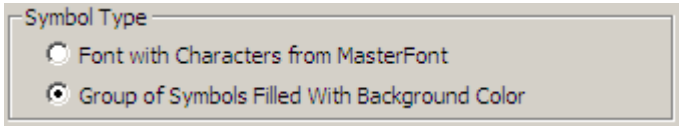
A: How to Draw a Group of Battery Indicator Symbols

This example applies only to the color version of IconEdit.

To make a new group press the ***New Font or Symbol Group*** button in the Main Toolbar:

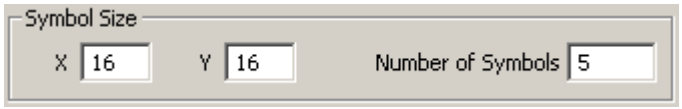
This opens the create dialog box:

Choose how characters or symbols should be organized and shown initially:

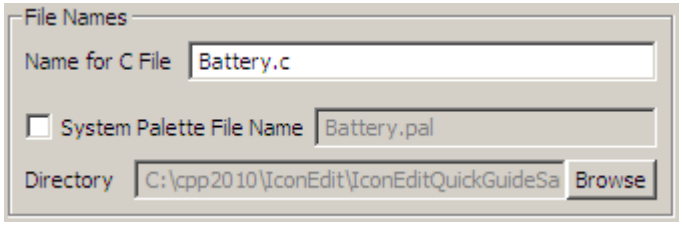


Choose a color format:

Choose size and number:

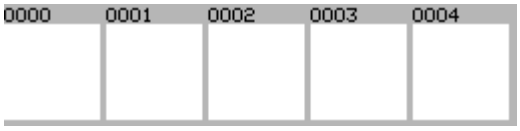


Give it a name:



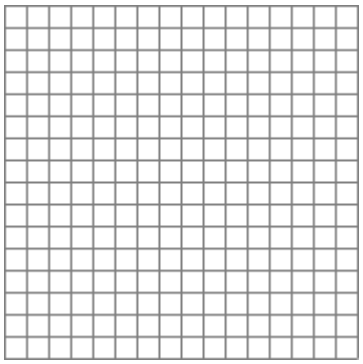
Press OK.

The group now looks like this:

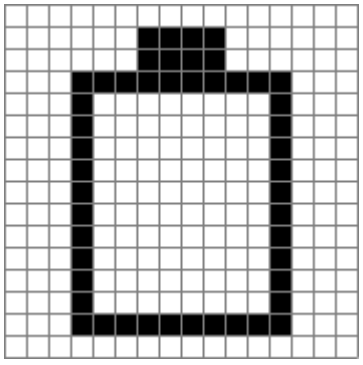


Right click the first symbol to change to pixel edit mode:

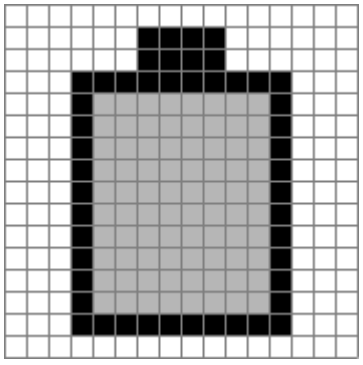
Change to a smaller size with the ***Change Zoom*** button:



Choose ***Hollow Rectangle*** and draw the battery outline:

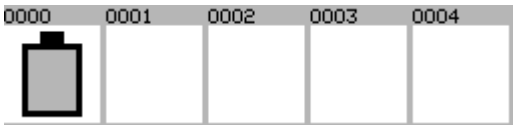


Choose a grey tone and the ***Surface Flood Fill*** tool and fill the battery:

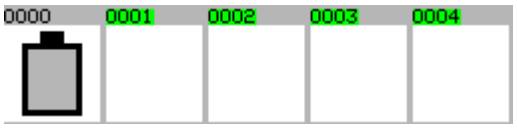


Copy to ClipBoard with

Change to Group edit with a right click on the battery:

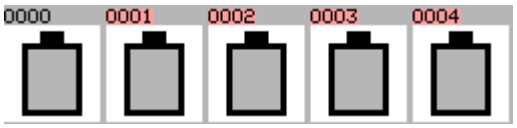


Mark symbol 0001 to 0004 with a left click on 0001 and Shift-click on 0004:

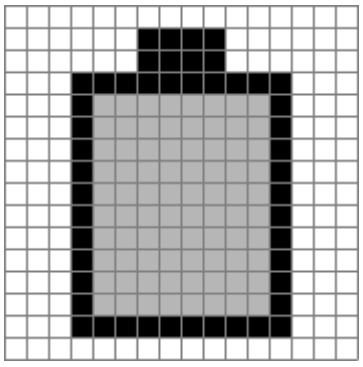


Paste symbol 0000 with ***Paste from ClipBoard***:

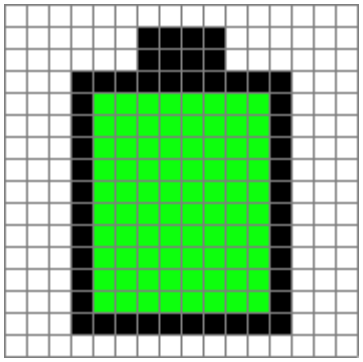
Choose **Overwrite Marked Symbols**:



Right click on symbol 0000 to change to pixel edit mode:

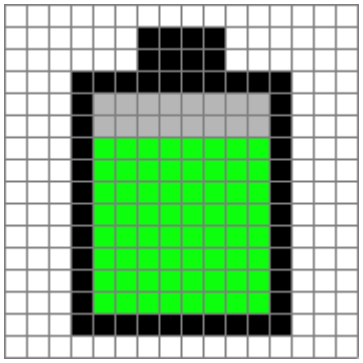


Choose a green color and fill the center of the battery:



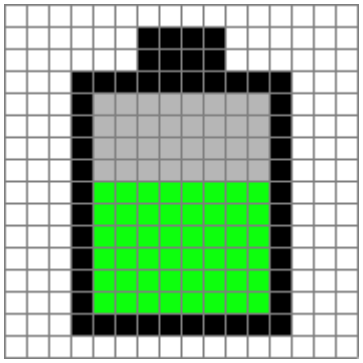
Click on symbol 0001 to change to the next battery.

Choose ***Solid Rectangle*** and draw 80% green:



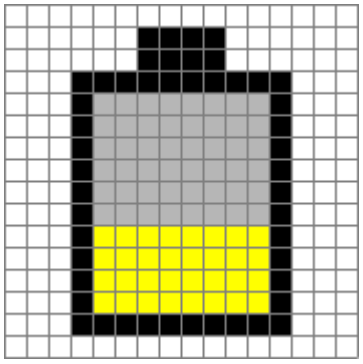
Click on symbol 0002 to change to the next battery.

Draw 60% green:



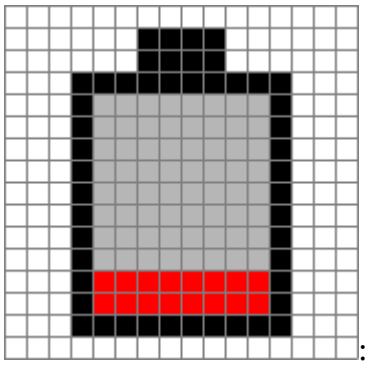
Click on symbol 0003 to change to the next battery.

Choose a yellow color and draw 40% yellow:

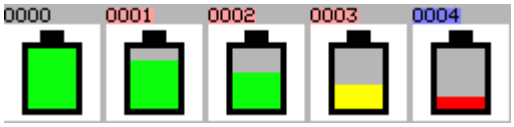


Click on symbol 0004 to change to the next battery.

Choose a red color and draw 20% red

 :

The battery indicators now look like this:



Your group is ready for saving....

Press the ***Save All As…*** button in the Main tool bar to save the pixel data in the *Battery.c* file in the proper directory.

1: How to Hide Unused Space with Single Color On Off Transparency

This example is a continuation and uses the group and settings of the previous example.

If the battery indicators should be displayed on a background image, the unused white space around the battery symbols should be transparent.

Press the Modify Symbol button:

Activate single color on off transparency, and choose white as the transparent color:

Press OK.

There is no apparent change, but activating ***Show CheckerBoard in Transparent Areas*** will reveal the change:

The palette also shows that the background is now transparent. The lower half of the transparent color's palette buttons is displayed mixed with white and black. In this case it is the white background:

Your group is ready for saving....

Press the ***Save All As…*** button in the Main tool bar to save the pixel data in the *Battery.c* file in the proper directory.

B: How to Use On Off Transparency to Draw Buttons with Round Corners

This example applies only to the color version of IconEdit.

On off transparency is a method for drawing symbols of any shape without having to use the 32 bit per pixel semitransparent data format. One of the possible colors in the data format is marked as fully transparent, and can be used to mark the unused pixels around the symbol to make them invisible. It works with all non transparent data formats from 24 bit RGB color to 2 bit palette and can reduce memory consumption scientifically.

The use of on off transparency in combination with tools with anti aliasing or smoothing should be done with care. Smoothing emulates partly drawn pixels by mixing the tool color with the previous color for each partly drawn pixel, if the pixel already has the transparent color this is the color the tool color is mixed with thereby creating a pixel with a fully opaque intermediate color.

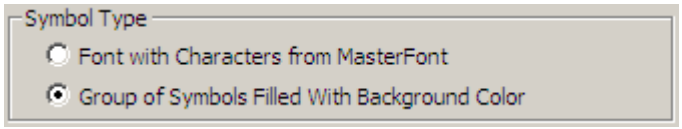
Smoothed edges of the symbol will appear correct if the target system has only one background color and the transparent color is chosen as a color close to that of the target background. If the symbol has to be used on several different backgrounds or on background pictures the smoothing will only appear correct on those areas where the background color is close to the transparent color. Therefore smoothing of symbol edges is only recommended if the target system has a single background color.

In this example we will make 3 buttons, OK for use with several backgrounds, Go and No for use on a light grey toolbar.

To make a new symbol press the ***New Font or Symbol Group*** button in the Main Toolbar:

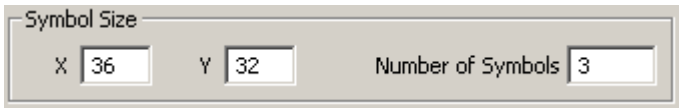
This opens the create dialog box:

Choose how characters or symbols should be organized and shown initially:



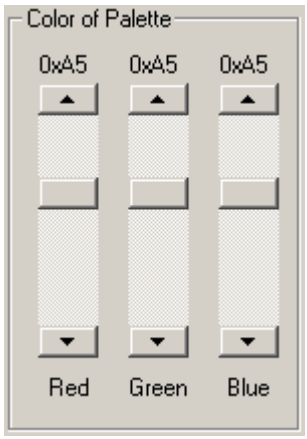
Choose a color format:

Choose size and number:



Set background and transparent color to match the target system, here light grey with the value 0xA5:

Press **Background Color**.



Press OK.

Use on off transparency:

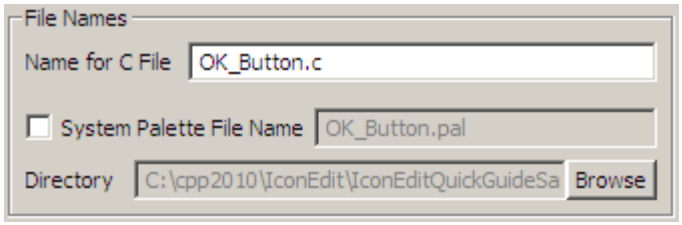
Press **Transparent Color**.

Press OK.

This automatically sets the transparent color to the background color.

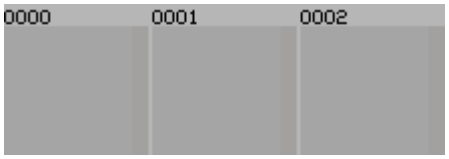
This is not a permanent link, both background and transparent can be changed individually afterwards.

Give the buttons a name:

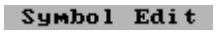


Press OK.

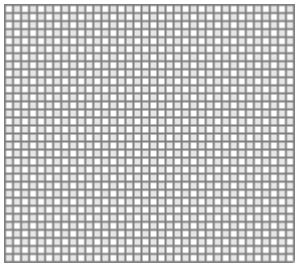
The group now looks like this:



Change to ***Symbol Edit***:



Use ***Show CheckerBoard*** to confirm that the whole symbol is transparent:

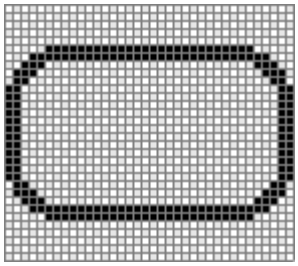


Set ***Line Width*** to 2:

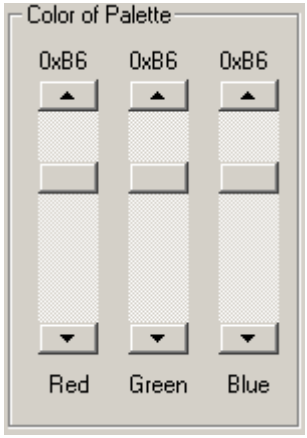
Press 2.

Choose ***Hollow Round Rectangle***:

Start from top left, move to bottom right, hold left button and press right button too, move back towards top left until the button corners have the right size:



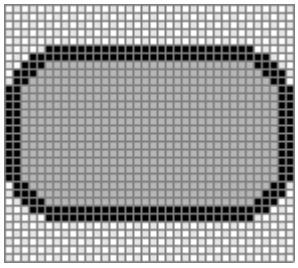
Right click a palette button and set the color to grey 0xB6:



Press OK.

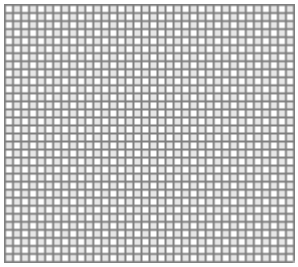
The palette button is now grey, and it is the chosen tool color:

Choose ***Surface Flood Fill***, and fill the centre of the rounded button:



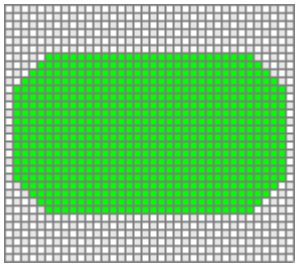
Click on ***Symbol 0001*** to move it to the symbol edit window.

0001



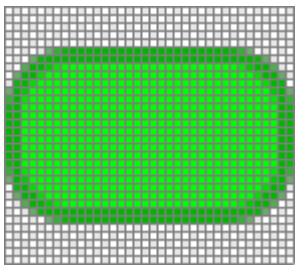
Smoothing has to be done on the right background for the Go button, so draw the centre first:

Choose bright green and ***Solid Round Rectangle*** and draw the centre:

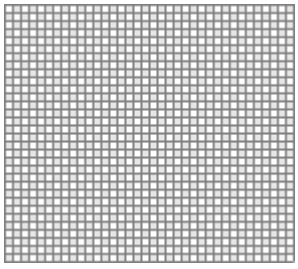


Choose a darker green for frame and activate ***Hollow Round Rectangle*** with smoothing by a right click for the smoothing:

Draw the frame:

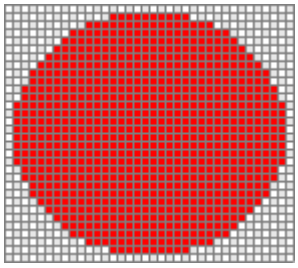


Click on ***Symbol 0002*** to move it to the symbol edit window.



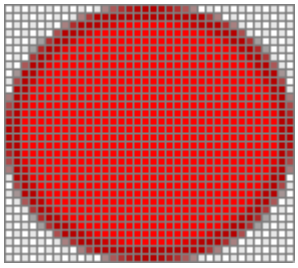
Smoothing has to be done on the right background for the No button, so draw the centre first:

Choose bright red and left click ***Solid Ellipse*** then draw the centre:

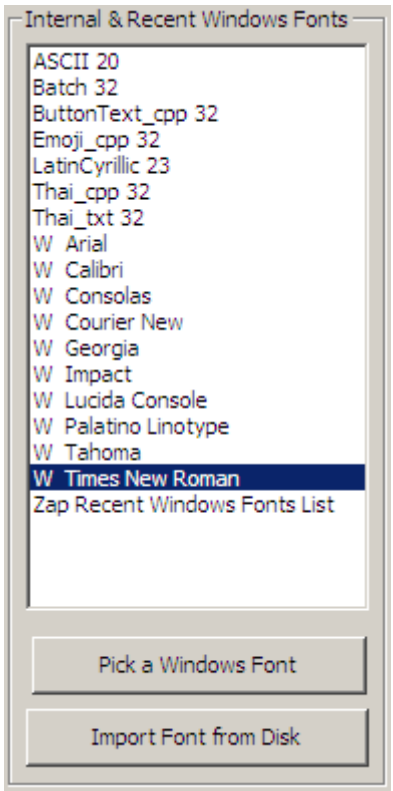


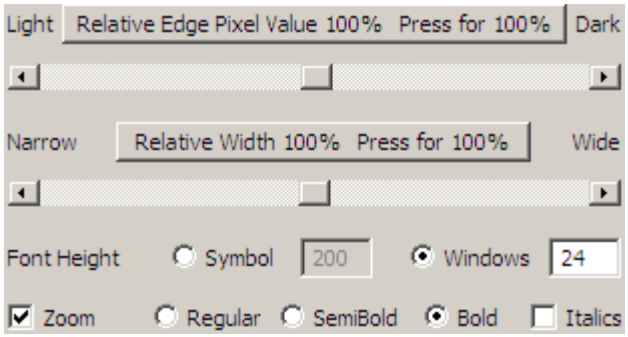
Choose a darker red for frame and activate ***Hollow Ellipse*** with smoothing by a right click for smoothing:

Draw the frame:



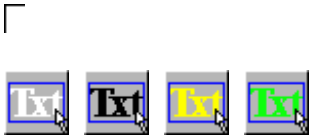
In ***MasterFont*** choose Times New Roman, 24 high, Neutral Darkness and Bold:





Press OK.

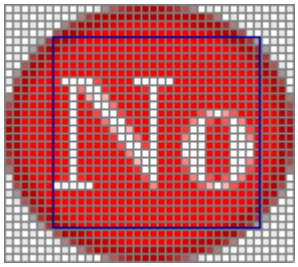
Choose white and right click ***Write Text Line*** in frame to activate smoothing or semi transparency:



Then activate the function by a left click.

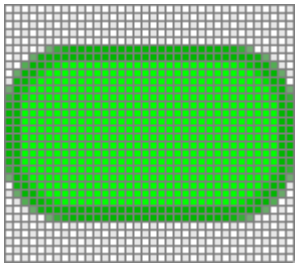
This automatically changes the ***Master Font Selector*** to the ***Virtual Keyboard***:

Write No and move it in place with the mouse:



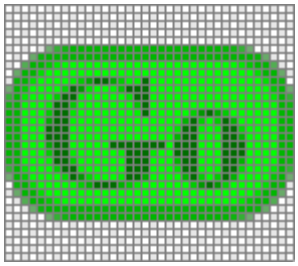
Click on ***Symbol 0001*** to move it to the symbol edit window.

0001

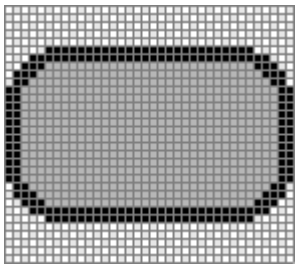


Choose dark green:

Write Go and move it in place with the mouse:

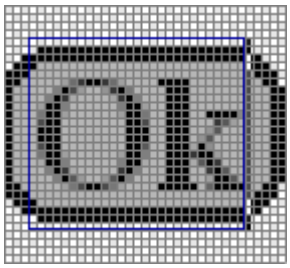


Click on ***Symbol 0000*** to move it to the symbol edit window.



Choose black:

Write OK and move it in place with the mouse:



The whole group now looks like this:

Your group is ready for saving....

Press the ***Save All As…*** button in the Main tool bar to save the pixel data in the *Ok_button.c* file in the proper directory.

1: How to Reduce On Off Transparency Memory Consumption

This example is a continuation and uses the group and settings of the previous example.

The button symbols do not really need 24 bit RGB, this example will show how to convert the symbols to 8 or 4 bit palette.

First stop the ***Write Text Line*** function by selecting ***Draw Pixels*** function:

Reduction to 8 bit palette:

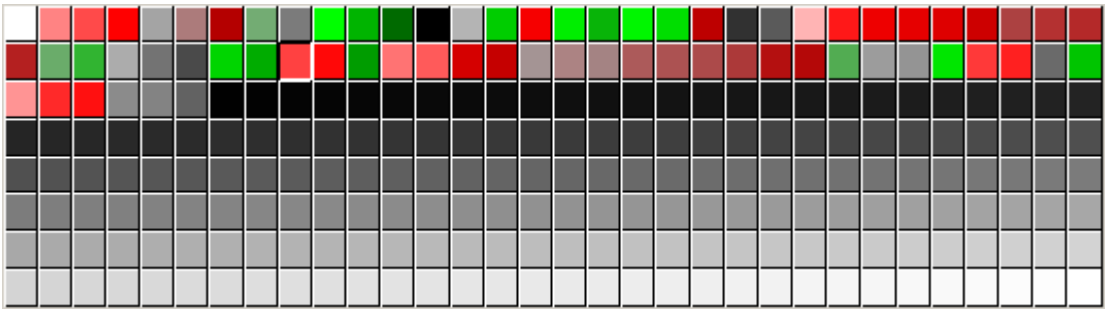
Press ***Modify Color Symbol***:

Choose 8 Bit Color Palette:

Press Modify Palette:

Press Optimize:

The optimizer reduces all colors to 5+5+5 bit resolution, and selects the most important colors, if there are only few important colors, the rest of the palette is filled with grey:



Press OK:

Press OK:

The symbols are now reduced from 16777216 possible colors to 256 possible colors.

It can easily happen that the On Off Transparency color and some nearby colors now are represented by the same color in the Palette. In this case 2 of the smoothing pixels on the Ok button are now transparent.

To remove the unwanted transparency, choose a color close to the On Off Transparency color, and use the ***Draw Pixels*** function to remove them:

Reduction to 4 bit palette:

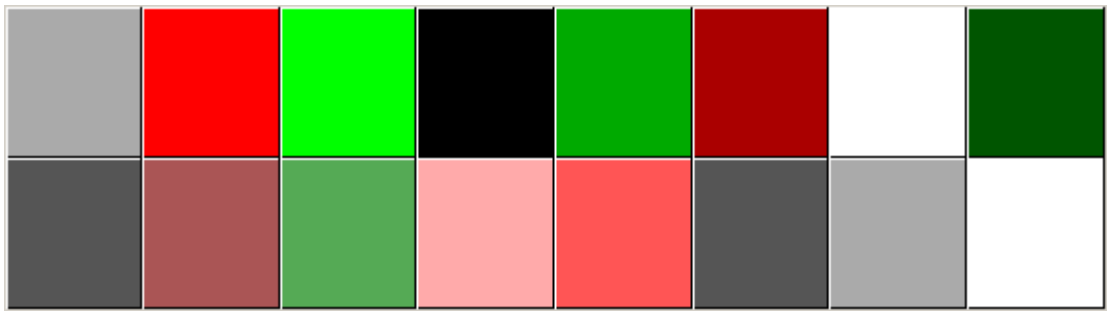
Press ***Modify Color Symbol***:

Choose 4 Bit Color Palette:

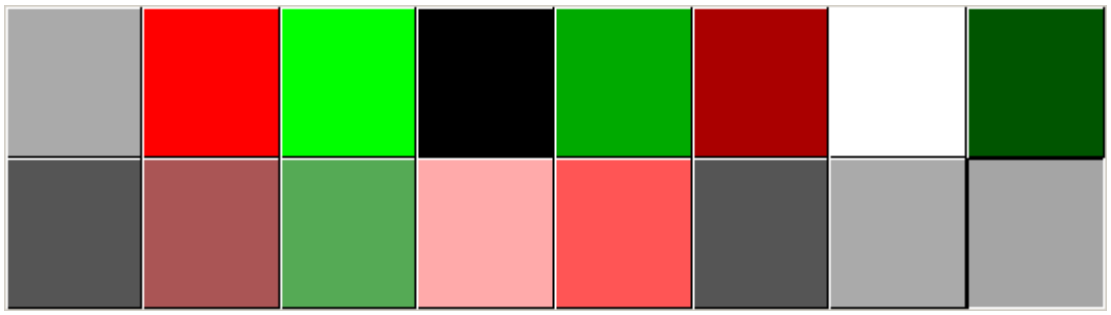
Press Modify Palette:

Press Optimize:

The optimizer reduces all colors to 2+2+2 bit resolution, and selects the most important colors, if there are only few important colors, the rest of the palette is filled with grey:



The On Off Transparency color is not represented in the palette, so enter A5A5A5 for index 15:



Press OK.

Press OK:

The symbols are now reduced from 16777216 possible colors to 16 possible colors.

It can easily happen that the On Off Transparency color and some nearby colors now are represented by the same color. In this case many of the smoothing pixels on the Ok button are now transparent.

To remove the unwanted transparency, choose a color close to the On Off Transparency color:

The transparent pixels tend to lie on a line, so use the Line Flood Fill function to remove them:

Right click ***Surface Flood Fill*** to get ***Line Flood Fill*** activate it with a left click and fill the transparent lines:

Comparison to full 32 bit ARGB transparency:

24 bit RGB 75% of 32 bit ARGB:

8 bit Palette 25% of 32 bit ARGB:

4 bit Palette 13% of 32 bit ARGB:

C: How to Add one Symbol on top of another using Semi-Transparency

This example applies only to the color version of IconEdit.

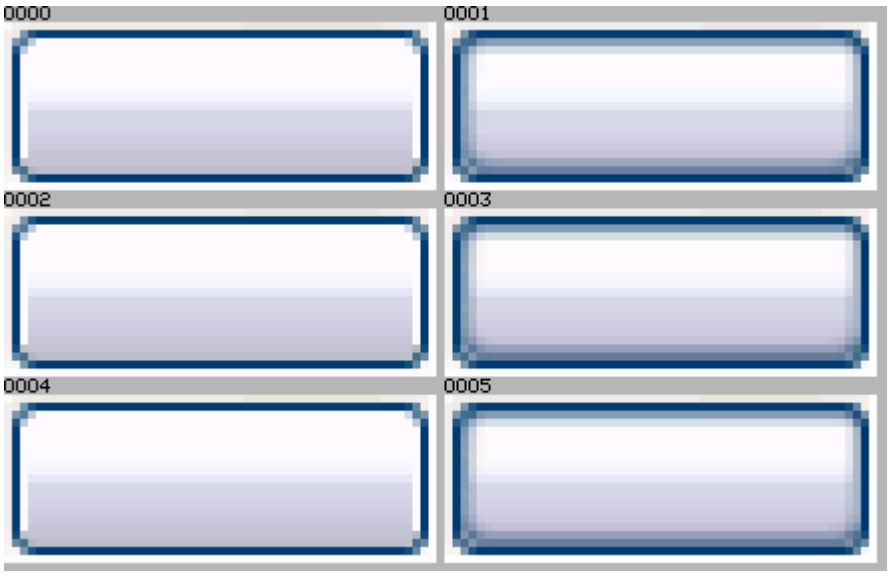
In this example, the aim is to add arrows and text to a set of shaded buttons while keeping the shading.

Drawing the buttons is done with 32 bit ARGB transparency, but the final result can be saved as normal RGB or as Palette files. If the final result is to be saved in palette data format, palette optimization would be appropriate.

Assume that the buttons already exist as *ShadedButtons.c*.

Press ***File Open***

And select *ShadedButtons.c*. They will look like this:



If the buttons are not already transparent select ***File -> Modify Existing Font or Symbol Group…*** to get the ***Modify Existing Data Set*** dialog box, and select 32 Bit ARGB:

Press OK.

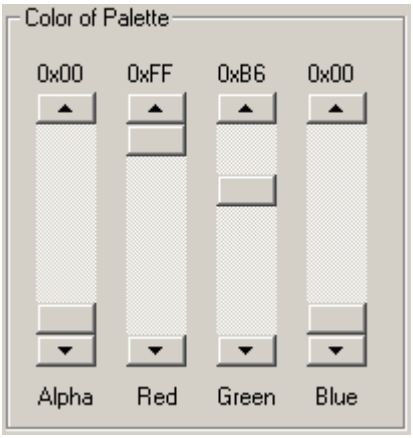
The symbol is now fully opaque with alpha at maximum value FF.

To get a working area for making an arrow use insert at the caret:

Place the caret near the end of the last button with the mouse and press ***Insert Empty Symbol at Caret*** to make symbol 0006:

Change to Symbol edit mode by right clicking on symbol 0006.

To avoid conflict with the basic color of the background later, right click a color that is not going to be used for the buttons and make it fully transparent:



Press OK:

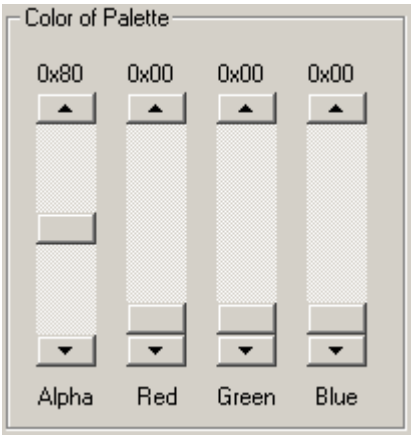
Select ***Surface Flood Fill***:

Click anywhere on the symbol to fill it with the new fully transparent color. Mouse help will show Alpha = 0 for fully transparent:

1: Add an arrow

Click on ***Blend or Overwrite Colors*** to activate the overwrite function to make it possible to draw without mixing tool and background color.

To make a semi transparent black arrow right click the black palette button, and set the rightmost (alpha) scrollbar to about 80 for half transparency:



Press OK.

The palette button and the drawing tools are now semi transparent black on white background:

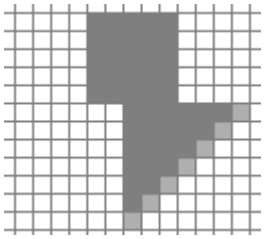
In the tool bar select ***Rectangle***

Draw a 5 x 5 rectangle.

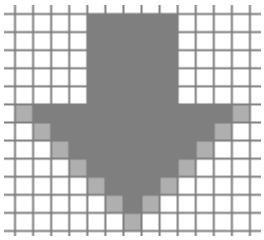
Select draw smooth line by right click on draw ***Triangle***:

White indicates that the tool will use anti-aliasing for smoothing.

Draw the right side of the arrowhead as 7x7 with the triangle function starting from the right angle:

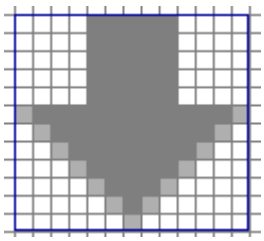


Then draw the left side as a 6x6 triangle:



Select ***Edit Inside Blue Frame***

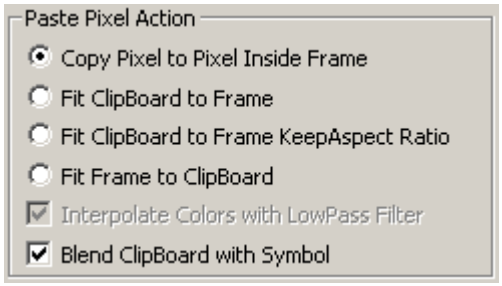
And make a frame with the size 13x13 around the arrow



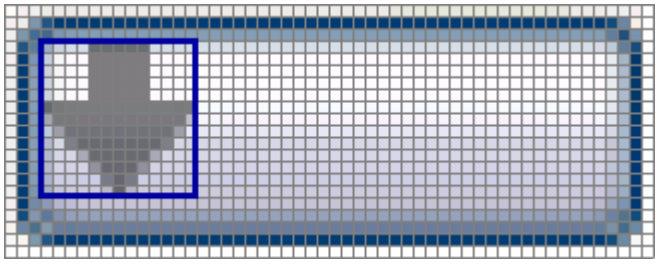
Copy to Clipboard with Ctrl + C and select symbol 0001.

Click on ***Blend or Overwrite Colors*** to activate the blend function to make it possible to mix tools and background colors.

Insert the arrow with Ctrl + V:

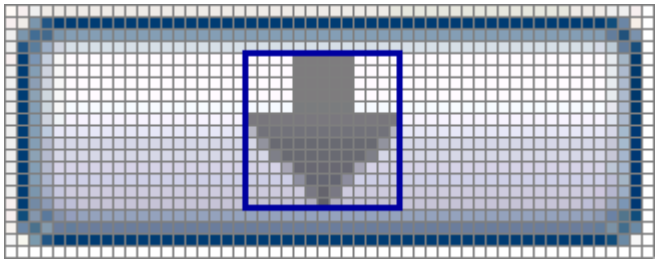


Press OK.



The picture is placed in a blue frame, and can be moved around with the mouse. The ‘Four Arrow’ cursor inside the frame indicates that the frame can be moved by holding the left mouse button down and moving the mouse, while the ‘Cross’ cursor outside the frame is for making a new frame thereby turning off the paste and move function.

Move the arrow to the middle with the mouse.



To make a green left arrow button change to symbol 0006 again, and make a semi transparent green color:

The floodfill function turns the start pixel into the new basic color and transparency. Connected pixels with the same basic color are changed too, but the transparency is scaled according to the new pixel's transparency.

To keep the semi transparency around 0x80 start the floodfill on the center of the arrow.

Put the frame around the arrow again, and turn the arrow left with the ***Turn 90 CW*** button. The button has a blue frame to indicate that it only turns inside the frame.

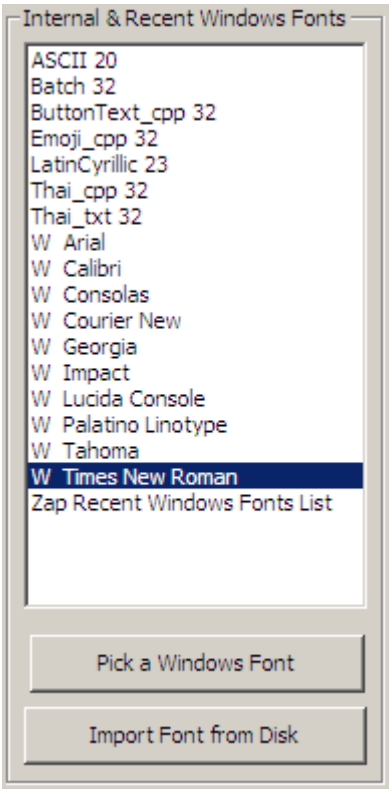
Copy the arrow to the Clipboard, select symbol 0000, paste it in frame, and move again.

Symbols 0000 and 0001 now look like this:

2: Add a text

To select a master font press ***Master Font*** in the main tool bar:

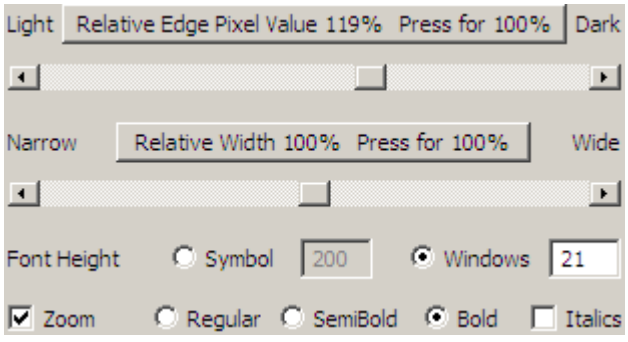
First select a Windows font:



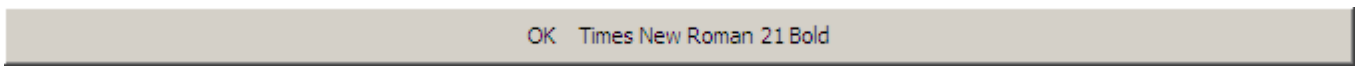
This is basically a grey tone vector font, and the conversion to black & white or to grey level has to be fine trimmed.

Adjust the darkness of the character until the connectivity is OK.

Characters with diagonal lines like / @ A W are usually the most critical.



Press OK



Make a semi transparent red:

Change to symbol 0003.

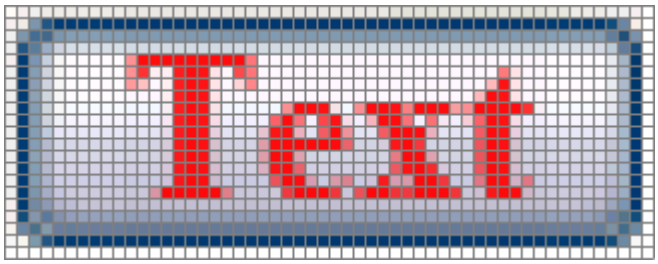
For smoothing and/or semi transparency right click the ***Draw Text from Keyboard*** button:

Click until it turns white to indicate anti-aliasing, then left click to activate.

This automatically changes the ***Master Font Selector*** to the ***Virtual Keyboard***:

Check that ***Blend or Overwrite Colors*** is on to enable blending of the text with the background:

Write “Text” on the keyboard and move the frame around the text with the mouse to get this:



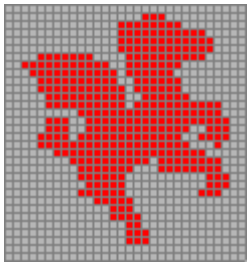
Switch to symbol 0002, make a semitransparent green, press the Draw Text from Keyboard button and write “Text” again:

3: Import a logo

Assume that the logo already exist as *Pegasus.c*.

Press ***File Open***

And select *Pegasus.c*. It will look like this:



Copy Pegasus to the clipboard with ***Copy to ClipBoard***:

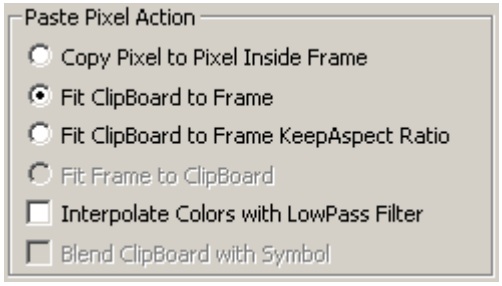
Switch back to ***ShadedButtons*** and select symbol 0006, clear the symbol with ***Cut to ClipBoard***.

Click on ***Blend or Overwrite Colors*** to activate the overwrite function to make it possible to draw without mixing tool and background color.

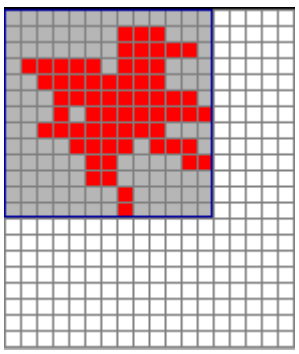
Select ***Edit Inside Blue Frame***

And make a frame with the size 13x13

Paste Pegasus to symbol 0006 with ***Paste from ClipBoard***:



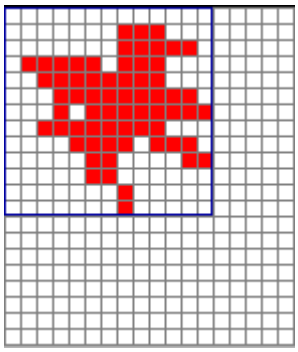
It now looks like this:



Select a fully transparent color

Select ***Surface Flood Fill***:

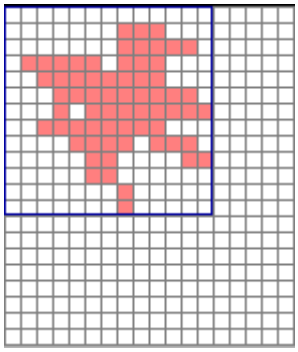
Click on the grey parts of the symbol to fill it with the new fully transparent color.



The logo should be blended with the shading of the button. Click on ***Blend or Overwrite Colors*** to activate the blend function and then right click the ***Change Alpha Level*** button and set it to -127

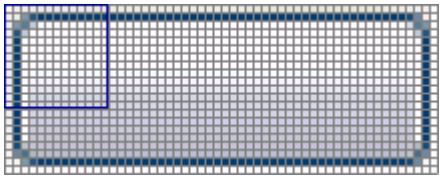
In the tool bar select ***Rectangle***

Draw a 13 x 13 rectangle over Pegasus to make it semi transparent and select the frame again.

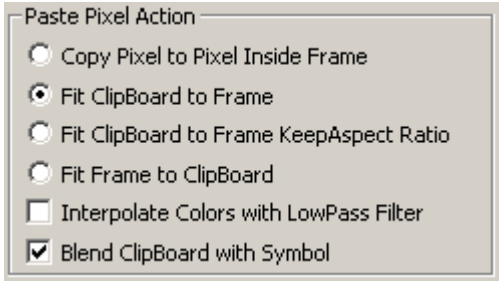


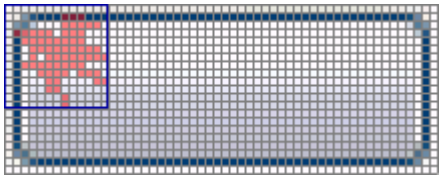
Pegasus is now semi transparent and can be blended with the background, copy it to the clipboard with ***Copy to ClipBoard***

Switch to symbol 0004:

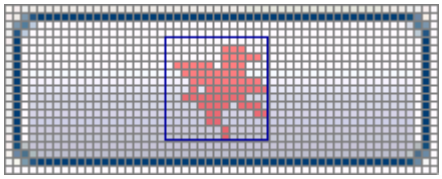


Paste with ***Paste from ClipBoard***





Move Pegasus in place with the mouse.



To remove the temporary symbols switch to Group, mark symbol 0006 with the mouse and delete with ***Delete***.

To save the symbols:

Press the ***Save All As…*** button in the main tool bar to save the pixel data in the *ShadedButtonsWithText.c* file in the proper directory.

D: How to Draw a 5 Arm Anti-Aliased Star by Adding Transparent Layers

This example applies only to the color version of IconEdit.

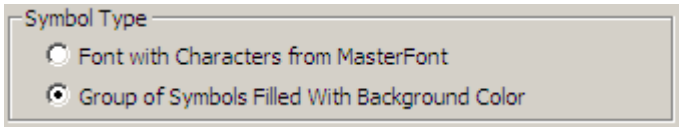
In this example, the aim is to turn one arm into 5 arms while keeping the shading.

Drawing is done with 32 bit ARGB transparency, but the final result can be saved as normal B&W, Alpha, Grey, RGB or Palette based files. If you have a B&W license you can use IconEdit in color demo mode in a separate directory for the first transparent part of the drawing process and then copy to the B&W version of IconEdit for saving.

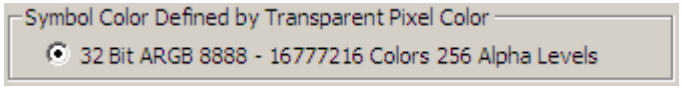
To make a new symbol press the ***New Font or Symbol Group*** button in the Main Toolbar:

This opens the create dialog box:

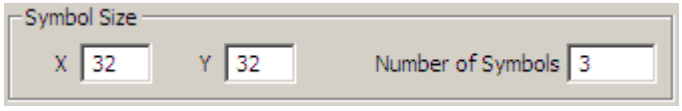
Choose how characters or symbols should be organized and shown initially:



Choose a color format:

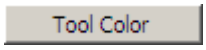


Choose size and number:

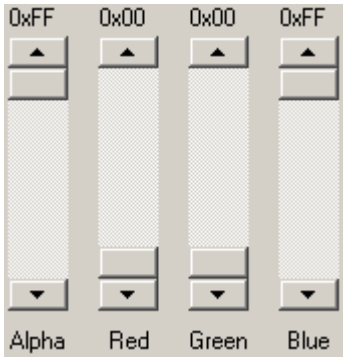


Set tool and background to the same color but different transparencies:

Press ***Tool Color***.



Set ***Alpha*** to maximum:

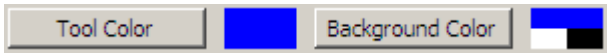


Press ***OK***.

Press ***Background Color***.

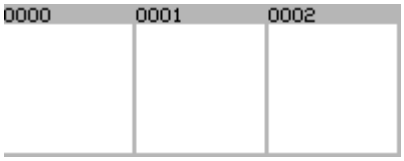
Set ***Alpha*** to minimum:

Press ***OK***.

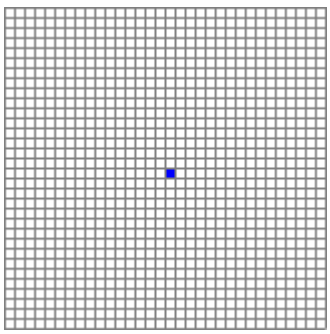


Tool and background are now the same color, but the tool is fully opaque, and the background is fully transparent as indicated by the transparency indicator at the bottom of the color field.

Press ***OK*** to get the 3 symbols:

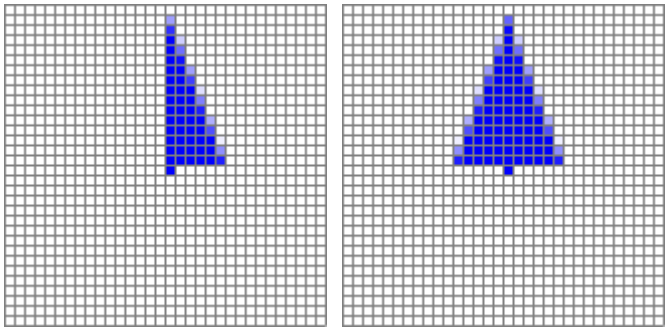


Right-click the first to edit and set a center point at 16,16:



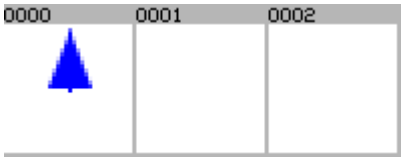
Select ***Solid Triangle*** with smoothing by right-click, the button changes color to indicate smoothing:

Draw 2 halves of the first arm as 6x15 triangles both starting at 16,15:

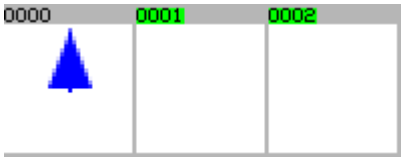


Copy to ClipBoard with

Switch to ***Group Edit***:

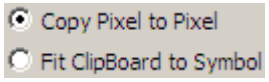


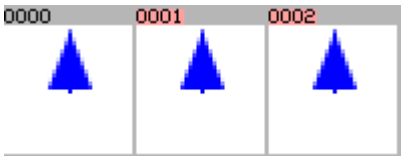
Mark the other 2 symbols with the mouse:



Paste with ***Paste from Clipboard***.

Choose ***Pixel to Pixel*** and ***Overwrite Marked Symbols***:

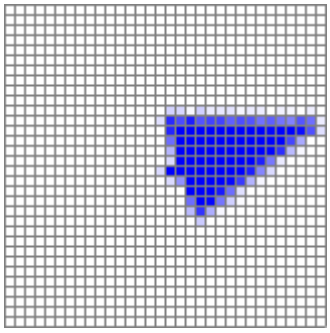




Right-click symbol 0001 and choose ***Turn 360 Degrees with ScrollBar***:

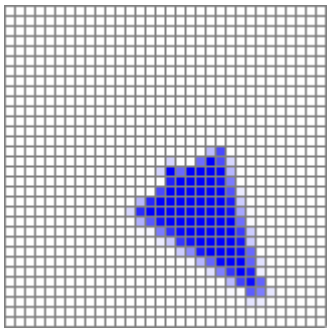
Left-click the center pixel 16,16 and scroll down to 72 degrees:

Press ***OK***.



Left-click symbol 0002.

Left-click the center pixel 16.16 and scroll down to 144 degrees:

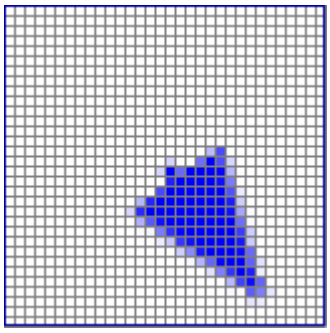


Press ***OK***.

Choose ***Blend Colors***:

Choose ***Edit Inside Blue Frame*** because blending ONLY works within a frame:

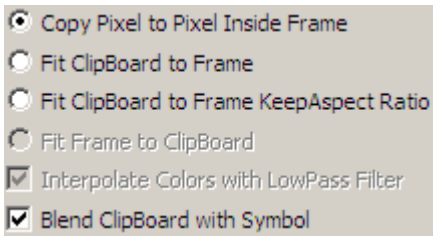
Right click the button for a frame the full size of the symbol with the mouse:



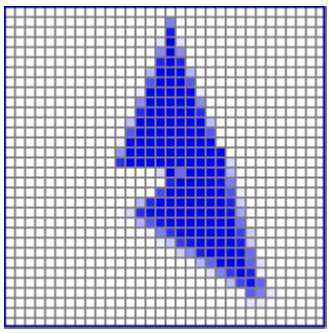
Copy to ClipBoard with

Choose symbol 0000 and ***Paste from Clipboard*** with

Choose ***Pixel to Pixel*** and ***Blend***:



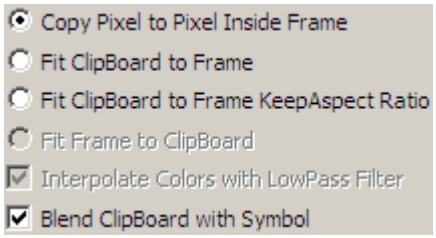
Press ***OK***.



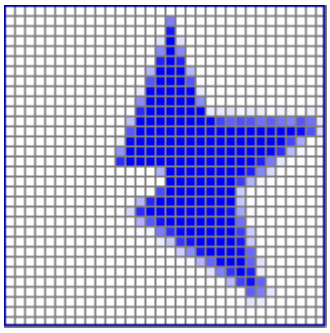
Choose symbol 0001, ***Copy to ClipBoard*** with

Choose symbol 0000 and ***Paste from Clipboard*** with

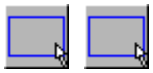
Choose ***Pixel to Pixel*** and ***Blend***:



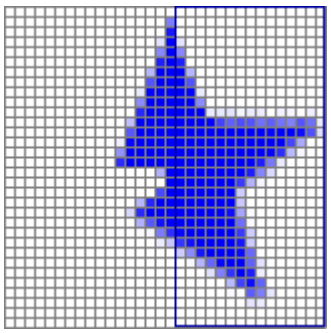
Press ***OK***.

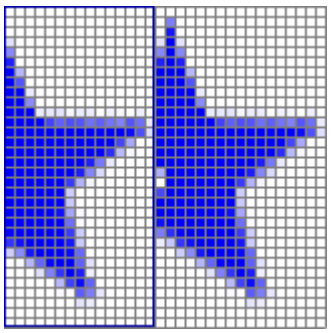


Turn off the blue frame and start on a new with 2 clicks on ***Edit Inside Blue Frame***:

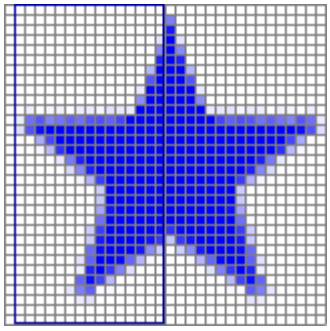


Make the new frame as just under half size and move it to the left:

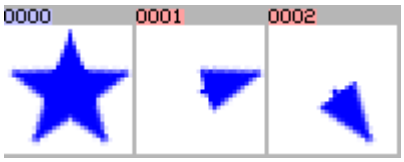




Use ***Mirror Vertical*** to make the rest of the star:



Remove the frame by click on ***Edit Inside Blue Frame*** or a mouse-tool such as ***Draw Pixels***, the whole group now looks like this:



To remove the temporary symbols switch to ***Group Edit***, mark symbol 0001 and 0002 with the mouse and ***Delete*** with

In natural size:

To save the symbol:

Press the ***Save All As…*** button in the main tool bar to save the pixel data in the *Star_5.c* file in the proper directory.

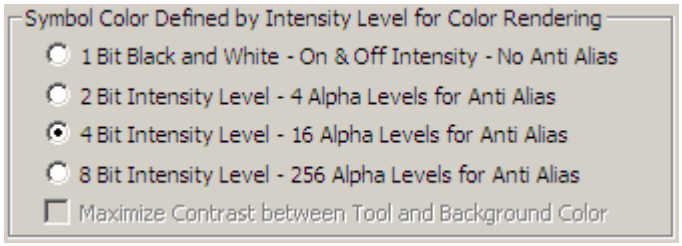
1: Save only the Alpha Channel with Reduced Memory Footprint

The symbol in *Star_5.c* has 32 bit per pixel, but all the relevant information is in the alpha channel with 8 bit per pixel.

In most cases even the 8 bit is overkill, below is shown resolutions of 8, 4, 2 and 1 bit per pixel:

In this case 4 bit seems reasonable so press ***Modify Transparent Color***:

Choose 4 bit as this is usually enough:



Press ***OK***:

The symbol is ready for use, and can later be displayed in any color:

The star is ready, press ***Save Dataset As*** to save as *Star_5.c*:

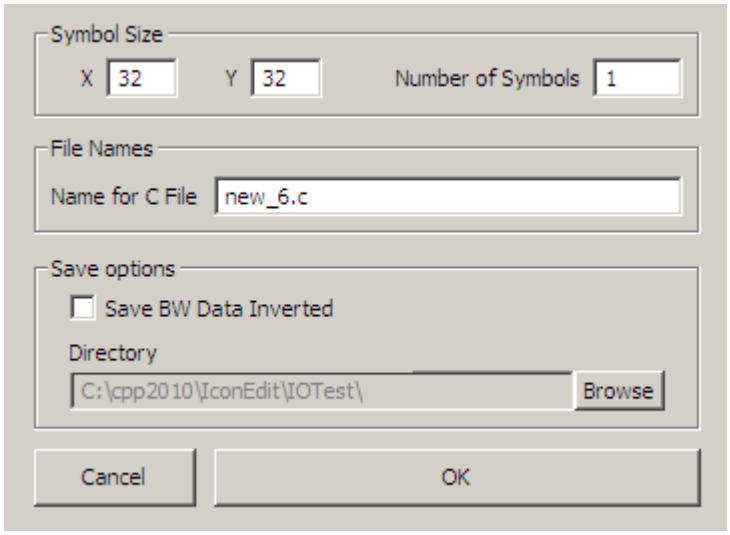
Finished in color mode.

2: Save from IconEdit with a Black & White license

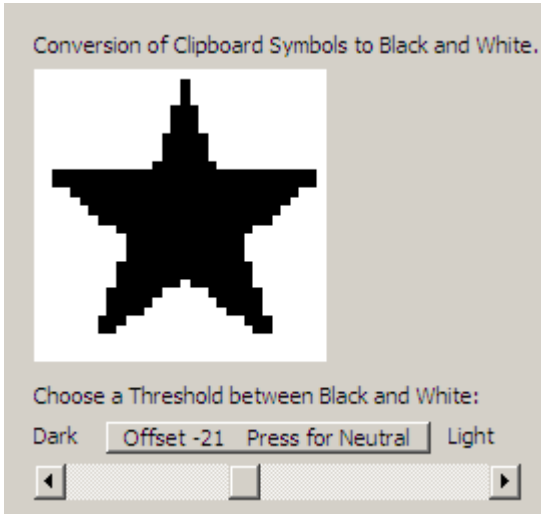
Copy to ClipBoard with

Start IconEdit B&W.

Paste from Clipboard as New



Press ***OK***.



Choose ***Offset*** for best look and press ***OK***:

Save As normally.

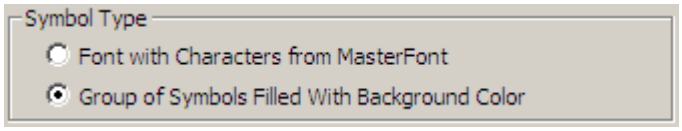
E: How to Draw a Speedometer by Adding Transparent Layers

Drawing is done with 32 bit ARGB transparency, but the final result can be saved as normal B&W, Alpha, Grey, RGB or Palette based files. If you have a B&W license you can use IconEdit in color demo mode in a separate directory for the first transparent part of the drawing process and then copy to the B&W version of IconEdit for saving.

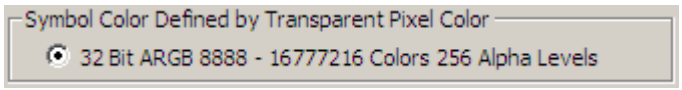
To make a new symbol press the ***New Font or Symbol Group*** button in the Main Toolbar:

This opens the create dialog box:

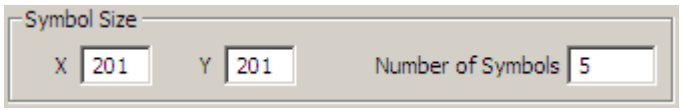
Choose how characters or symbols should be organized and shown initially:



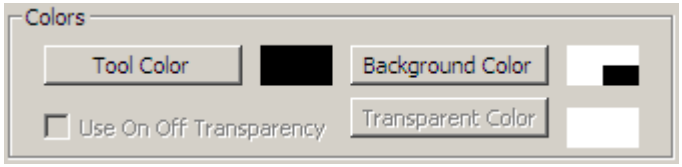
Choose a color format:



Choose size and number:

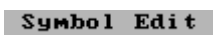


Check that the Tool is Black and the Background is fully transparent White:



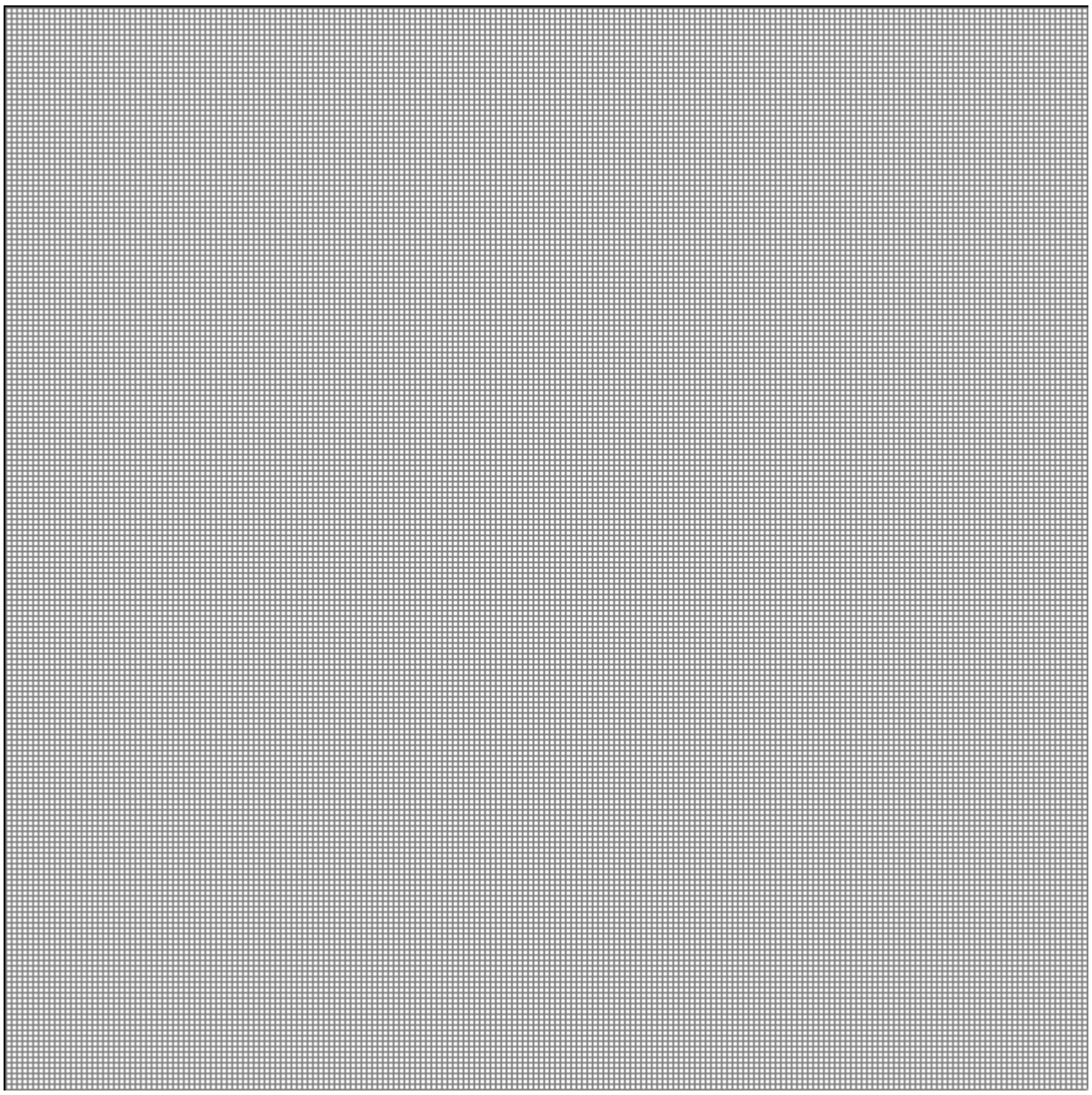
Pres ***OK***.

Change to ***Symbol Edit***:

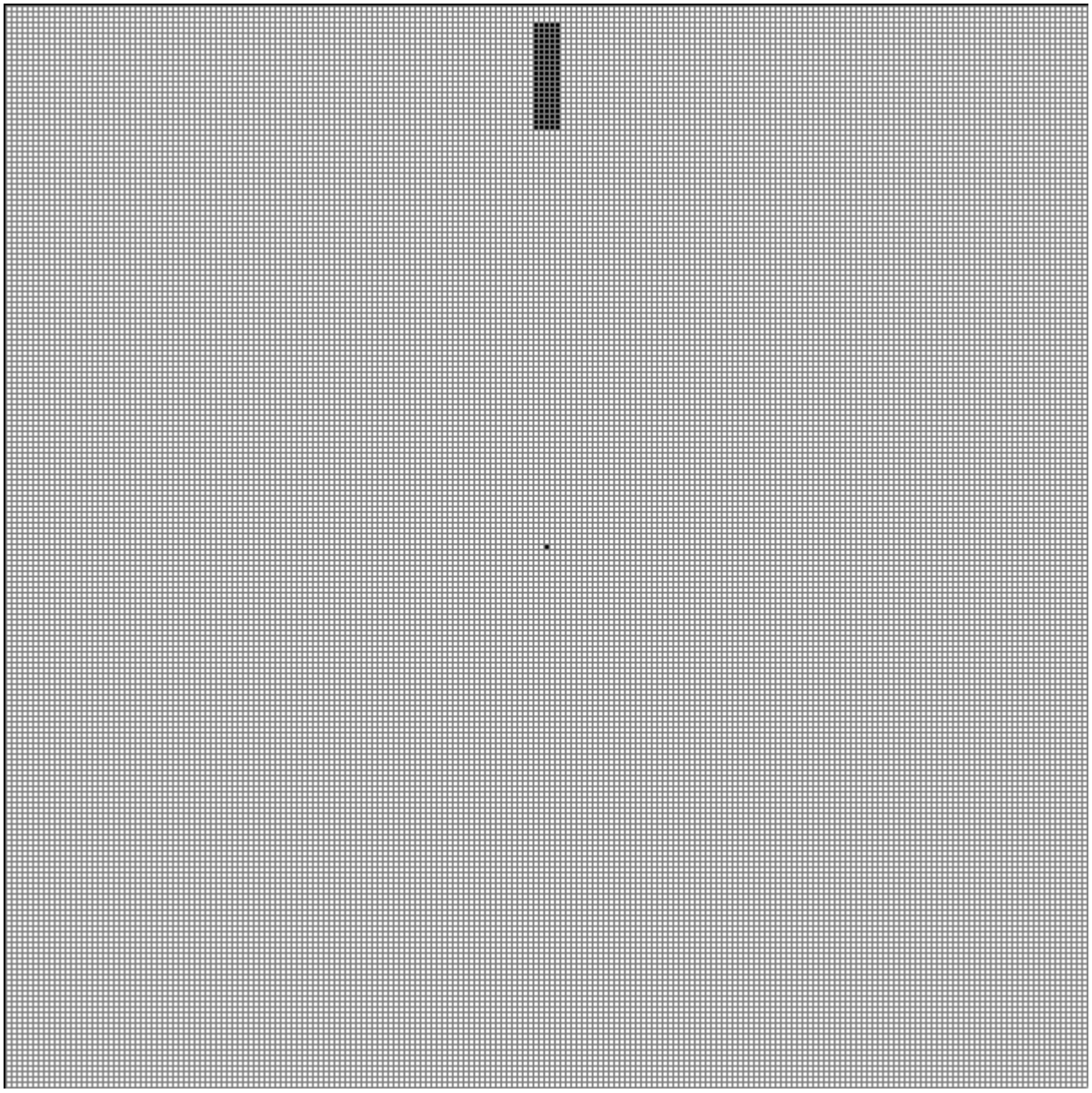


Make the symbol bigger with ***Change Zoom to 3***:

Activate the grid with ***Show Grey Pixel Grid***:



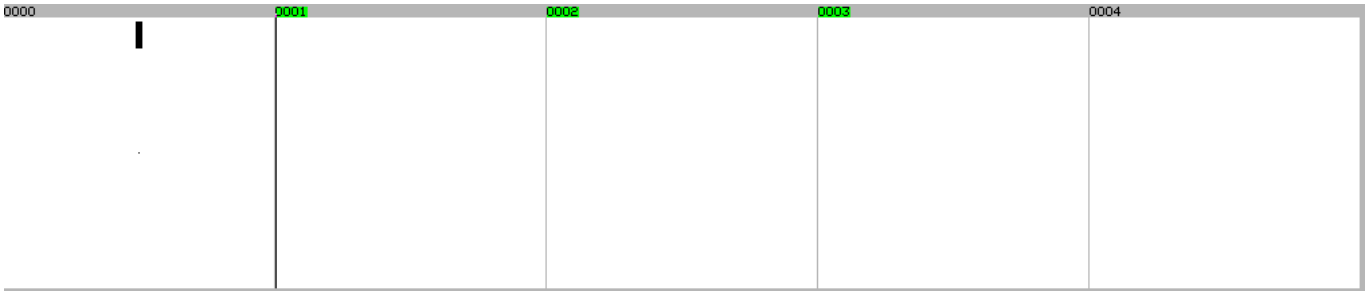
Mark the center at 100,100 and draw a 5x20 rectangle at 98,3



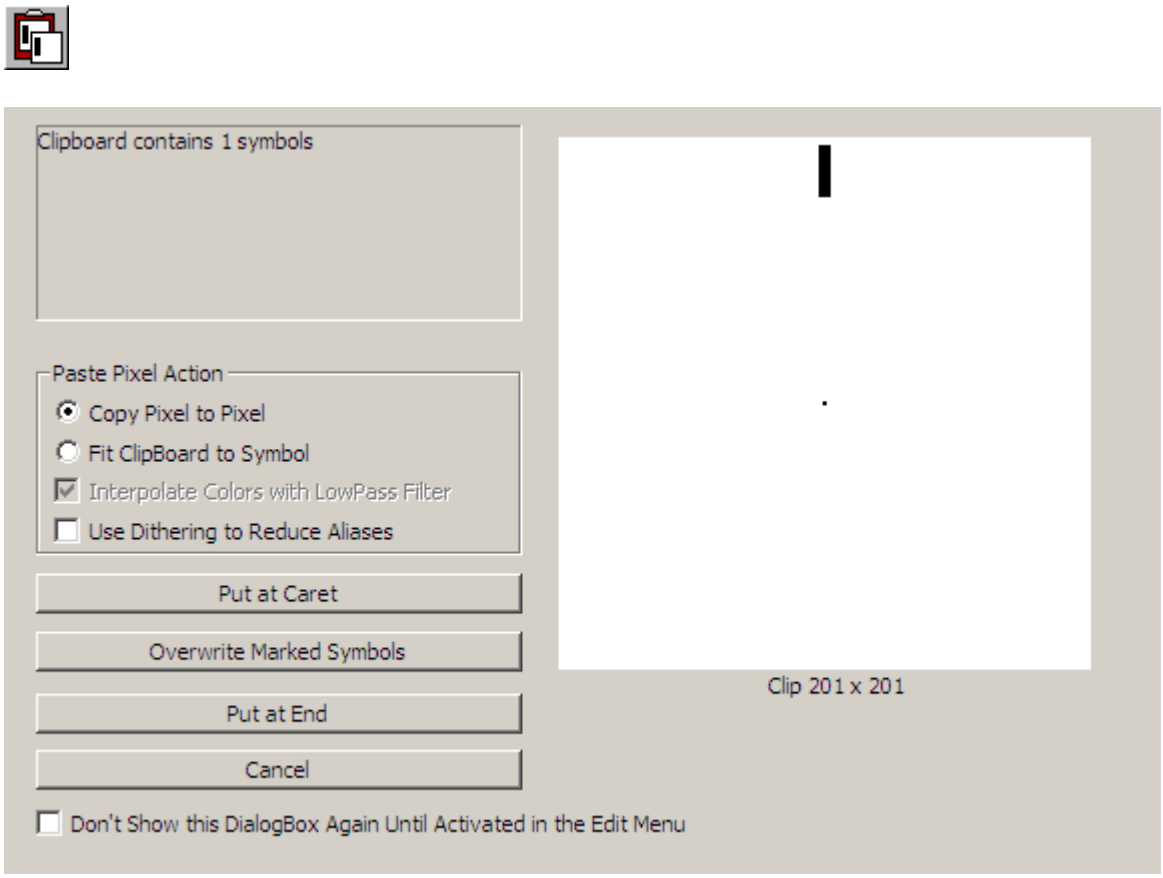
Copy to ClipBoard with

Change to ***Group Edit***

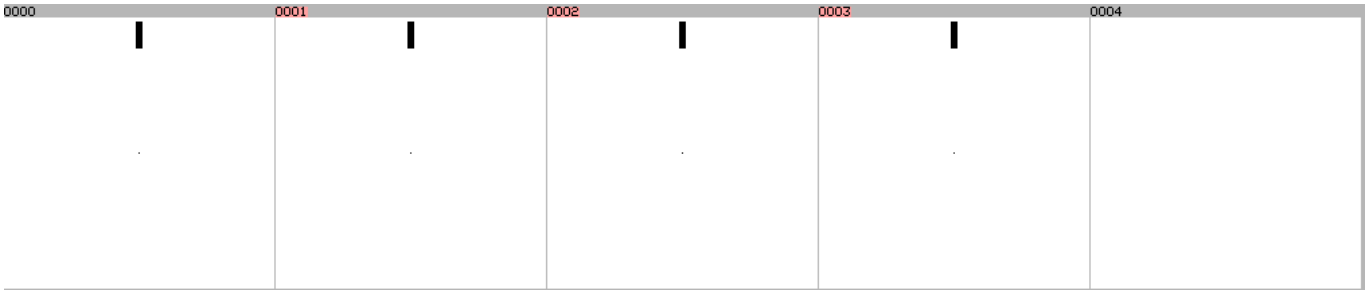
Mark symbols 1, 2, 3 with the mouse:



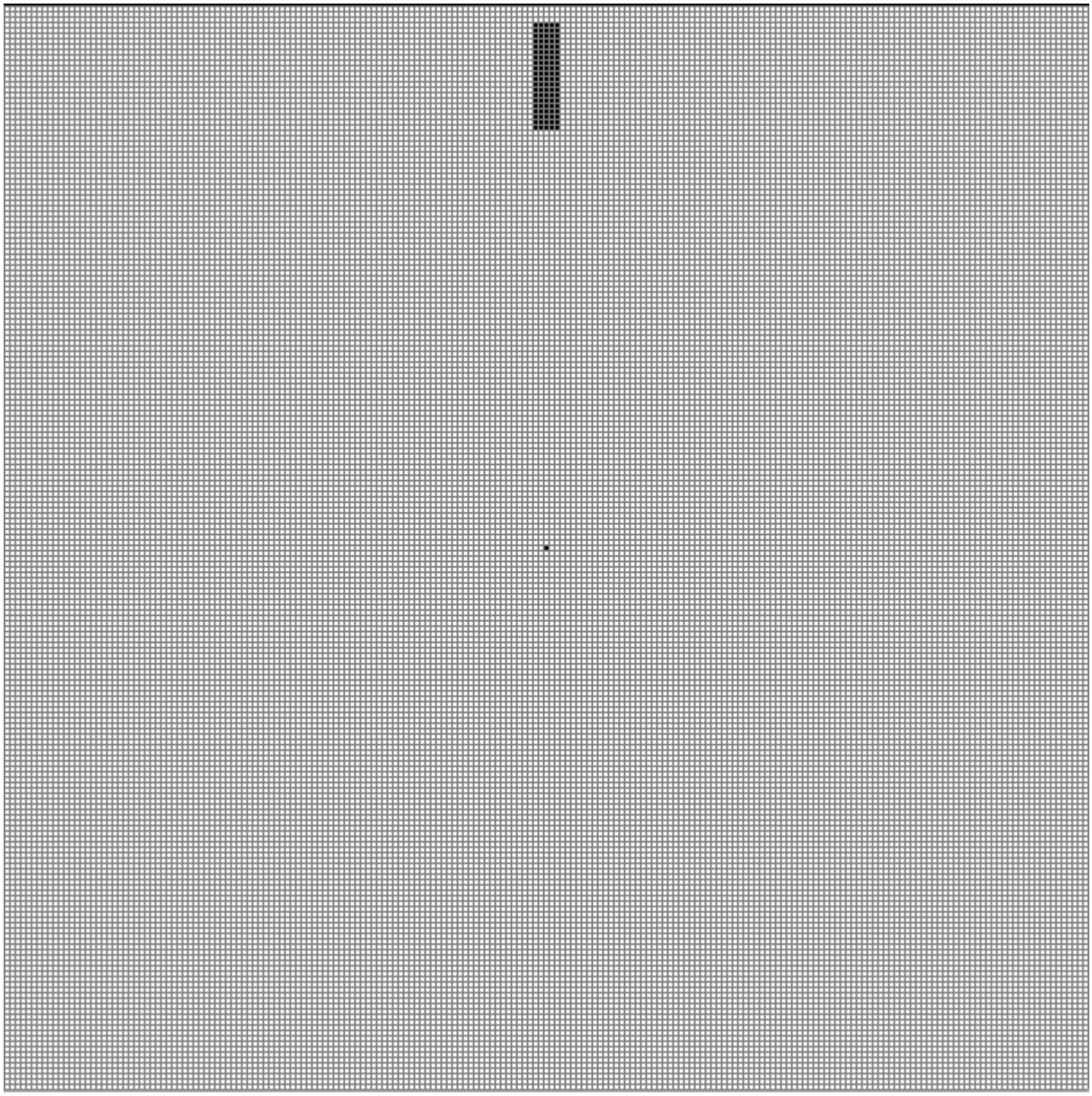
Paste from Clipboard with



Press ***Overwrite Marked Symbols***:

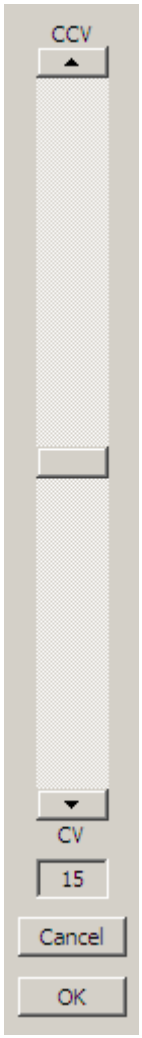


Right click symbol 1 to change to symbol edit:

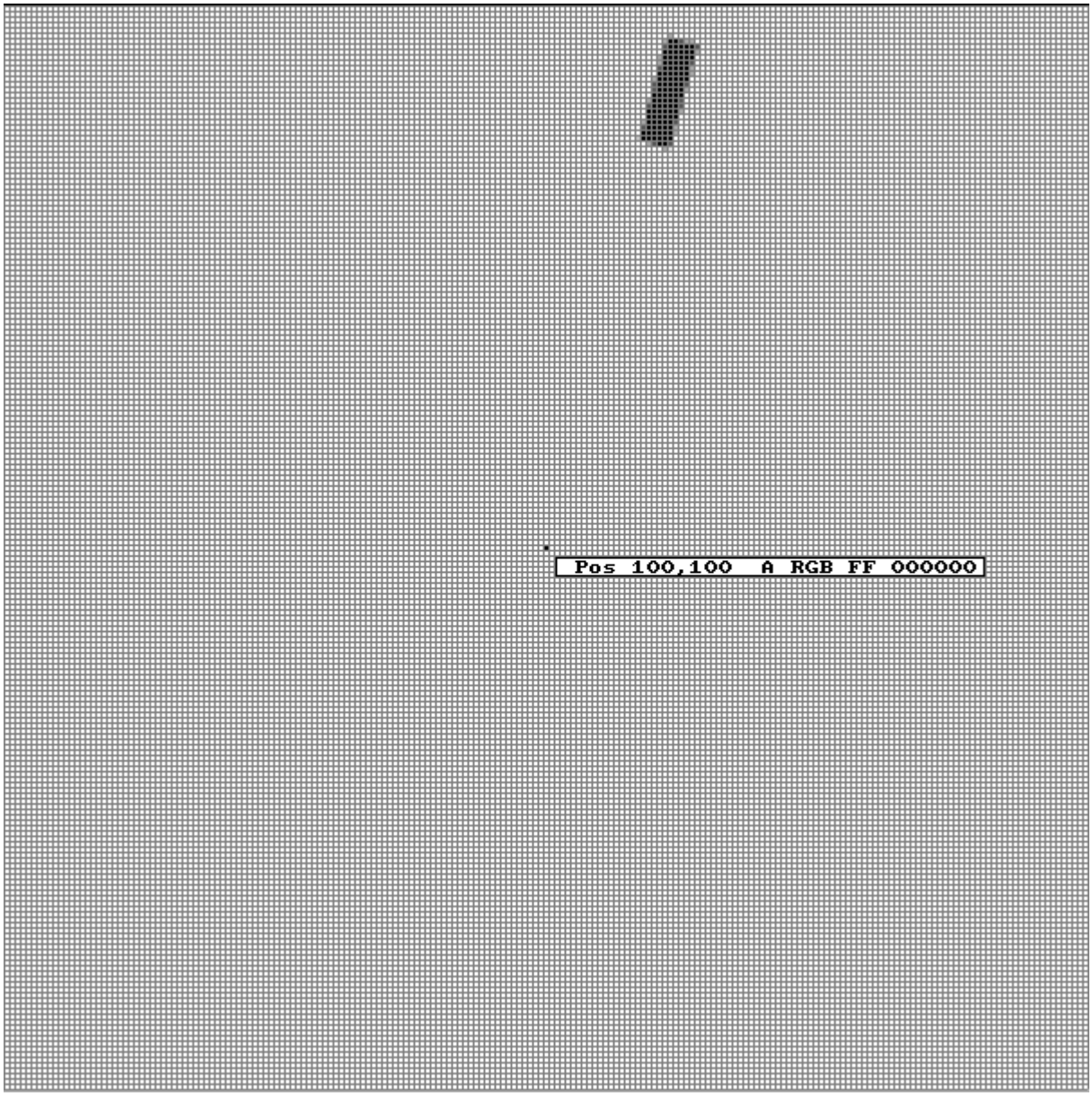


Right click ***Turn 360 Degrees with ScrollBar*** to activate smoothing:

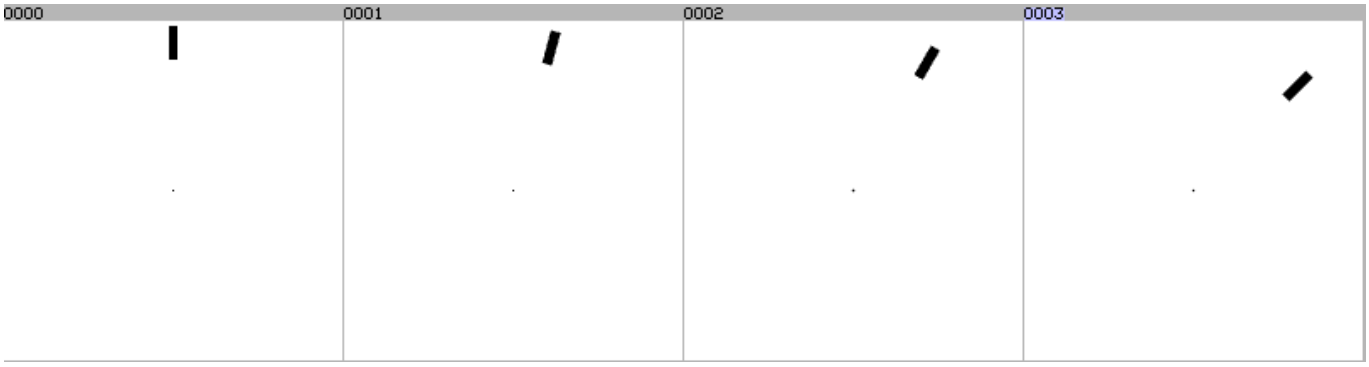
Click the Center and turn 15 degrees:



Press ***OK***:



Change to symbol 2 and 3 with ***Tab*** and repeat the turning with 30 and 45 degrees:



Change back symbol 1 with ***Shift Tab*** and ***Copy to ClipBoard***. with

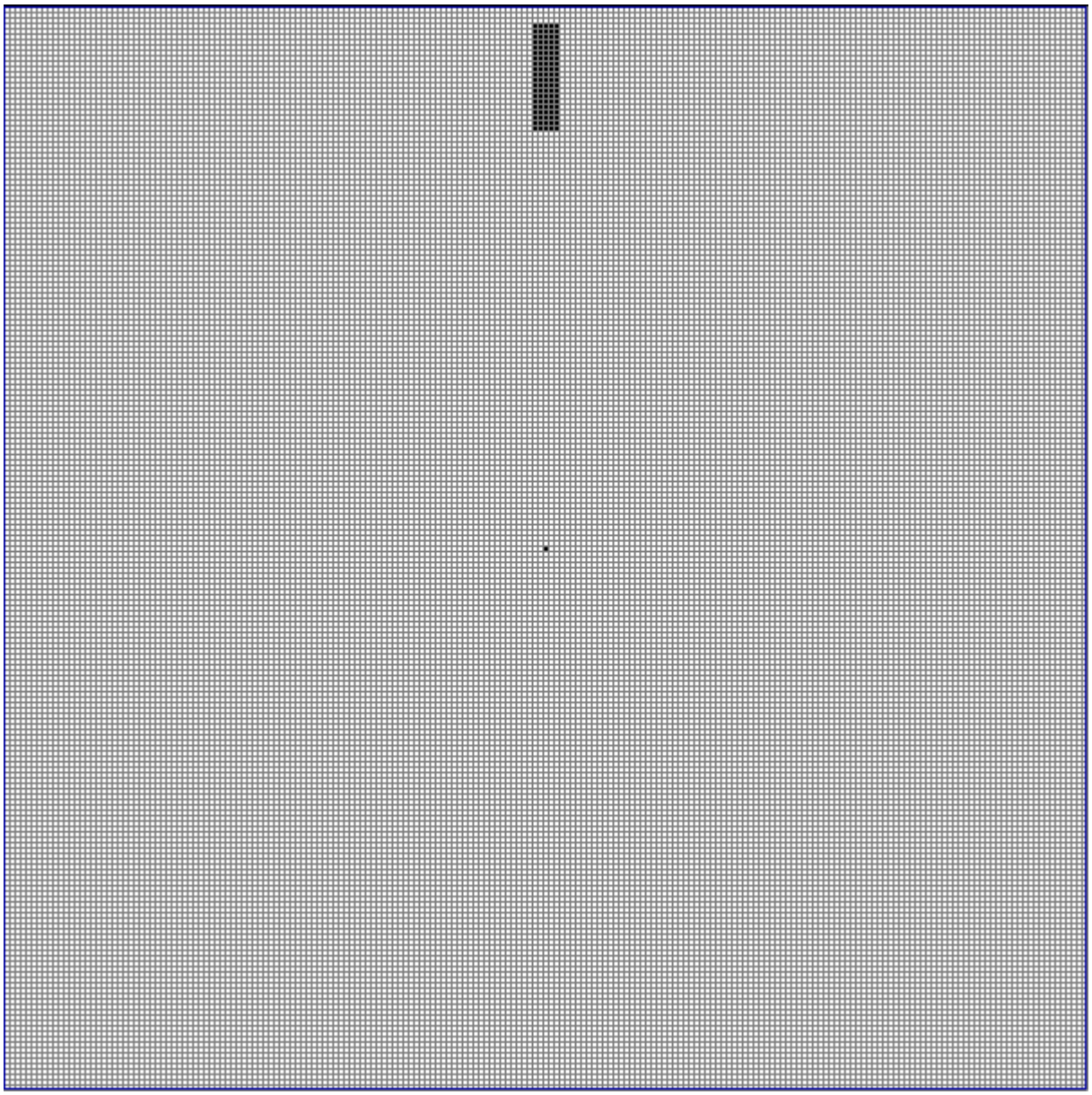
Change back symbol 0 with ***Shift Tab***.

Activate ***Overwrite Colors*** to ***Blend Colors***:

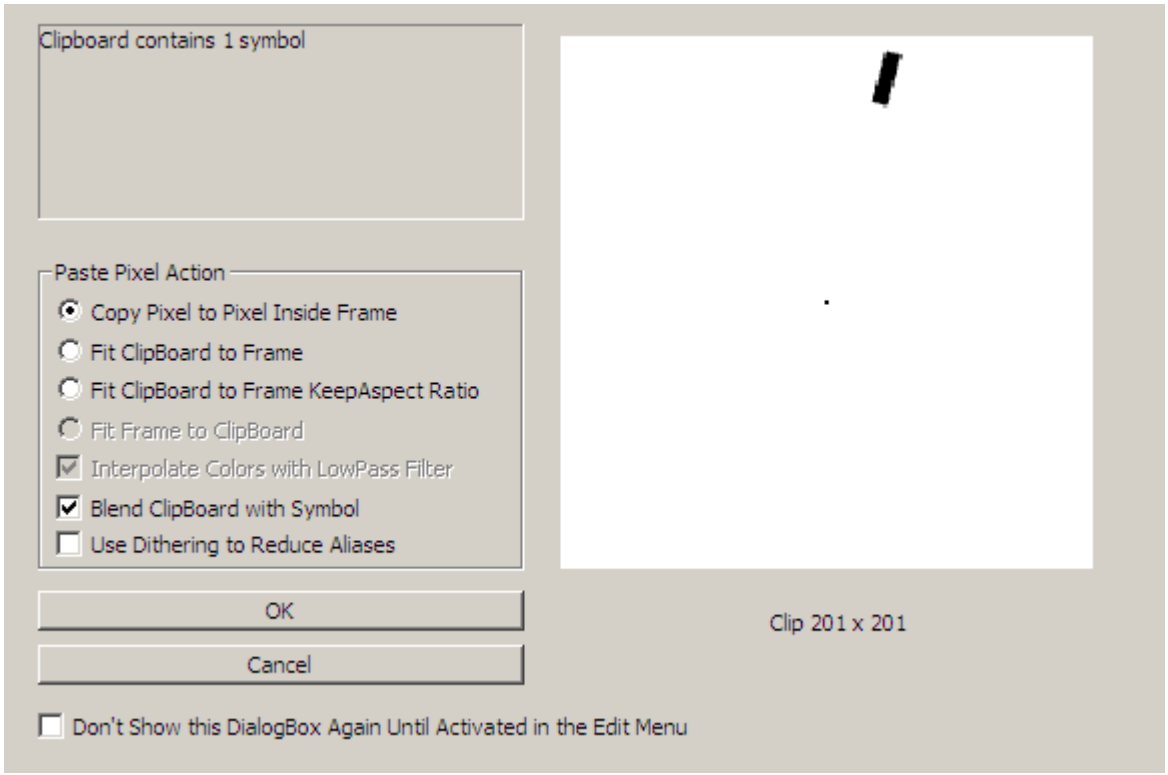
Activate ***Edit Inside Blue Frame***:

Mark the whole symbol (blending **only** works inside the frame):

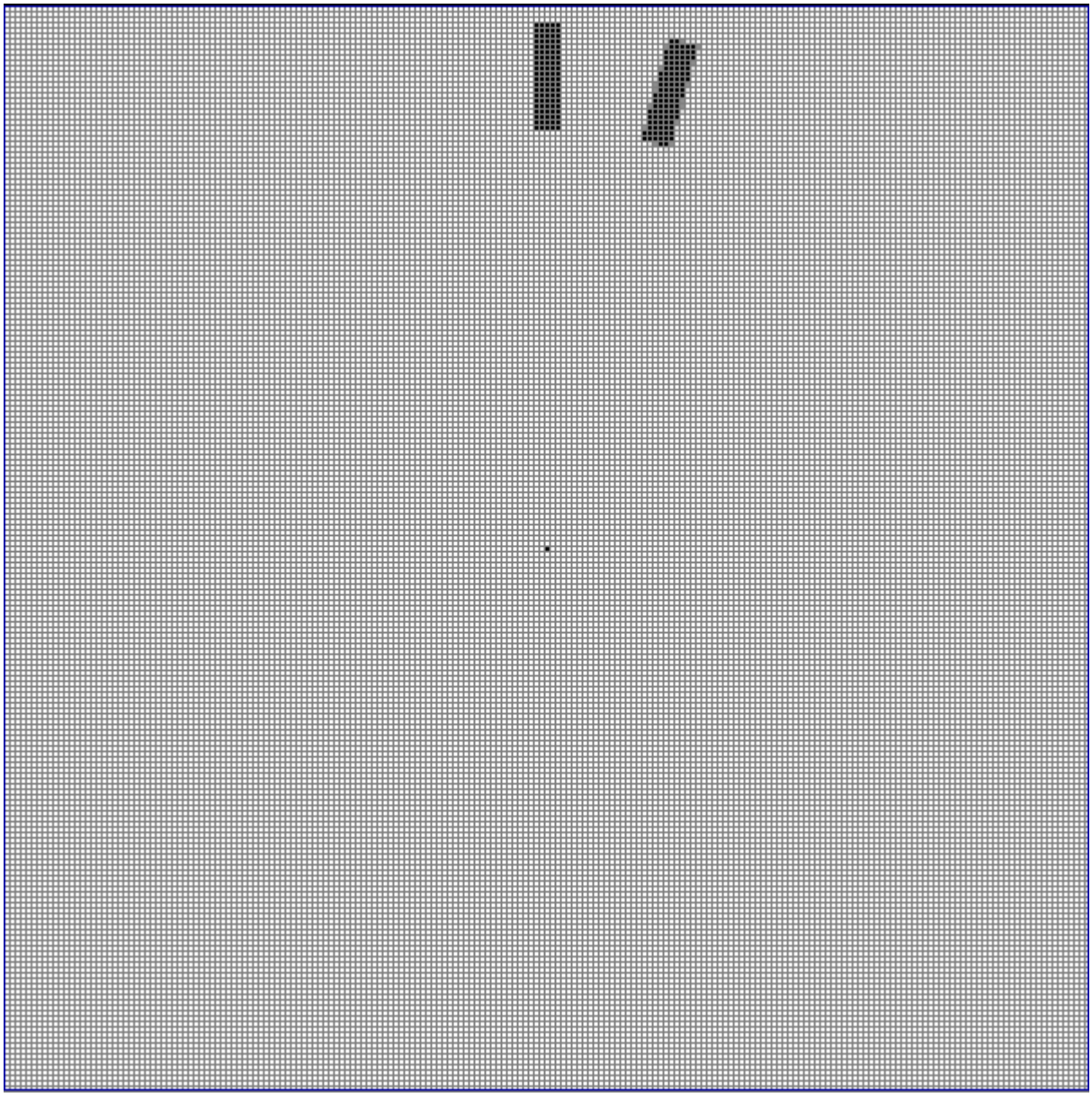
Right click the button for a frame the full size of the symbol with the mouse:



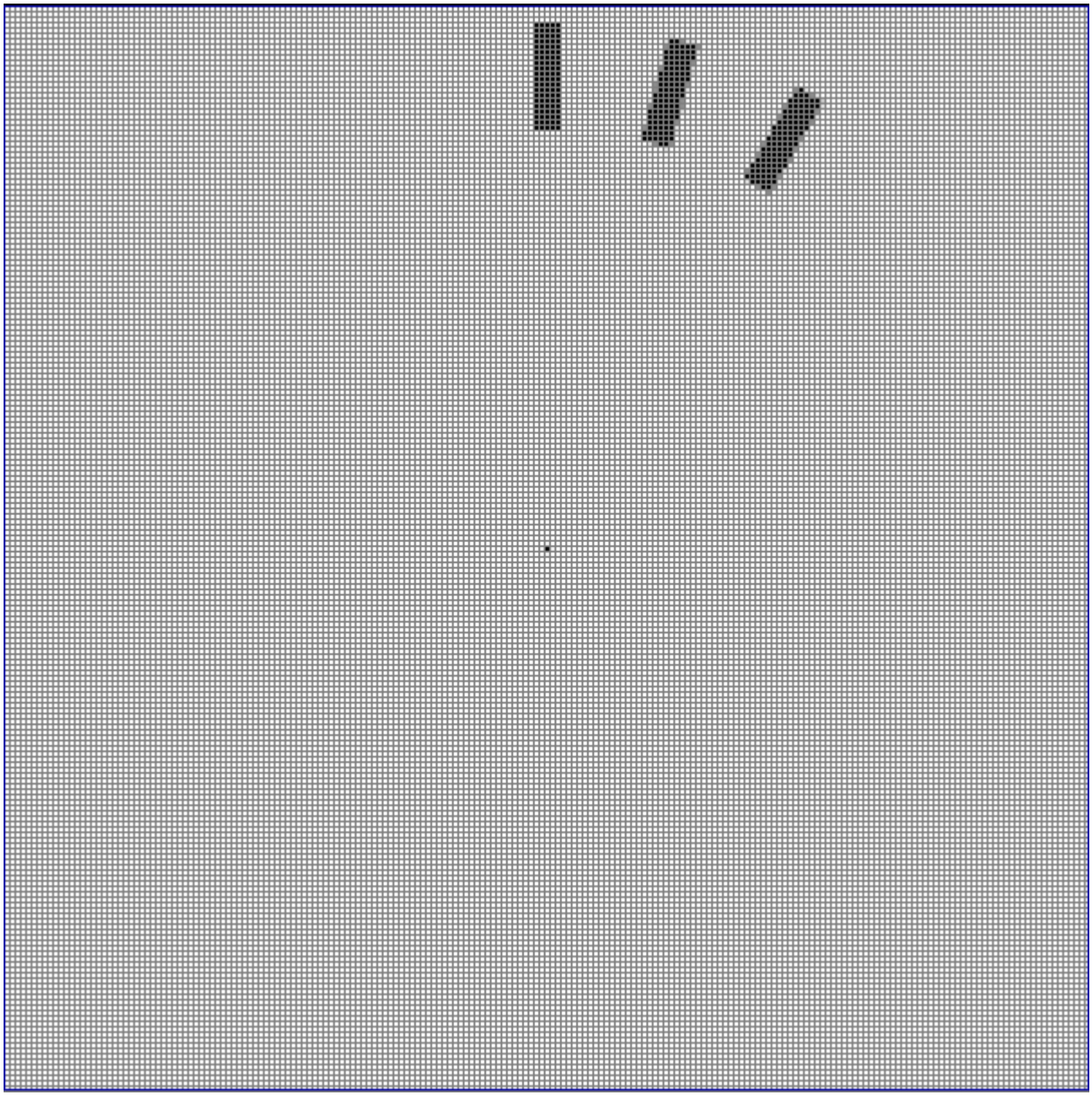
Paste from Clipboard with



Choose ***Blend ClipBoard with Symbol*** and press ***OK***:



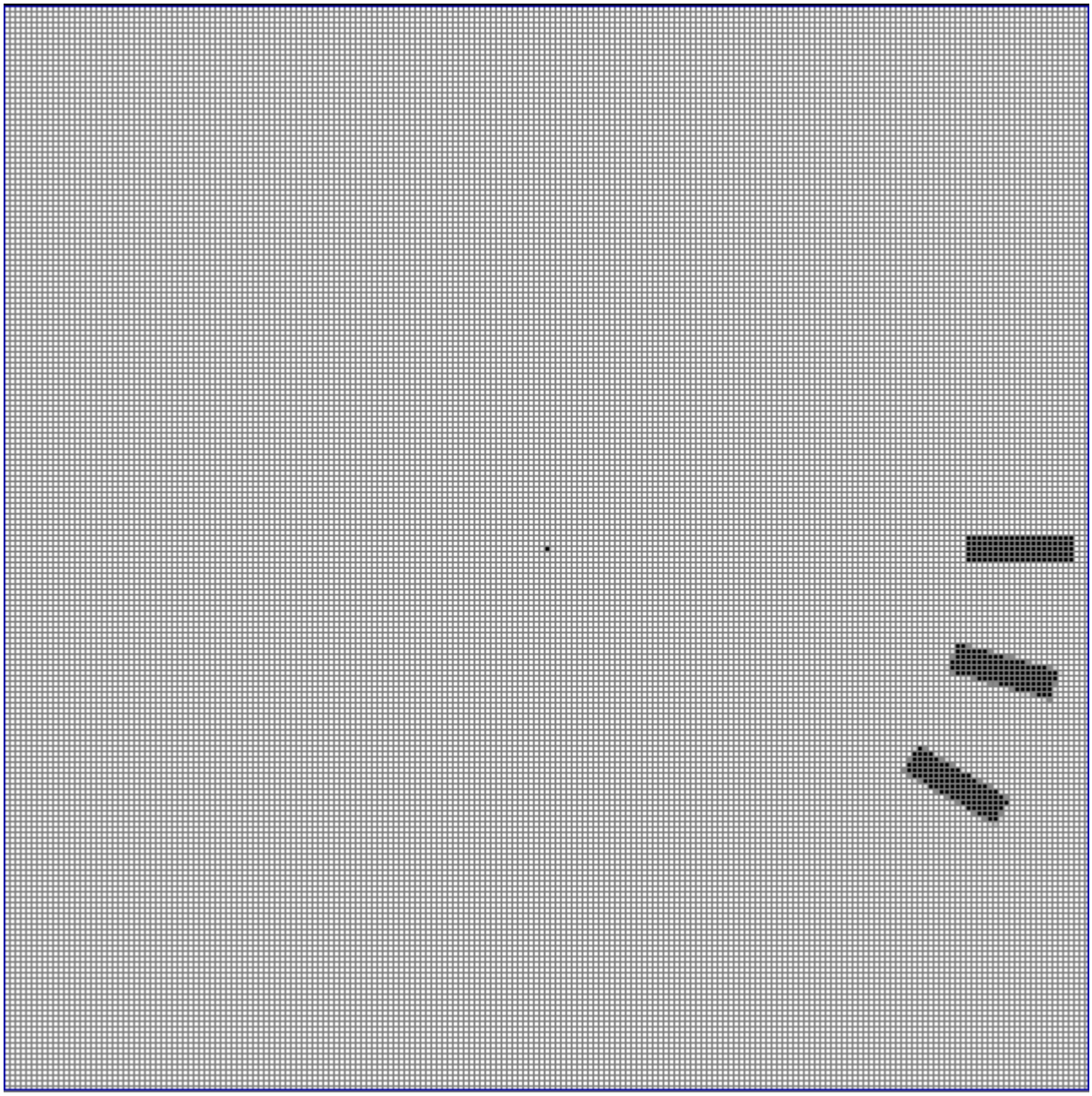
Repeat with symbol 2:



Copy to ClipBoard with

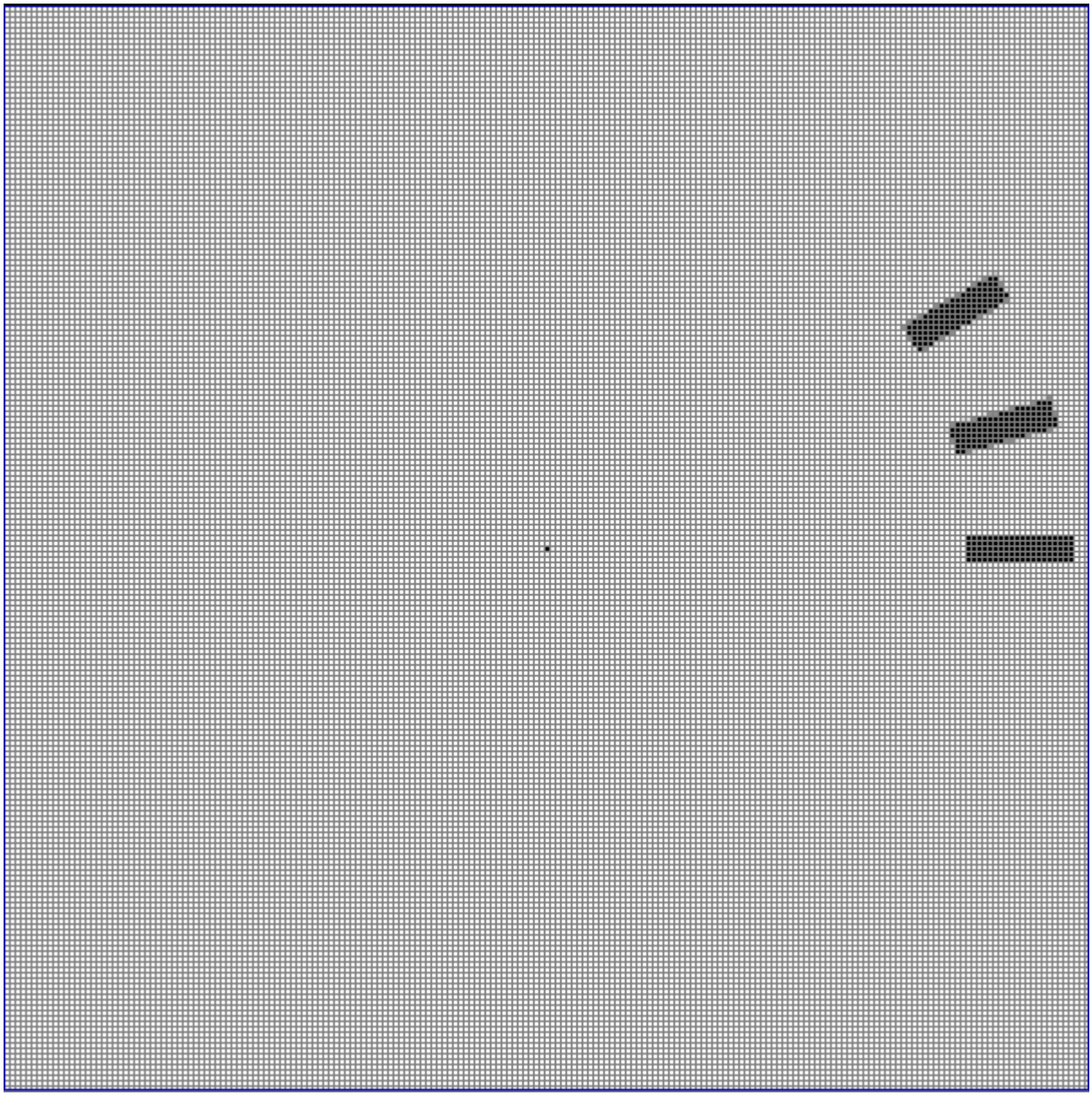
Change to symbol 4 and ***Paste from ClipBoard*** with

Turn ***90 degrees CW***:

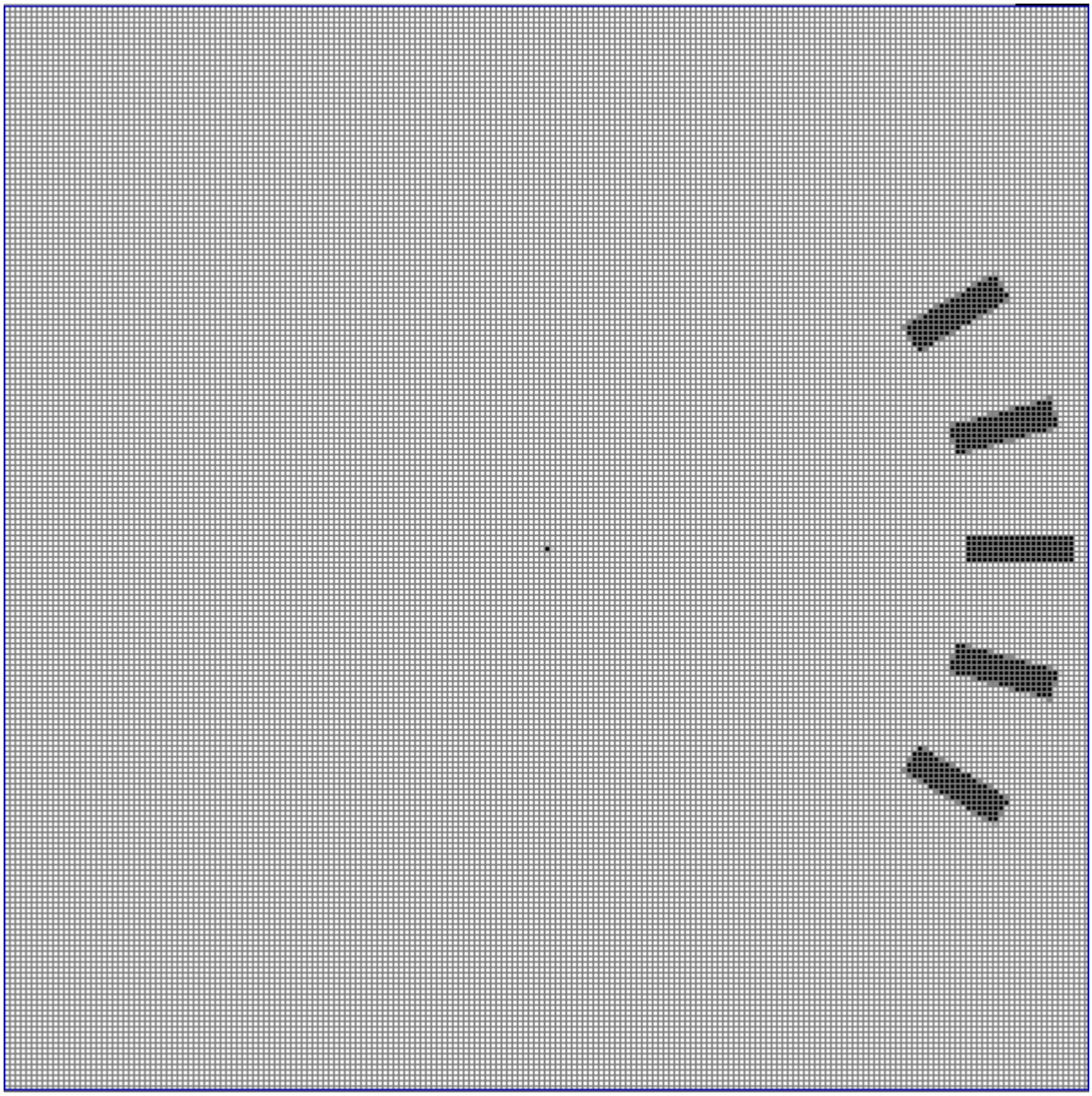


Copy to ClipBoard. With

Mirror Horizontal:

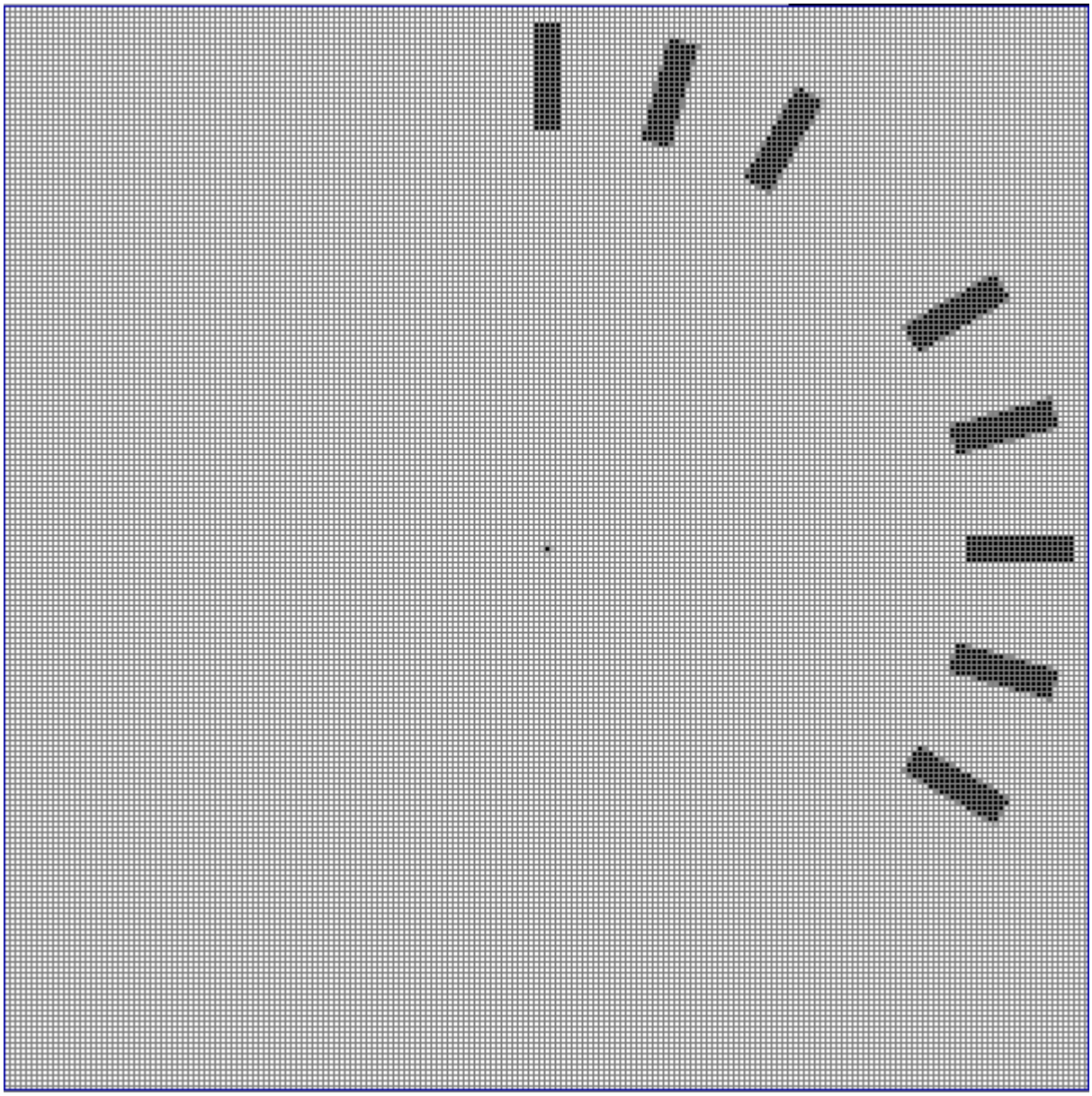


Paste from Clipboard:



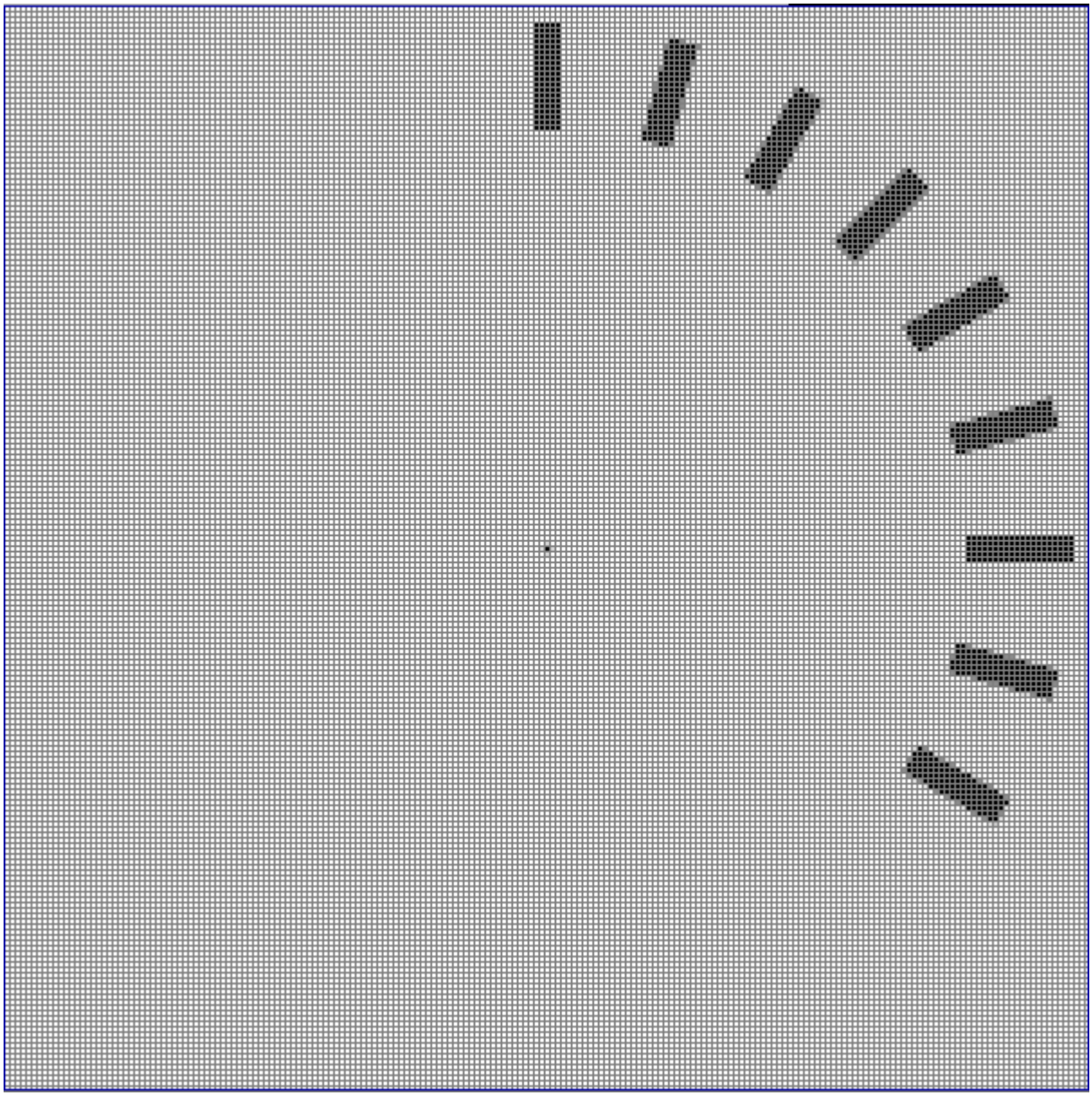
Copy to ClipBoard. With

Change to symbol 0 and ***Paste from Clipboard***:



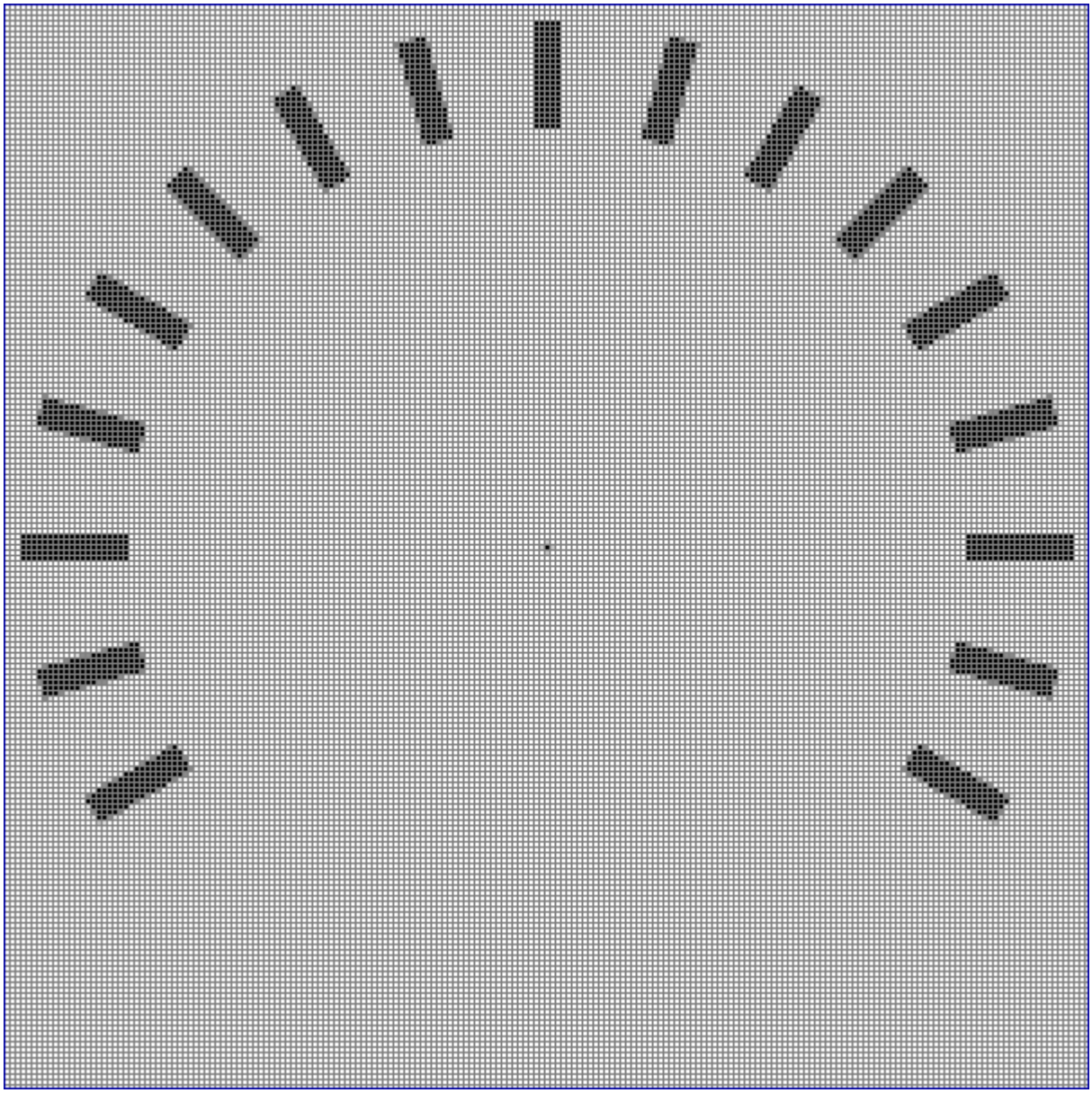
Change to symbol 3 and ***Copy to ClipBoard*** with

Charge back to symbol 0 and ***Paste from Clipboard***:

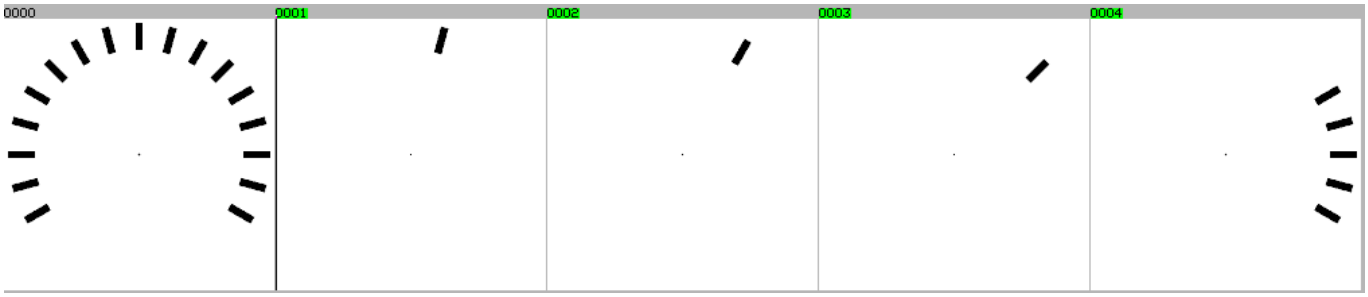


Copy to ClipBoard.

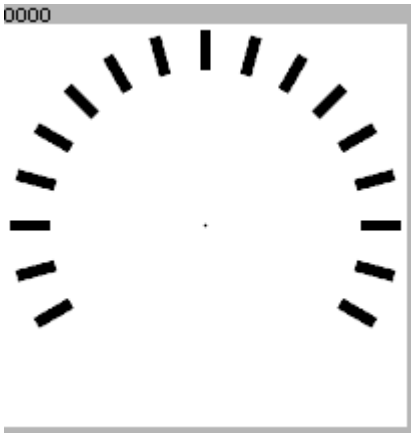
Mirror Vertical and ***Paste from Clipboard***:



To delete the temporary symbols mark symbol 1 to 4



Press ***Delete***



The basic speedometer scale is ready, press ***Save Dataset As*** to save as *Speedometer.c*:

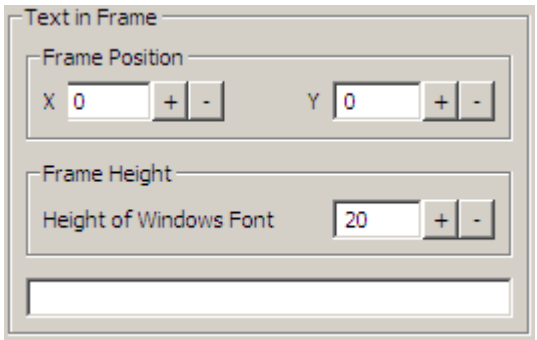
1: Add Numbers to the speedometer

To add anti-aliased numbers click ***Write AntiAliased Text***:

To add normal B&W numbers right click ***Write AntiAliased Text*** and the click ***Write Normal Text***:

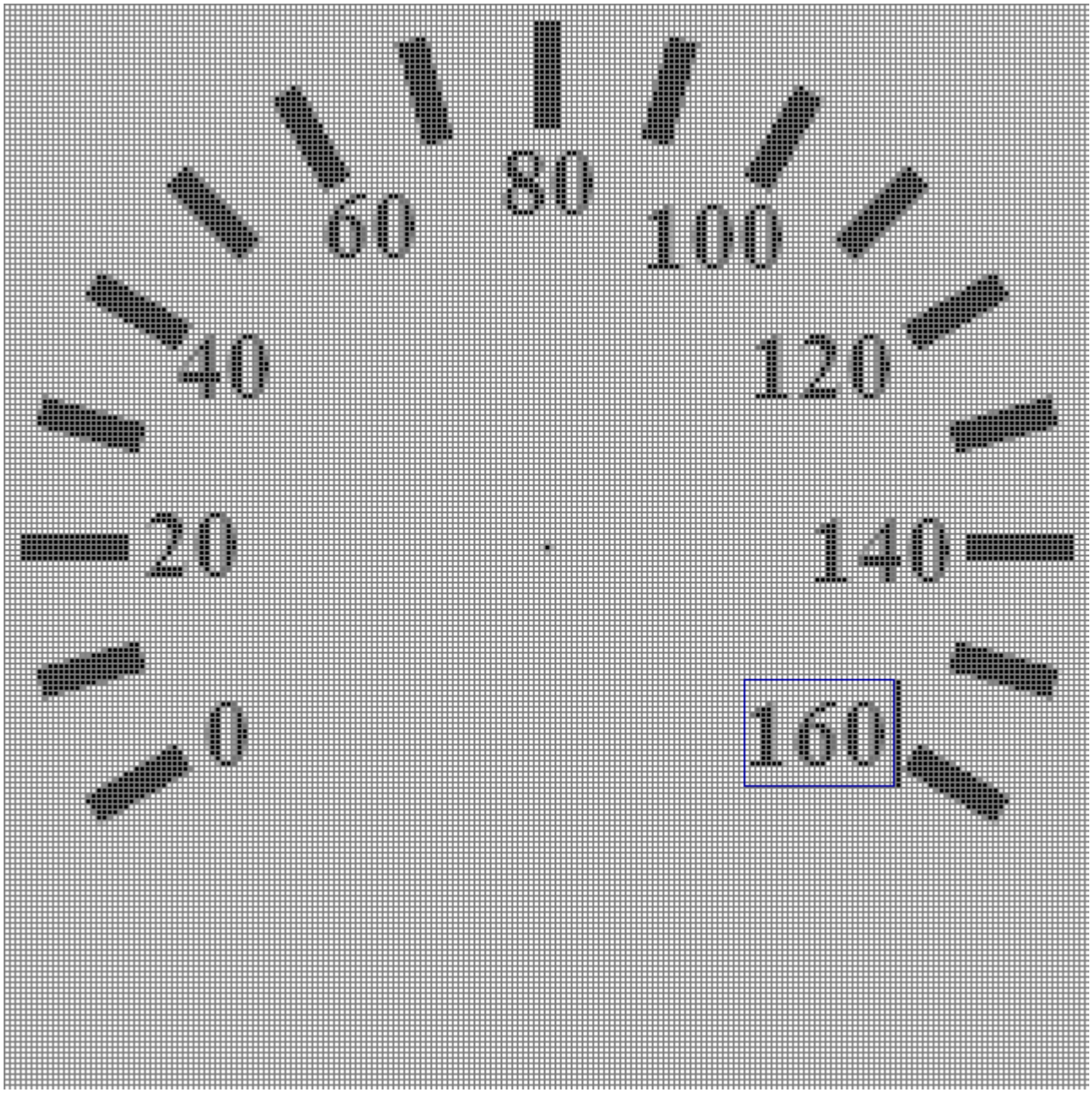
In both cases this creates an empty blue frame where you can write the numbers and move them in position with the mouse, and automatically changes the ***Master Font Selector*** to the ***Virtual Keyboard***:

Press ***Virtual Keyboard*** to set a font height:

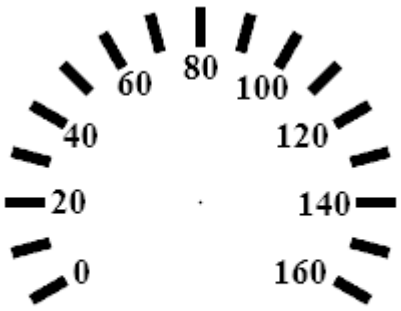


Press **OK**.

After each number is in place click the ***Text*** button twice to start a new frame:



In normal size:

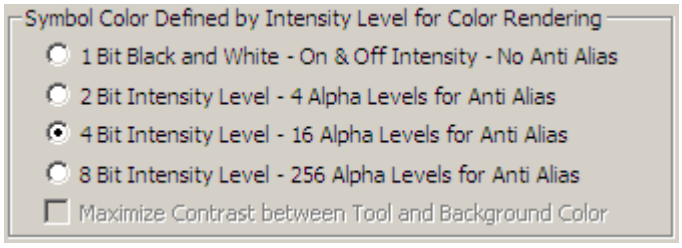


2: Save only the Alpha Channel with Reduced Memory Footprint

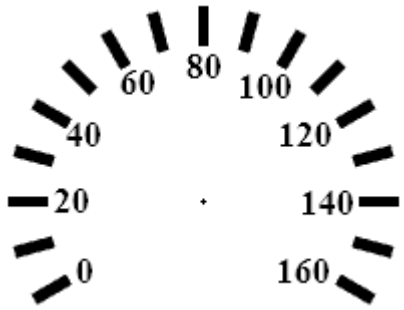
32 bits per pixel is overkill for this symbol, 4 bits is usually enough.

Click the ***Modify*** button:

Choose 4 bit:

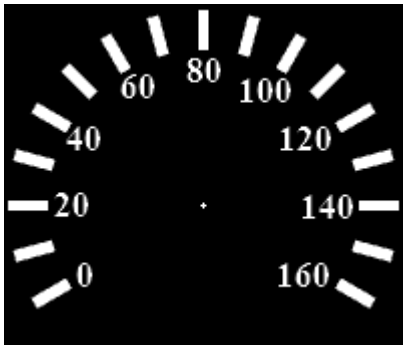


Press ***OK***:



Change is not really visible, but memory footprint is reduced by a factor 8.

Finally switch to white on black background with ***Invert Black & White***:



The speedometer is ready, press ***Save Dataset As*** to save as *Speedometer_160.c*:

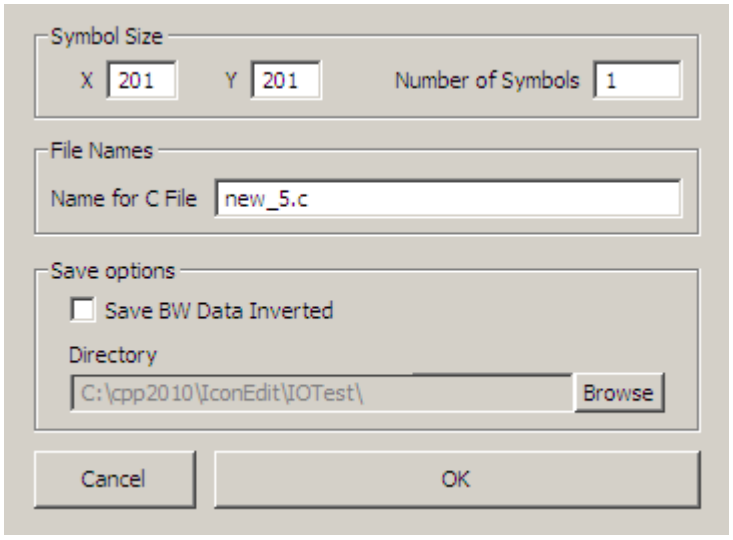
Finished in color mode.

3: Save from IconEdit with a Black & White license

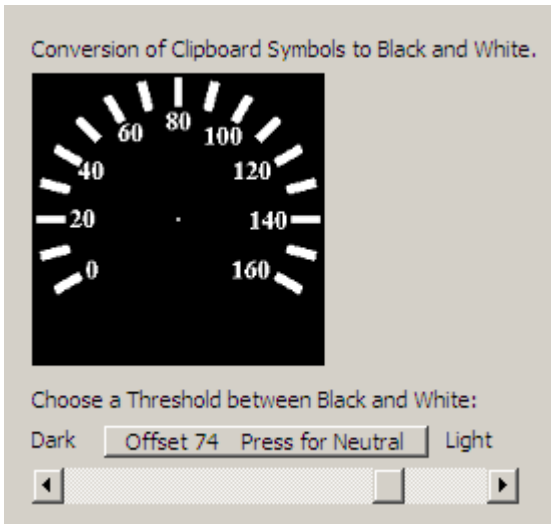
Copy to ClipBoard with

Start IconEdit B&W.

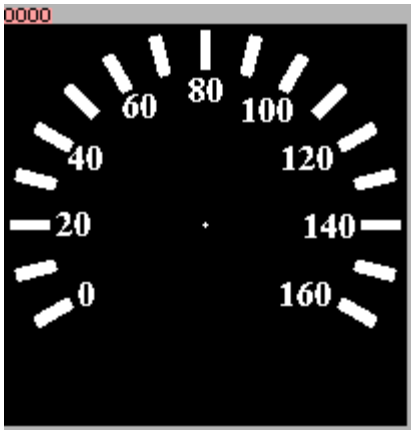
Paste from Clipboard as New



Press ***OK***



Choose ***Offset*** for best look and press ***OK***.



Save As normally.

F: How to Design a Screen Preview with Library Fonts

In this example wee will make a small speedometer screen with a look like this:

Vertical Speed

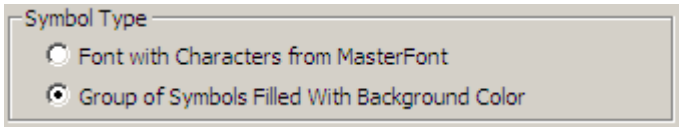
0.82 m/s

For that wee will use the two library fonts *T_16x16_b_g2* and *T_32x32_b_g2*, use the normal open function to load the two fonts:

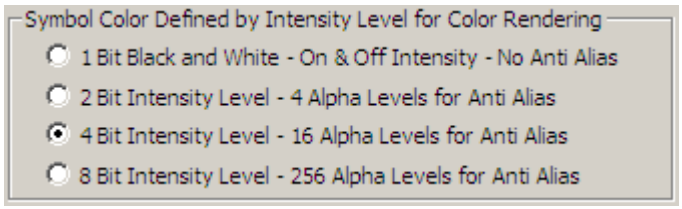
To make a new symbol press the ***New Font or Symbol Group*** button in the Main Toolbar:

This opens the create dialog box:

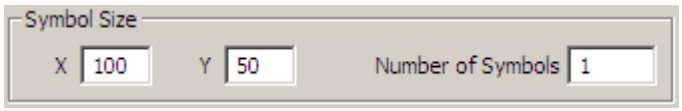
Choose how characters or symbols should be organized and shown initially:



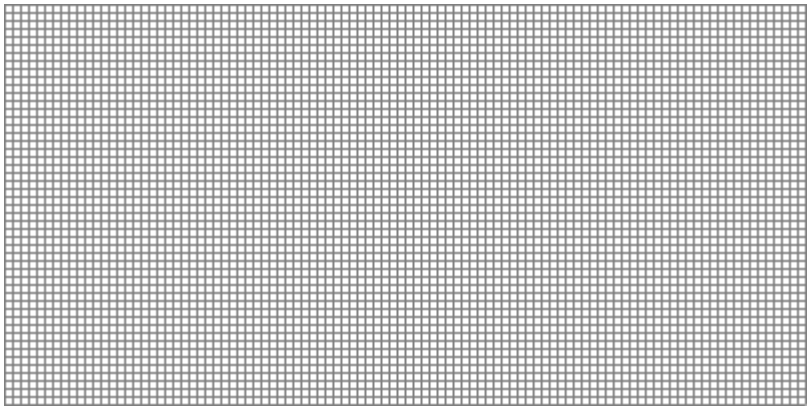
Choose the color format of the target display:



Choose the size of the display:



Press **OK** to get an empty screen:

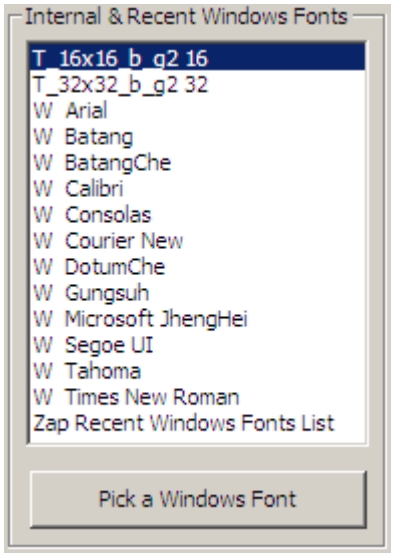


Click the ***Text in Frame*** button

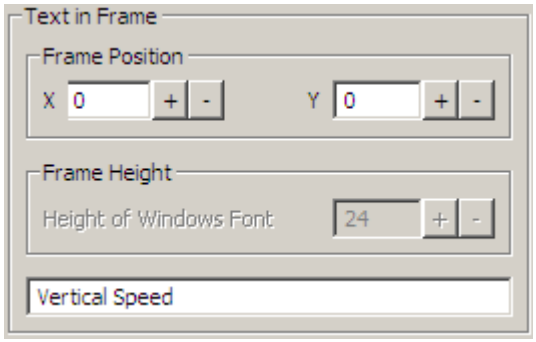
This automatically changes the ***Master Font Selector*** to the ***Virtual Keyboard***:

Open the ***Virtual Keyboard***:

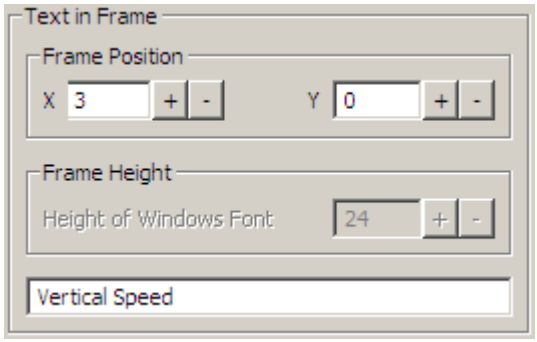
Choose the 16x16 font for the top line:



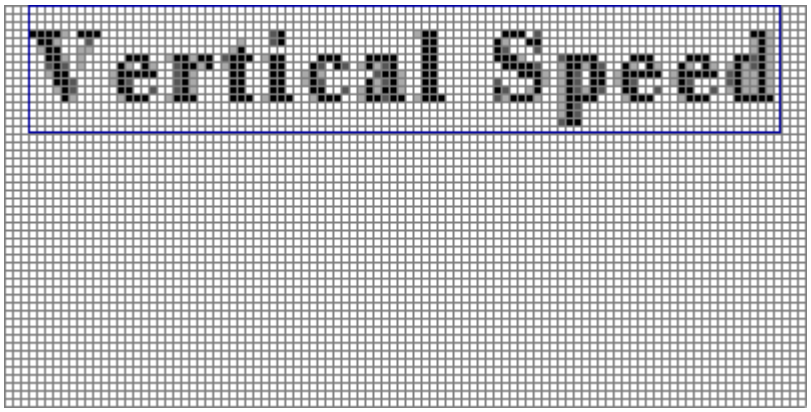
Write or paste the top line text:



Move the text to the center:



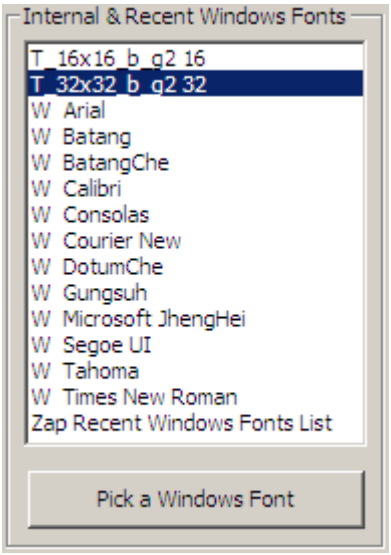
Press **OK**:



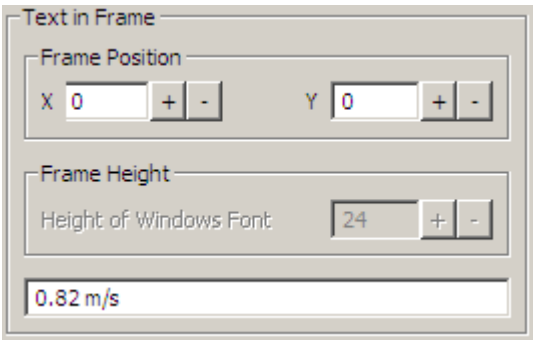
Click the ***Text in Frame*** button to finish the first line, and again to start the next line:

Open the ***Virtual Keyboard***:

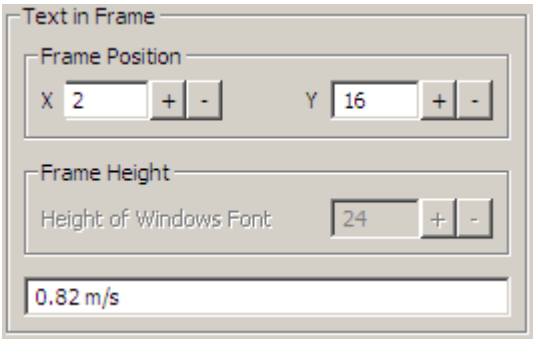
Choose the 32x32 font for the bottom line:



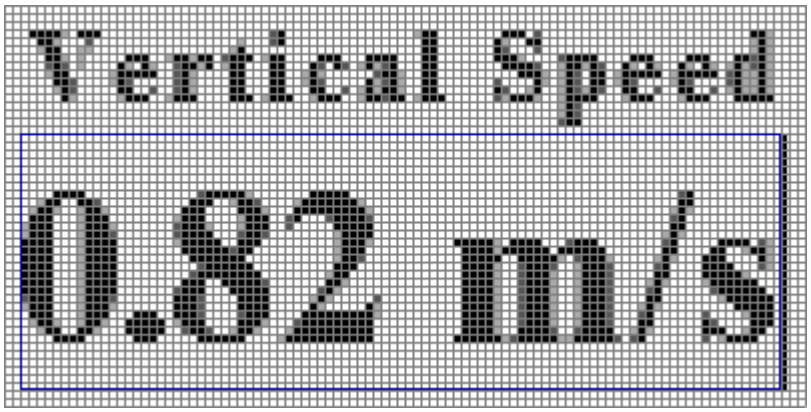
Write or paste the bottom line text:



Move the bottom line in place:



Press **OK**:



Click the ***Text in Frame*** button to finish the second line:

Your screen preview is finished, and can be saved as an image *ScreenPreview.png* with ***Image->Save Character or Symbol in Edit Window as PNG File***

09: How to Import and Convert Images

A: How to Convert Several Images to a Single Symbol File

This example applies fully to the color version of IconEdit, but only the parts about opening, adding and resizing applies to the black and white version.

The conversion of digital pictures to RAMTEX sym format can be done in many different ways; this method is primarily for changing a group of different bitmaps with the same aspect ratio into RAMTEX color or black and white images of the same size and with a common color resolution.

First open the Bitmap *Violet.bmp*.

Press the ***File Open*** button in the Main tool bar:

Find the file, and press open. This generates a group of one symbol:

Subsequently added bitmaps will have the same size and color resolution as the first. To avoid losing information during the import of these bitmaps, change the color resolution of the first bitmap to the maximum resolution for bitmaps.

In the color version of IconEdit open the ***Modify Existing Data Set*** dialog box and select 24 bit RGB:

Press OK.

Then import the other bitmaps Red, White and Yellow, to this group using:

Image -> Import and Add Image to Symbol Group

The bitmaps now look like this:

The images now all have the same size and color resolution, but they will take up almost a whole Mega byte of memory. This can be reduced by a third by changing to 8 bit color RGB organized as 3,3,2.

1: Reduce Memory Consumption to 8 bit per Pixel - RGB

Press the ***Modify Existing Data Set*** button in the Main tool bar to get the dialog box.

Select 8-bit RGB:

Press OK.

This reduces the number of possible colors to 256 fixed colors, they are fixed because there are only these 256 colors in the 3,3,2 RGB format.

2: Reduce Memory Consumption to 8 bit per Pixel – Optimized Palette

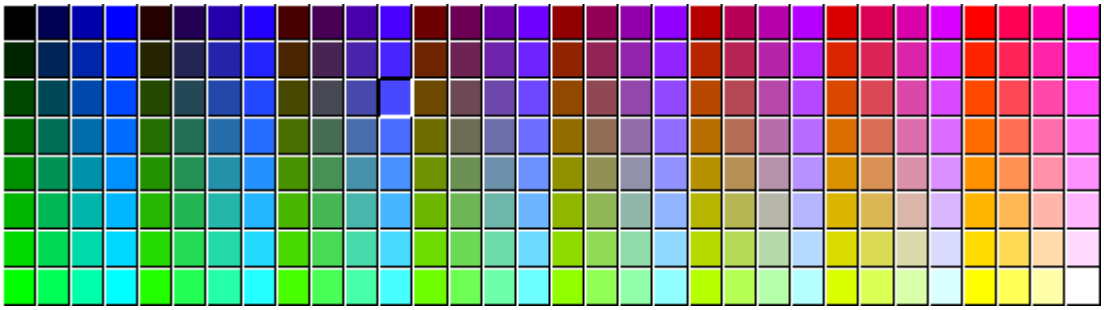
A better color approximation to the original can be achieved by making an optimized palette for this group of images. The palette will need less than one kilo byte of memory for 256 colors that are optimized for this group of images:

Press ***UnDo*** and open the ***Modify Existing Data Set*** dialog box again:

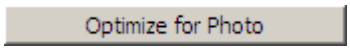
Select 8 bit color palette:

To optimize the palette press:

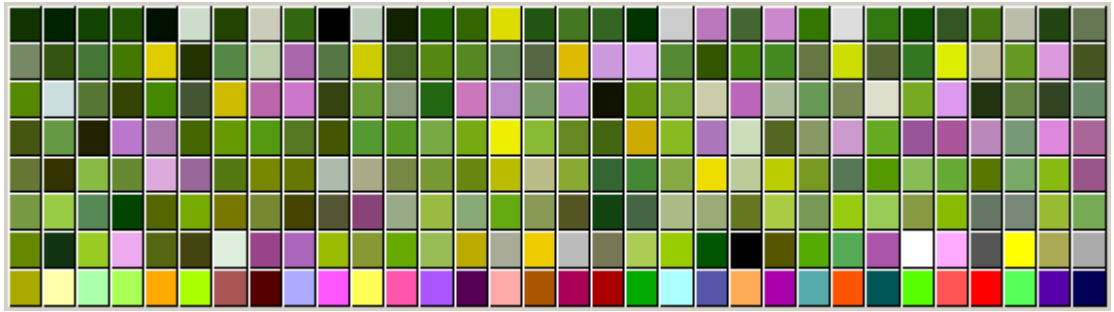
This gives the same 256 default colors as the 8 bit RGB:



Press:



This generates a much larger palette internally in the program, converts all pixels to indexes in this palette, sorts all pixel indexes according to frequency, and picks the most common. Then there is a check for “lost” color areas in the palette, and important but rare colors that lie far from all the others are added:



Press OK.

Press OK.

This is lossy compression. It sacrifices color precision but, unlike the normally used JPEG, this method does not create a halo around sharp edges or loose small details.

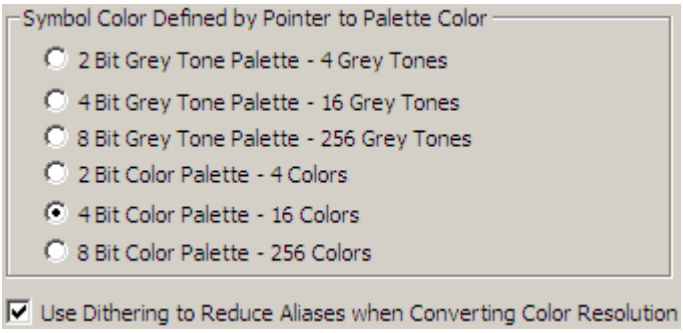
The colors and the pointers to the colors for each pixel are stored in the data format, so they do not need to be unpacked by the CPU before use thereby making the rendering on the target display simpler and faster:

3: Reduce Memory Consumption to 4 bit per Pixel – Optimized Palette with Dither

A half as big memory footprint can be achieved by a smaller palette with dither:

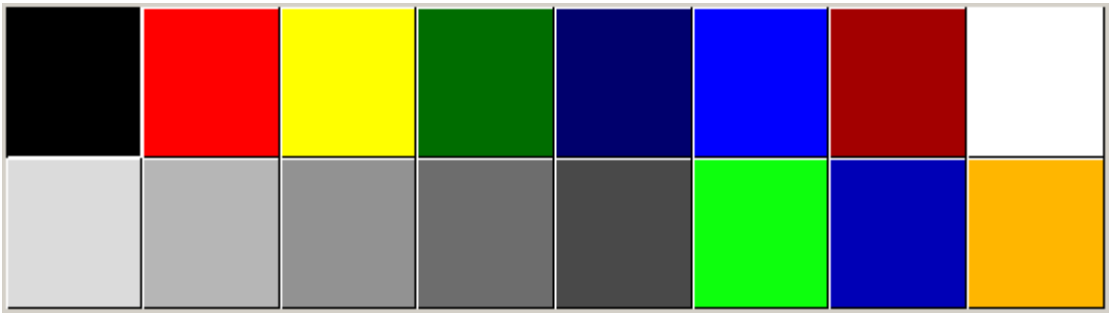
Press ***UnDo*** and open the ***Modify Existing Data Set*** dialog box again:

Select 4 bit color palette and activate dither:



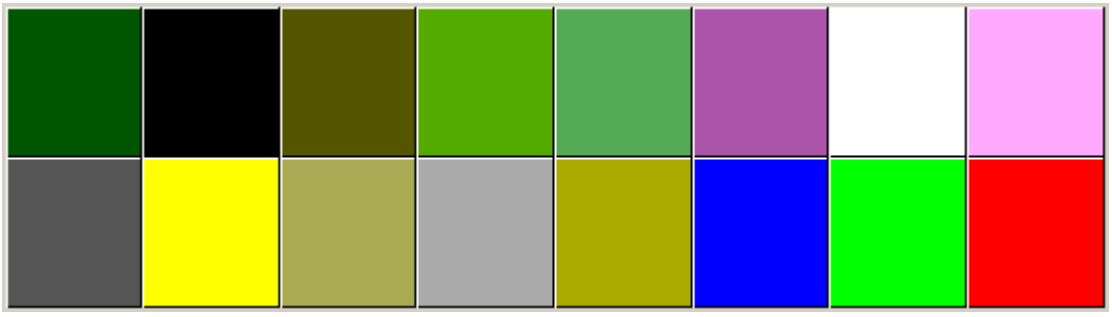
To optimize the palette press:

This gives 16 default colors:



Press:

This generates a much larger palette internally in the program, converts all pixels to indexes in this palette, sorts all pixel indexes according to frequency, and picks the most common:



Press OK.

Press OK.

This is lossy compression with diffused colors. It sacrifices color precision and makes a slightly grainy image, but, unlike the normally used JPEG, this method does not create a halo around sharp edges or loose small details.

The colors and the pointers to the colors for each pixel are stored in the data format, so they do not need to be unpacked by the CPU before use thereby making the rendering on the target display simpler and faster:


0000
0001
0002
0003

B: How to Import a Large Picture for Reduction from the ClipBoard

This example applies to both the black & white and the color version of IconEdit.

Start Paint, and open the *Decoration.jpg* picture by pressing Open in the File menu.

In Windows 8 Paint is called mspaint.exe and situated in Windows\SysWOW64 or Windows\System32.

Select all with Ctrl+A and put the picture on the Clipboard with Ctrl+C:

 *

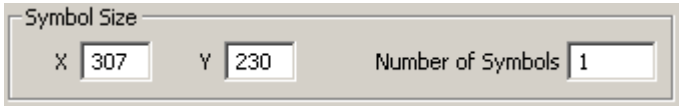
Wait for Paint to finish.

Start IconEdit.

Press ***Paste from ClipBoard asNew Font or Group***:

This opens a create dialog box:

Reduce it in size to 307x230:



Press OK.

Your Picture is ready for saving....

Press the ***Save All As…*** button in the Main tool bar to save the pixel data in the *Decoration.c* file in the proper directory.

C: How to Import and Reduce a QR Code Image

This example applies to both the black & white and the color version of IconEdit.

QR Codes are two dimensional barcodes developed by Denso Wave. There are many on-line QR Code generators that can make and save QR Code as a PNG image. This example assumes you already have done that and will show you how to turn it into a symbol file.

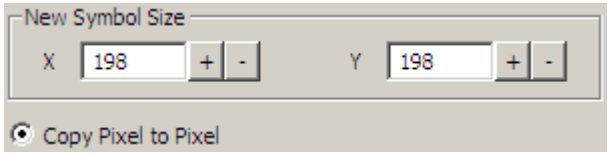
Press the ***File Open*** button in the Main tool bar:

Find the file *qrcode.png*, and press open. This generates a group of one symbol:

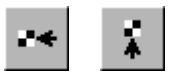
In this case each dot is 6x6 pixels; on a display that by itself have large pixels this is not necessary, 1x1 should be enough even on a normal PC screen.

The symbol is 200x200 which is not dividable with 6

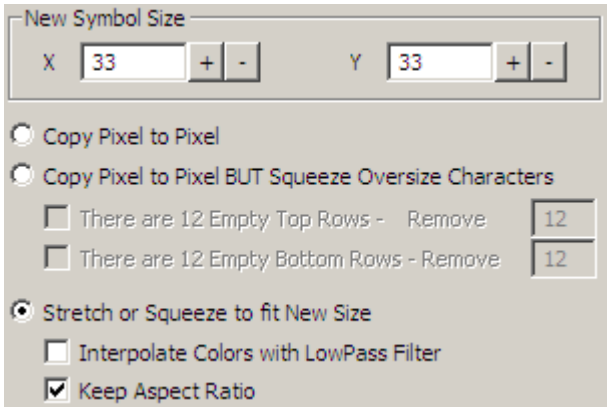
Press ***Resize All*** to reduce the symbol 198x198:



Press ***OK***. Mark the symbol and use the ***Rotate*** buttons to move the QR Code 1 pixel left and up to make the frame symmetrical:



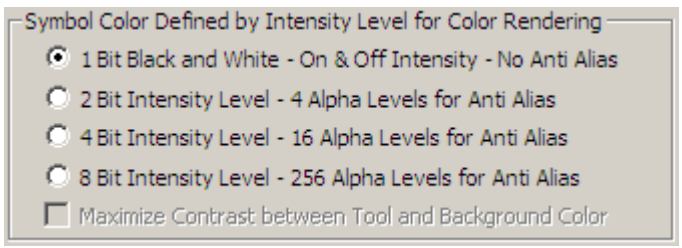
Press ***Resize All*** to reduce the symbol 33x33:



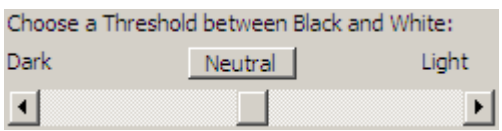
Press ***OK***.

The PNG format support full 32 bit color, but QR Codes should be black & white with 1 bit per pixel.

Press ***Modify Color*** and change to Black and White:



Press ***OK*** and choose ***Neutral***:



Press ***OK***.

Your QR Code is ready for saving....

Press the ***Save All As…*** button in the Main tool bar to save the pixel data in the *qrcode.c* file in the proper directory.

10: How to Display Asian Languages on Small Embedded Systems

This chapter demonstrates problems with and solutions for writing Asiatic texts on simple displays and it applies to both the black & white and the color version of IconEdit.

A large number of alphabets are defined in Unicode as basic characters and diacritics that have to be combined by the rendering engine at the time of displaying the text. This requires many very special and language specific runtime rendering and layering rules such as multi-symbol overlaying, special symbol specific x,y positions offset adjustments, etc, etc. In addition to that neighbouring characters with or without diacritics may form ligatures, or the characters may change form according to their position in a word.

On relatively powerful computer systems, personal computers and smart phones, this is not a significant challenge for the (usually multi core) processor, but on small embedded systems the amount of memory necessary to hold the rendering code and the processing time for the rendering would be prohibitive.

This chapter will describe different methods of circumventing this problem by using Windows build-in rendering engine for Windows fonts and storing the rendered characters or texts as symbols.

Limitation for Practical Use: *The use of Auto-generated Character symbols instead of the original combination of a basic character and one or more diacritics is only practical on embedded systems that do not have a keyboard for text input, but only predefined and static texts compiled into the embedded program.*

Limitation in Windows: *The support for different languages in Windows is under constant development and some of the methods described in this chapter only works on relatively new versions of Windows.*

If the amount of text is small each whole text could be stored as an image in each their symbol. This method would be fast to implement at the price of high memory consumption.

An alternative method for larger amounts of text is to build the necessary combined characters into a font in the Code Point range reserved by Unicode for private use or special characters and modify the text to use the combined characters. This should be done for the first time when the text strings are finished and the font is designed, and then the text modification must be repeated every time the original text is changed or updated.

The text modification can be done and saved directly as a Unicode text in IconEdit with the **Export Modified Text** command, but far greater control over the format of the modified text can be achieved by using the **String** text modifier utility supplied as a part of the IconEdit package.

The issues that IconEdit can help solve:

How to display Right to Left scripts:

Texts in Unicode are always stored in writing order from Left to Right and most basic display systems displays the characters in a text Left to Right. This means that for Left to Right scripts such as Latin, Greek or Cyrillic the text characters can be displayed one by one without problems, but for Right to Left scripts such as Arabic, Syrian or Hebrew there has to be a pre-processing text modifier that reorders the characters from writing order to display order, in the simplest cases just a mirroring.

An example in Hebrew:

Text is displayed like this in an editor:

אסטראנאמיע

but it is stored in writing order in the text file like this:

עימאנארטסא

so it has to be mirrored:

אסטראנאמיע

before it can be displayed correctly on a simple display.

How to display Arabic scripts:

Texts in Arabic are written with connected characters that change their form according to their position in a word, therefore Unicode defines up to 5 different looks (code-points) of each Arabic character, a Generic form that is stored in the text string when the text is written and up to 4 presentation forms: Initial, Medial, Final and Isolated to be used for the display. So before the texts can be displayed there has to be a pre-processor that converts the generic code-points to presentation code-points.

An example in Arabic:

Text is displayed like this in an editor:

السماوية

but it is stored in writing order in the text file as generic characters like this:

ةيواماسلا

so it has to be mirrored:

السماويةة

and then changed to presentation forms:

السماوية

before it can be displayed correctly on a simple display.

How to display Vietnamese:

Texts in Vietnamese are written as Latin characters with tonal marks, and Unicode has a special group of code-points that define the look of these combinations. Unfortunately this means that in some instances the texts are stored as basic characters followed by tonal marks, and in other instances the texts are stored as the special Vietnamese characters. So before the texts can be displayed there has to be a pre-processor

that identifies eventual tonal marks after basic characters and converts them to the special Vietnamese characters.

An example in Vietnamese:

Text is displayed like this in an editor:

nghệ̣của

but it might be stored as in keyboard sequence in the text file with separate diacritics like this:

nghệ.cu ̉a

so it has to be combined:

nghệ̣của

before it can be displayed correctly on a simple display.

How to display Southern Asiatic scripts:

Texts in most Southern Asiatic languages are written and stored as basic characters followed by a number of diacritics or tonal marks; before the text can be displayed these have to be combined to form actual characters or ligatures. Unicode only defines the generic look of all these code-points and leaves the, often very complex, combination of these to the rendering engine. One way around this problem for small systems is to have a pre-processor that identifies the code-point combinations that lead to these actual characters or ligatures, create them on the development system, usually Windows, store them at new code-points, and change the texts to use the new code-points.

An example in Devanagari:

Text is displayed like this in an editor:

विकिपीडिया

but is stored as a keyboard sequence with diacritics in the text file like this:

वि◌ कि◌ पी◌ डि◌ या

so it has to be combined and rearranged:

विकिपीडिया

before it can be displayed correctly on a simple display.

The method IconEdit can offer:

When a master text or ‘C’ code file is read into IconEdit it creates or updates a font to fit the text or the text strings in the ‘C’ code in the master. As part of this process the master is modified with the above mentioned pre-processors to make it displayable by simple Left to Right display systems, and it is the

characters needed for the display that is added to the font. The modified master can then be saved under a separate name and used by the display system to show the text or text strings in the master.

Warning: This process creates a linked set of 5 files: *Master.h* that is read into IconEdit to create *Font.sym*, *Font.cp* and *Font.c* and the modified version of the master *Modified.h*, so it is very important that these 5 files are considered an undividable unit and are stored and backed up together.

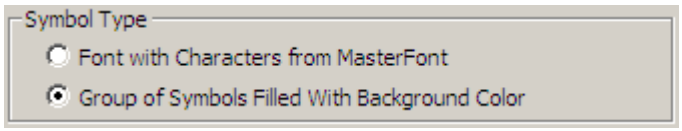
A: How to make an Image of a Text

In this example we will use the Devanagari text "खगोल शास्त्र " ("astronomy") to build an image of that text.

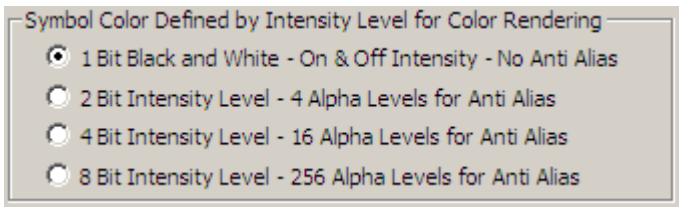
To make a new symbol press the ***New Font or Symbol Group*** button in the Main Toolbar:

This opens the create dialog box:

Choose how characters or symbols should be organized and shown initially:

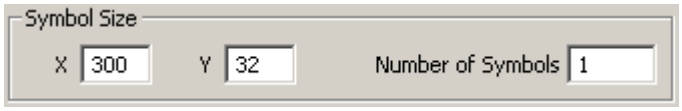


Choose a color format, here Black & White. The other options are only available in the color version of IconEdit:



The master font is already a Windows Unicode font, but we should change that later.

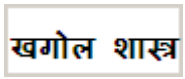
The size has to fit the text, x size is a guess that can be changed later:

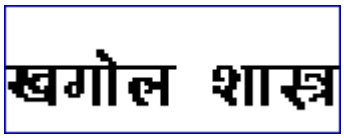


Press OK and turn pixel grid off for clarity:

Copy the text खगोल शास्त्र to the clipboard and paste it in IconEdit with ***Paste from ClipBoard***:

The text is shown in the paste dialog box:

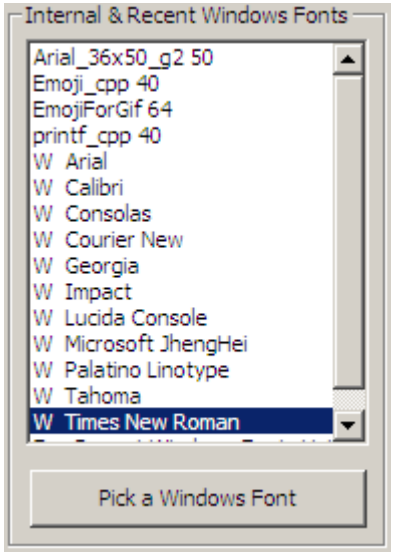




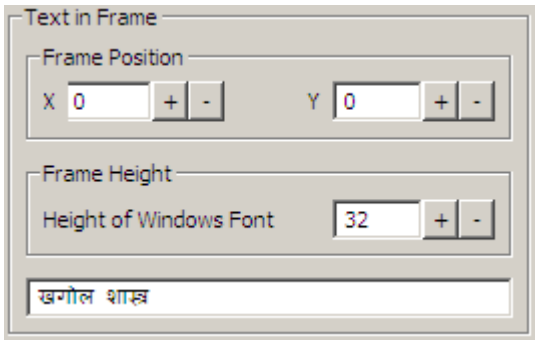
This automatically changes the ***Master Font Selector*** to the ***Virtual Keyboard***:

Maybe change the look of the text with the ***Virtual Keyboard*** button:

Pick a master font look:



Change size and position or edit the text:



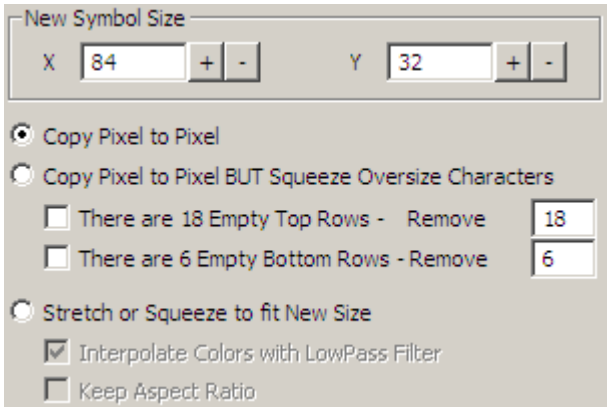
Press **OK** when finished.

Maybe also change the text writing modes **anti-alias**, **normal**, **semi transparent anti-alias**, **semi transparent normal** with right click on the already pressed write text button:

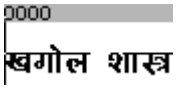
The frame is 84 x 32 pixels, to reduce the size of the symbol change to **Group Edit:**

Press the **Resize All Symbols** button:

Set size and copy option:



Press OK:



Symbol 0000 is ready to save

Press the ***Save All As…*** button in the Main tool bar to save the data as *AstronomyImage.c* in the proper directory.

B: How to Combine Characters and Diacritics or Surrogates as New Symbols

IconEdit can identify combined characters in a text file for a large number of scripts and build a set of symbols representing the combined character and then put the Auto-generated Presentation Character as a new symbol at a Code Point in the private area Unicode range.

Warning: *Emojis and other surrogate characters are only supported on Windows version 8 and newer.*

1: Plain Text with Auto-generated Composed Characters

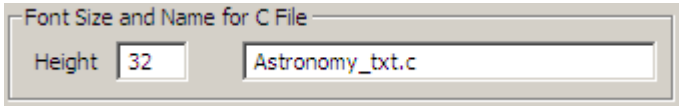
In this example we will use a long Devanagari text about astronomy ("खगोल शास्त्र ") to build a small font for the text.

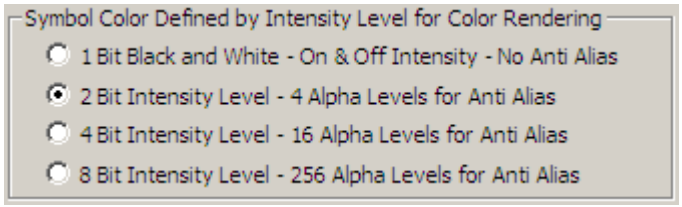
Press the ***File Open*** button in the Main tool bar:

Open the Unicode text *Astronomy.txt*:

खगोलशास्त्र (अंग्रेज़ी:Astronomy) विज्ञान के एगो शाखा हउवे। एम्मे आकाश की ग्रह नक्षत्र आ आकाश गंगा आदि क पढ़ाई आ रिसर्च होल। प्राचीन काल में एहमें ग्रहन क गति आ आकाश में स्थिति क पूर्वानुमान लगावे का भी अध्ययन होखे।

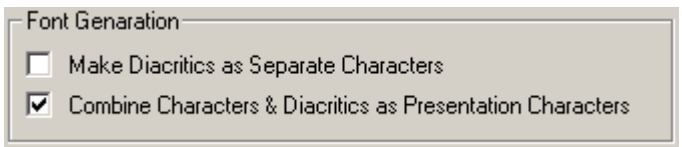
Choose size and color mode:





Press **OK**.

The text file is analyzed for possible Auto-generated Characters, and if any are found it is possible to choose what is generated:



Press **OK**.

The combined Auto-generated Presentation Character symbols are placed as Code Points at unused positions in the 0xE000 to 0xF8F0 private range, in this case from 0xE700 to 0xE723:

The text is displayed automatically with the new font and blue marks to identify the characters individually:

खगोलशास्त्र (अंग्रेज़ी:Astronomy) विज्ञान के एगो शाखा हउवे।
एम्मे आकाश की ग्रह नक्षत्र आ आकाश गंगा आदि क पढ़ाई आ
रिसर्च होल। प्राचीन काल में एहमें ग्रहन क गति आ आकाश में
स्थिति क पूर्वानुमान लगावे का भी अध्ययन होखे।

The Font is ready to save

Press the ***Save Dataset As…*** button in the Main tool bar to save the font as *Astronomy_txt.c* in the proper directory, the modified text should be saved separately, see below. The Sym and Code Point Character List files and the Auto-generated Character Alias list in the Code Point Character List files saved automatically.

If ***Save Modified Text Automatically when the Font is Saved*** is activated you can also save the modified text as *Astronomy_Modified_txt.txt* or any other name.

After the font has been created the text can be viewed and modified to use the Auto-generated Presentation Characters instead of the original combination of a basic character and one or more diacritics.

The text view can be turned off and on by pressing the ***Show Imported Text for Check of Font Look*** button:

खगोलशास्त्र (अंग्रेज़ी:Astronomy) विज्ञान के एगो शाखा हउवे।
एम्मे आकाश की ग्रह नक्षत्र आ आकाश गंगा आदि क पढ़ाई आ
रिसर्च होल। प्राचीन काल में एहमें ग्रहन क गति आ आकाश में
स्थिति क पूर्वानुमान लगावे का भी अध्ययन होखे।

If not already saved, the input file can be saved as a new modified file with the menu command:

Text ->Save Modified Text with Autogenerated Presentation Characters as New Text File…

Save the new file as *Astronomy_Modified_txt.txt.* The file can be read by most text editors and compilers, but the text strings will only be displayed correctly when the font *Astronomy_txt.c* is used for the display. On normal text editors the Auto-generated Presentation Characters will be shown as blocks because they are not according to the Unicode standard:

ख□ल□□ (□□□:Astronomy) □□न □ ए□ □□ हउ□।
ए□ आ□श □ □ह न□□ आ आ□श □□ आ□ क प□ई आ
□स□ □ल। □□न □ल □ एह□ □हन क ग□ आ आ□श □
□□ क □□□□न ल□□ □ □ अ□यन □□।

2: “C” Text Catalogue with Auto-generated Composed Characters

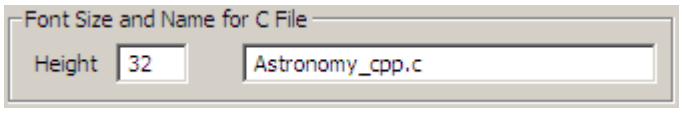
In this example we will use a small text catalogue with Arabic and Thai text strings about astronomy to build a small font for the text catalogue.

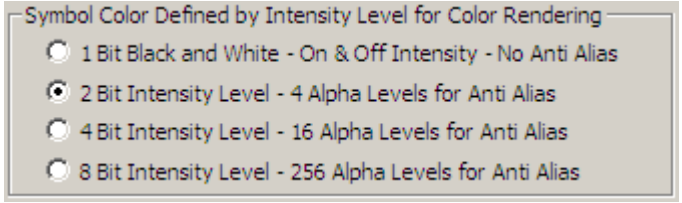
Press the ***File Open*** button in the Main tool bar:

Open the Unicode text *Astronomy.cpp*:

```
// Disclaimer: These texts are only for demonstration of the principle
// and may not make any sense to someone familiar with the language
#ifdef ARABIC
wchar_t szAstronomy_00[]={L"الدراسة العلمية للأجرام"};
wchar_t szAstronomy_01[]={L"مثل النجوم، والكواكب،"};
#elif THAI
wchar_t szAstronomy_00[]={L"ดาราศาสตร์ คือวิชาวิทยา"};
wchar_t szAstronomy_01[]={L"อาทิ ดาวฤกษ์ ดาวเคราะห์"};
#endif
```

Choose size and color mode:

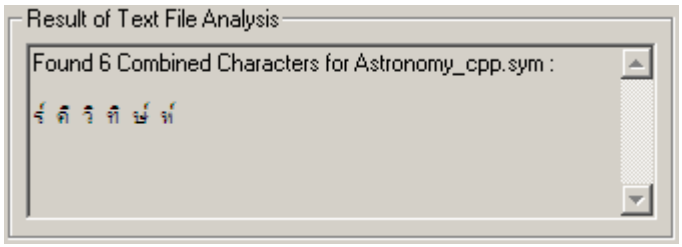


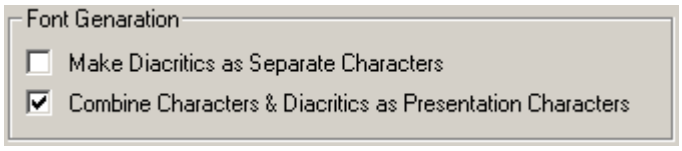


Press OK.

The text catalogue file is analyzed for Combined Characters, Presentation Characters and Mirrored Characters inside the ‘C’ strings while the surrounding code and comments are ignored.

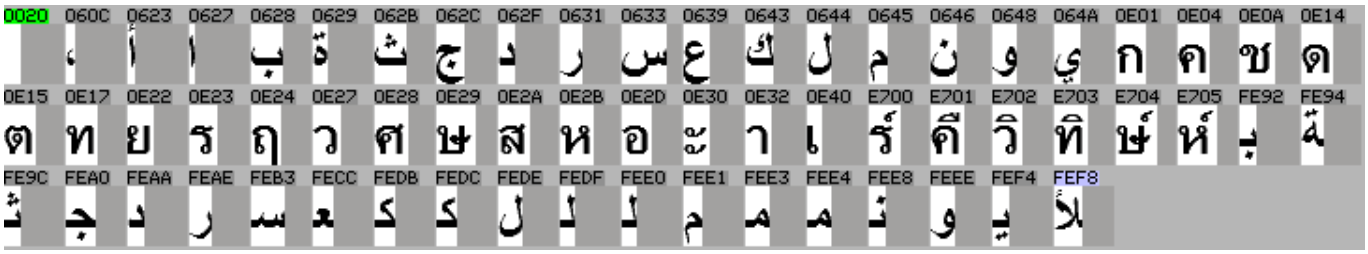
If any Combined Characters or Mirrored Characters are found it is possible to choose what is generated:



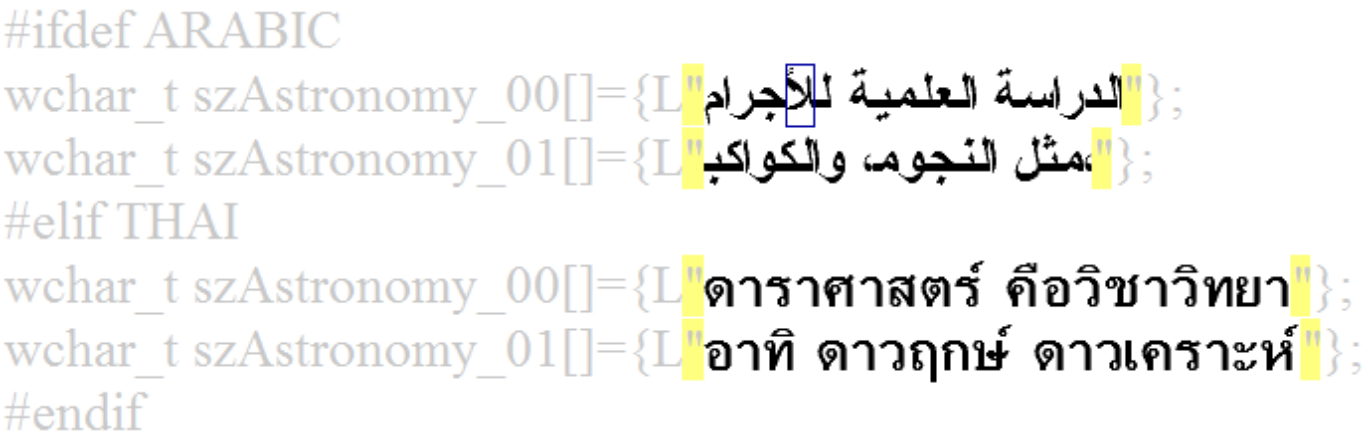


Press OK.

The combined Auto-generated Presentation Character symbols are placed as Code Points at unused positions in the 0xE700 to 0xF8F0 private range, in this case from 0xE700 to 0xE705. The Unicode Presentation Characters, if any, already have Unicode values in the range 0xFE00 to 0xFEFF:



The texts in the strings are displayed automatically with the new font and a blue mark to identify the characters individually:



The Font is ready to save

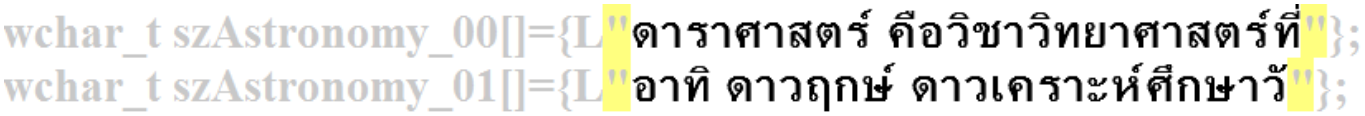
Press the ***Save Dataset As…*** button in the Main tool bar to save the font as *Astronomy_cpp.c* in the proper directory, the modified text should be saved separately, see below. The Sym and Code Point Character List files and the Auto-generated Presentation Character Alias list is saved in the Code Point Character List file automatically.

If ***Save Modified Text Automatically when the Font is Saved*** is activated you can also save the modified text as *Astronomy_Modified_cpp.cpp* or any other name.

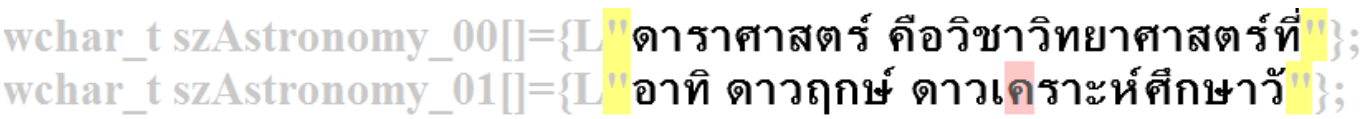
After the font has been created the text catalogue can be viewed and modified to use the Auto-generated Presentation Characters instead of the original combination of a basic character and one or more diacritics.

The text string viewed can be turned on and off by pressing the ***Show Imported Text for Check of Font Look*** button:

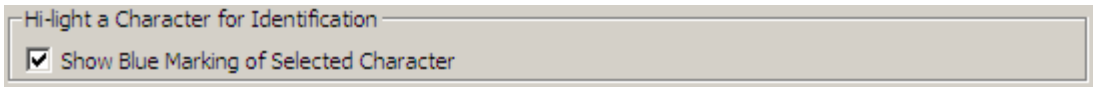
The view has light-grey code and black strings enclosed by yellow double quotes:



If any characters, presentation characters or diacritics are missing in the font they are shown by the chosen Windows font with pink background. Please note that diacritics always appear after the main character in a text string even if they are supposed to be placed in front of the main character:



Characters can be identified by blue marks, right click ***Show Imported Text*** and Choose ***Highlight***



Press **OK**, and click a character.

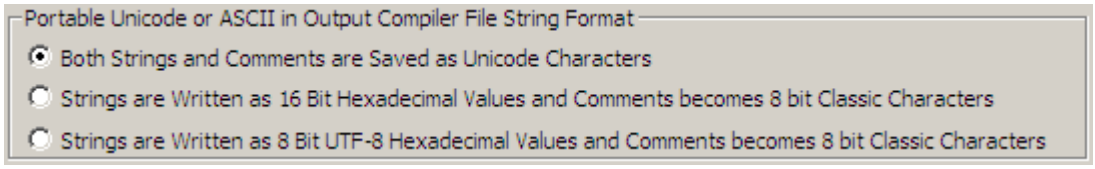
If not already saved, the input file can be saved as a new modified file with the menu command:

Text ->Export Modified Text with Autogenerated Presentation Characters as New Text File…

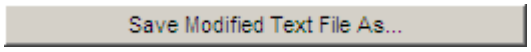
This opens a new window with the modified text and the string format and save dialog box:

```
#ifdef ARABIC
wchar_t szAstronomy_00[]={L"الدراسة العلمية للأجرام"};
wchar_t szAstronomy_01[]={L"،مثل النجوم، والكواكب"};
```

Normally 16-bit Unicode is fine, but for very old compilers you may need one of the other formats:



Choose a portable format and press **Save Modified Text File As…**.



Save the new file as *Astronomy_Modified_cpp.cpp*. The file can be read by most text editors and compilers, but the text strings will only be displayed correctly when the font *Astronomy_cpp.c* is used for the display.

A list of comma separated values that contain the Auto-generated Presentation Characters and the hexadecimal Code Point of the corresponding symbols as ‘C’ strings can be saved for documentation.

Text -> Export Autogenerated Presentation Character Alias List as CSV File…

Save file as *Astronomy_cpp.csv*. The file has this form and can be read by most text editors and spreadsheets:

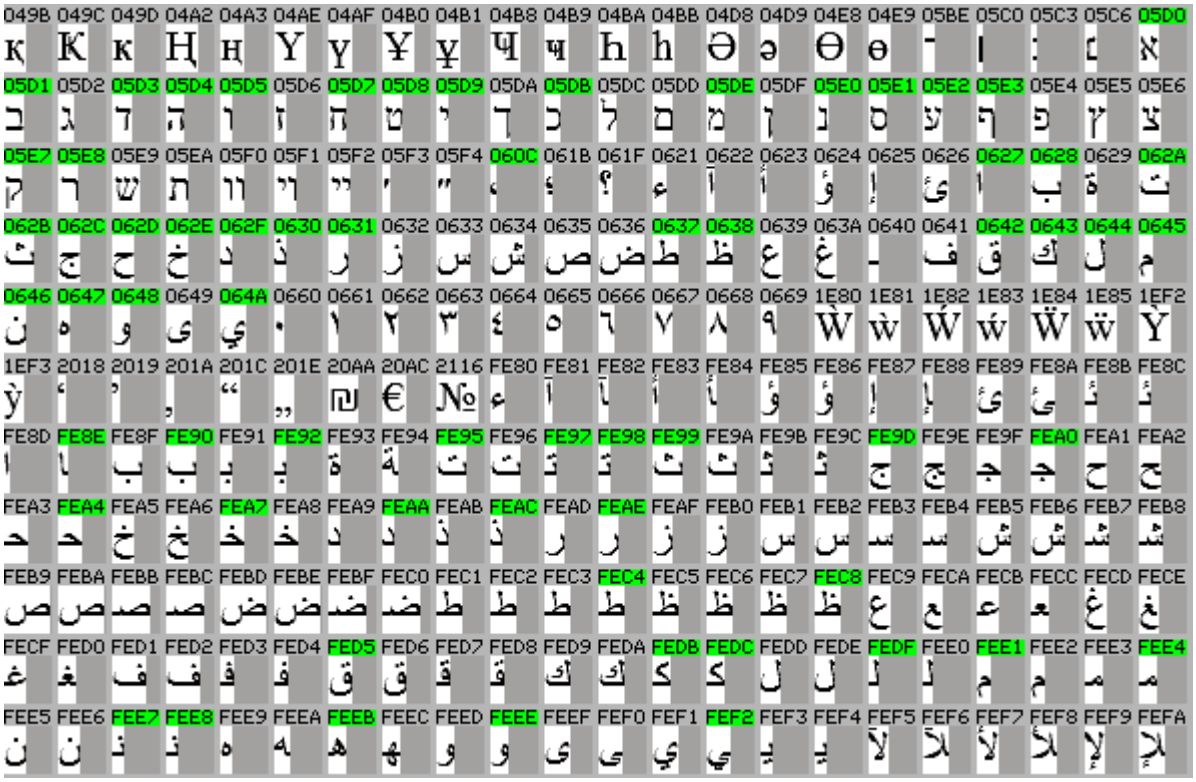
```
"Combined Character";"Hexadecimal Replacement"
"ร์";"\xE700"
"ค์";"\xE701"
"ว์";"\xE702"
"ท์";"\xE703"
"ท์";"\xE704"
"ษ์";"\xE705"
"ห์";"\xE706"
"ค์";"\xE707"
"ว์";"\xE708"
```

C: How to Use Arabic Presentation Characters

In this example we assume a font with Arabic characters already exists and that it is opened. Such fonts can be found included European Fonts library.

Open the text file *Astronomy.cs* with **Text –> Import Text or Text Catalogue to Mark or Create Characters**:

The characters found in the texts will be highlighted in green in the font:



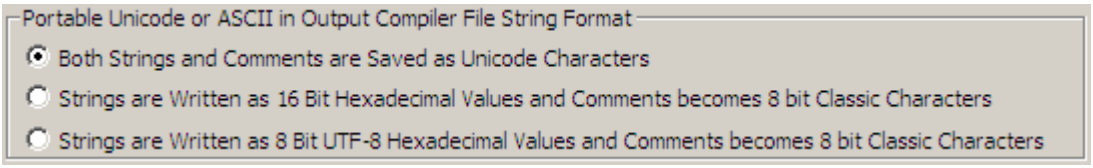
The text is now run through both pre-processors and shown as most basic display systems that displays the characters in a text Left to Right would show it:

The text is shown automatically with the font. Press **Show Imported Text for Check of Font Look** to hide or show the text:


```
#if ARABIC
string szAstronomy_00 = "والظواهر التي تحدث خارج نطاق ";
string szAstronomy_01 = "النجوم، والكواكب، والمذنبات، والمجرات";
#elif HEBREW
string szAstronomy_00 = "היא ענף במדעי הטבע החוקר";
string szAstronomy_01 = "כוכב ו -נומוס - חוק";
#endif
```

The pre-processed text can be saved with the command **Text -> Export Modified Text with Autogenerated Presentation Characters as New Text File…**

This opens the string format dialog box:



Choose a portable format and press **OK**.

Save a new code file as *Astronomy_Modified_cs.cs*

Complication: If the pre-processed text in *Astronomy_cs.cs* is read into a text editor it will trigger the text editors own pre-processors and the text will be garbled, therefore *Astronomy_cs.cs* should only be used for compilation and the original *Astronomy.cs* should be kept and used for future modifications to the text.

D: How to Maintain Linked Texts and Fonts

In this example we assume the linked set of 5 files *Master.h*, *Font.sym*, *Font.cp* and *Font.c* and the modified version of the master *Modified.h*, already exists.

If the pre-processed text in *Modified.h* is read into a text editor it will trigger the text editors own pre-processors and the text may be garbled or un-displayable, therefore any changes to the texts should be done in the *Master.h* file.

An example of how a modified text can appear in a normal text editor:

Master text file in normal text editor:

L"אסטראונאמיע"
L"السماوية"
L"nghệcủa"
L"आकाशकी"
L"ดาวฤกษ์"

Modified text is garbled or un-displayable in normal text editor:

L"עימאנארטסא"
L"ﻪﻴﻮﻤﺴﻟا"
L"nghệcủa"
L"आ□श□"
L"ดาวฤก□"

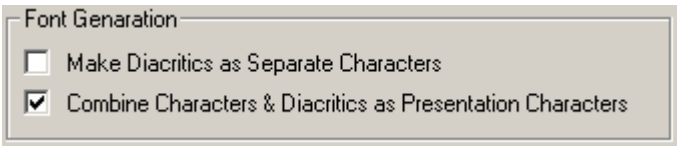
The Hebrew text is mirrored.
The Arabic text is mirrored and alternative presentation characters are use.
The Vietnamese text appears unchanged, but is now stored as pre-composed characters.
The Devanagari and Thai texts consists partly of composed characters placed in the Unicode private area and can only be displayed correctly with the linked font. The text editor will display the correct Code Points, but the looks of these are not defined by Unicode, so the default symbol for undefined characters is displayed for each private character instead.

Updating the font:

When the *Master.h* file is updated start IconEdit and open *Font.c* with **File -> Open…**

Then open the text file *Master.h* with **Text –> Import Text or Text Catalogue to Mark or Create Characters…**

If there are new additional characters in *Master.h* the **Choose Autogenerated Characters** dialogbox may pop up, choose **Combine Characters**:



When and if the question **Insert New Additional Characters** pops up answer **Yes** and save the modified font with **File -> Save in C format**.

The new pre-processed text can then be saved with the command **Text -> Export Modified Text with Autogenerated Presentation Characters as New Text File…** as a new code file *Modified.h.*

E: How to Make and Synchronize Extra Linked Fonts

In this example we assume the linked set of 5 files *Master.h*, *Font.sym*, *Font.cp* and *Font.c* and the modified version of the master *Modified.h*, already exists.

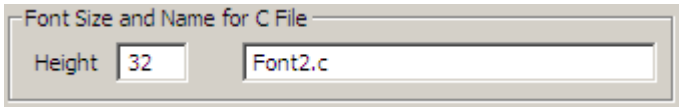
If you want to use several fonts with the same modified text all characters including presentation characters, pre-composed characters and combined characters have to be in prefect synchronism. These extra fonts are not exact copies of each other, but their Code Point Character Lists have to be identical; for lack of a better word let's call them siblings.

1: How to Create an Extra Font with an Alternate Look or Size for a Linked Set

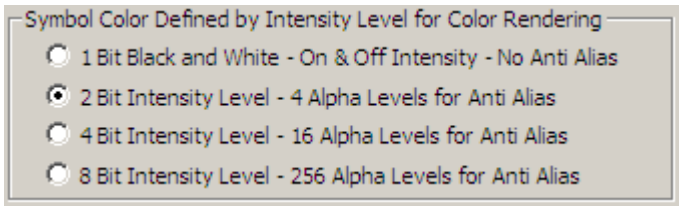
However different a new additional font should look it has to have the same Code Point as the other fonts in the linked set, so we start with that.

Open the Code Point file *Font.cp* with **File –>Import Code Point Character List to Create a Text Optimized Font…**

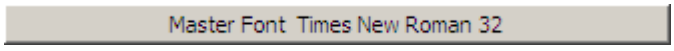
Choose **Size** and **Name**:



Choose a color format:



And maybe another master font:



Press **OK**.

Save the new extra font as *Font2.c* with **File -> Save in C format**.

2: How to Update Extra Fonts to Keep Synchronism between the Fonts

When the *Font.c* file is updated start IconEdit and open *Font2.c* with **File -> Open…**

Then open the Code Point Character List file *Font.cp* with **File –>Import Code Point Character List to Mark or Create Characters…**

When and if the question **Insert New Additional Characters** pops up answer **Yes** and save the modified font as *Font2.c* with **File -> Save in C format**.

F: How to Include Non-Unicode Characters in Texts

Unicode characters cover almost all living languages in the world, but you may need some that are not defined by Unicode. In this hypothetical example we assume that you need an inverted **u** in the text “Empty Your C**u**p”.

Unicode provides a large area for Private Characters starting at Code Point E000, and that is where we will put **u**.

Change the original string in the file *Cup.h*:

wchar_t Cup[]={“Empty Your Cup”};

to:

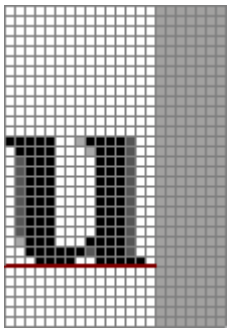
wchar_t Cup[]={“Empty Your C\xE000p”};

and import *Cup.h* as font:

To get this font:

And this text:

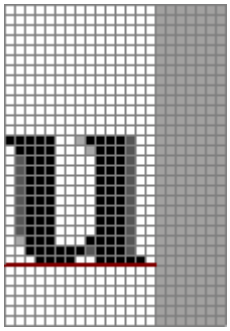
Right click u to edit the character and place a base line for later use:



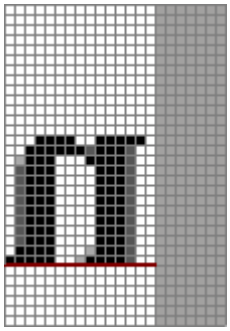
Copy med ***Copy to ClipBoard***

Change to the Private Character E000 and paste with ***Paste from ClipBoard***

:



Mirror and Rotate to the base line:

To get this text:

```
wchar_t Cup[]={"Empty Your Cup"};
```

Save the font as *Cup_h.c*.

11: How to Use the Backward Compatibility Tools

This example applies to both the black & white and the color version of IconEdit.

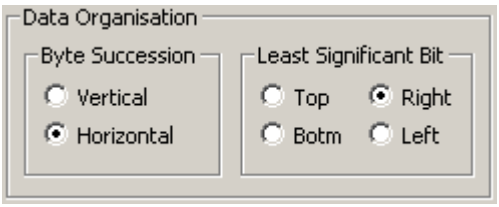
The old programs LcdIcon, LcdColor and FontEdit had a number of black & white formats where the Bit orientation and the Byte succession could be chosen at any possible value. These formats are all supported in the IconEdit program, but in a slightly different way.

The *Name.h* and *Name.bin* files are opened normally in the menu **File -> Open…** but now they are converted to the standard internal group format also used by the *Name.sym* files, and the menu **File -> Save in C Format** and **File -> Save in C Format As…** will now save the data as *Name.sym* and *Name.c* group files in the standard RAMTEX format.

The advantage of this change to the standard internal group format is that now the group can be converted to a font and the Code Point Character List information for the font can be included in the new extended *Name.h* and *Name.bin* formats. The original formats were meant for virtual buttons and classic 8-Bit fonts with 256 characters, and these do not require a Code Point Character List. The new extended formats are meant for 16-Bit Unicode fonts where only the necessary characters for the texts are present, and therefore require a Code Point Character List.

Data can be saved in the old formats by saving a **Group** with the **File-> Save Black&White As H File…** and **File -> Save Black&White As BIN File…**

This opens an Export Format dialog box:



Choose a Byte and Bit option and press OK.

Data can be saved in the new extended formats by saving a **Font** in a similar way the only difference is that now the Code Point Character List information is included in the file. In the *Name.h* file the Code Point is added to each symbol header. In the *Name.bin* file all the Code Points are added to the end of the file.

12: How to Use the Batch Commands

This chapter applies to both the black & white and the color version of IconEdit and demonstrates various typical scenarios.

The IconEdit batch system can generate text optimized fonts, convert output text strings to UTF-8, 16, or 32, and bulk convert a whole directory of images, symbols, or fonts to symbol format.

Parameter overview:

StartFile can be of the types .c .sym .bdf .h .cbn .bin .ief, or *.*
:Height for new files, can be almost any value, but heights under 12 are not recommended
:A (**Master Font)** add missing characters in Input Font, Master Font is optional
:B ext bulk convert to symbol format **ext** can be bmp, png, jpg, jpeg, bdf, ief, sym, .bin
:C value character width of added characters 50% to150%
:D monospace font
:E d/i/k/w+value encoding of classic 8-bit texts in InputText and OutputText
:F OutputFont is always optimized C-source code symbol format of type .c or .sym
:G value edge pixel value of added characters 13% to 187% for greytone and 58% to 148% for b&w
:H y/n yes or no to private characters in output text string as Hexadecimal, else as normal Characters
:J OutputText text fitting OutputFont, should be same type as InputText
:K keep width of InputFont in OutputFont by squeezing any larger new added characters
:M y/n yes or no to Match output text string character type, string prefix, and hexadecimal notation
:N y/n yes or no to add U to unsigned values
:O InputText optimize font for text, **InputText** can be of the types .txt .c .h .cpp .cs .cp
:P InputText expand font for text, **InputText** can be of the types .txt .c .h .cpp .cs .cp
:Q quiet, error messages only in **`%ERRORLEVEL%`** and the file ***BatchErrorLog.txt*** so no pop-ups
:R InputFilter is remove characters from font, **InputFilter** can be of the types .txt .c .h .cpp .cs .cp
:S save Unicode output text as UTF-8 file format
:T FontName:value choice of TrueType or OpenType font already installed in Windows
:U u/8/1/3 output text string type Unicode, UTF-8, UTF-16, UTF-32
:V r/l output text string converter for mostly RightToLeft or LeftToRight text
:W n/s/b weight of added characters Normal, SemiBold Bold
:X u/x output text string hexadecimal notation for UTF \u0000 or \x0000
:Y y/n italics Yes or No
:Z 1/2/4/8/A/B/C/D 1,2,4,8 bit per pixel grey and 4,8,16,24 bit per pixel color

For a detailed description of all batch options and commands see ***IconEditManual.pdf Appendix 10***.

A: Text optimized font from a text catalogue:

1: New font with or without anti alias

Minimum command look for text optimizd new font based on a text or text catalogue:

```
C:\iconedit\IconEdit.exe :24 :OC:\texts\Text.cpp
:FC:\fonts\OptimizedFont.c :Z2 :A
```

The text is assumed to be Unicode and the optimized font will be a Unicode font as C-source code.

2: Font based on existing Master Font

Minimum command look for text optimizing a font based on a text:

```
C:\iconedit\IconEdit.exe C:\fonts\MasterFont.c :OC:\texts\Text.cpp
:FC:\fonts\OptimizedFont.c
```

The master font and text is assumed to be Unicode and the optimized font will be a Unicode font as C-source code regardless of the master font type such as ief or bdf.

If the text is a classic 8-bit font encoding add the **:E** code-page type parameter so IconEdit can convert it to Unicode.

In this case the text catalogue is in Windows CodePage 1250 Latin Central European, but the master font still has to be Unicode:

```
C:\iconedit\IconEdit.exe C:\fonts\MasterFont.c :OC:\texts\Text.cpp
:FC:\fonts\OptimizedFont.c :Ew1250
```

IconEdit will convert the text strings in the catalogue as Windows Latin Central European to Unicode and make a Unicode font accordingly. The converted text can be written as a new Unicode text catalogue by adding the **:J** output text command:

```
C:\iconedit\IconEdit.exe C:\fonts\MasterFont.c :OC:\texts\Text.cpp
:FC:\fonts\OptimizedFont.c :Ew1250 :JC:\texts\UnicodeText.cpp
```

This will make the optimized font and the Unicode text catalogue a matched pair.

B: New copy of a font with the same characters but different look:

Minimum command look for font copy with new look with serifs and 2 bit per pixel anti alias:

```
C:\iconedit\IconEdit.exe :24 :OC:\fonts\OldFont.cp
:FC:\fonts\NewFont.c :A :TTimes_New_Roman:18 :Z2
```

The old font is assumed to be Unicode. The new font can now be used for the same texts as the old font.

C: Text optimized font for several plain texts or text catalogues:

Minimum command look for text optimizing a font based on a text catalogue:

```
C:\iconedit\IconEdit.exe C:\fonts\MasterFont.c :OC:\texts\Text_1.cpp
:FC:\fonts\OptimizedFont.c
```

The master font and text strings are assumed to be Unicode and the optimized font will be a Unicode font as C-source code regardless of the master font type such as ief or bdf.

The optimized font for the first text catalogue can subsequently be expanded with any number of texts or text catalogues one at the time from the original master font:

```
C:\iconedit\IconEdit.exe C:\fonts\OptimizedFont.c
:PC:\texts\Text_2.cpp :FC:\fonts\OptimizedFont.c
:AC:\fonts\MasterFont.c
```

The optimized font can now be used for both Text_1 and Text_2

D: Text optimized font for modified C-source code text strings:

Minimum command look for text optimizing a font based on a collection of text strings:

```
C:\iconedit\IconEdit.exe C:\fonts\MasterFont.c :OC:\texts\Text.cpp
:FC:\fonts\OptimizedFont.c
```

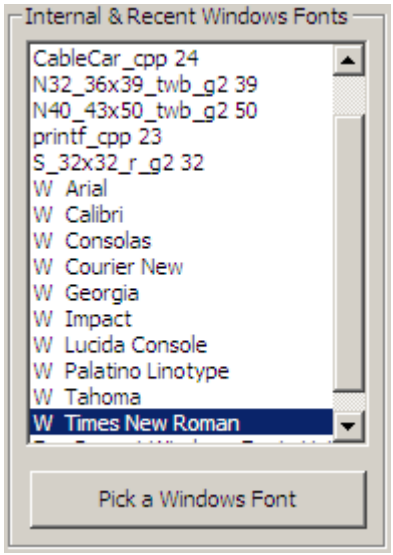
The master font and text strings are assumed to be Unicode and the optimized font will be a Unicode font as C-source code regardless of the master font type such as ief or bdf.

If characters from the text strings are missing in the master font you get a warning and should add the **:A** add missing characters command

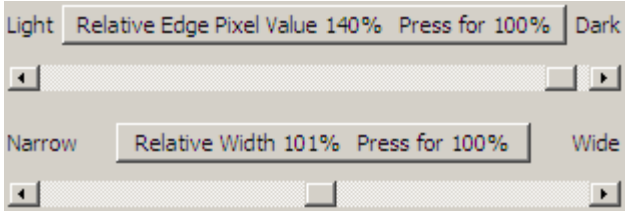
```
C:\iconedit\IconEdit.exe C:\fonts\MasterFont.c :OC:\texts\Text.cpp
:FC:\fonts\OptimizedFont.c :A
```

This adds the characters with default Windows font settings for the Arial TrueType font. For more control start IconEdit with your master font and open the compare dialog box:

Find the Windows font and the settings that give the best match:



Use the additional settings:

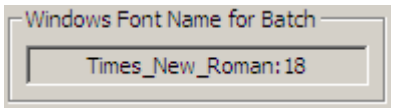


To get the closest match:

Press **OK**.

Open the normal Master Font dialog box:

And copy the Windows font name for the batch command:



Add the font settings: **:T** for Windows font name **:G** for edge pixel value **:C** for character width **:W** for bold and **:Y** for italics:

```
C:\iconedit\IconEdit.exe C:\fonts\MasterFont.c :OC:\texts\Text.cpp
:FC:\fonts\OptimizedFont.c :A :T Times_New_Roman:18 :G140 :C100 :Wb :Yn
```

If the font width is critical to your application also add the **:K** keep original font width command

```
C:\iconedit\IconEdit.exe C:\fonts\MasterFont.c :OC:\texts\Text.cpp
:FC:\fonts\OptimizedFont.c :A :T Times_New_Roman:18 :G140 :C100 :Wb :Yn
:K
```

A converted text can, or in some cases must, be written as a new Unicode text by adding the **:J** output text command, here are some typical cases:

1: European Languages

For European languages based on the Latina alphabet the UTF-8 string compression can give great memory footprint savings, for all other alphabets it often makes it worse.

Memory consumption for different string formats:

UTF-16 hexadecimal and pure **Unicode** characters always uses **2.0** byte per character ROM space.

UTF-8 hexadecimal characters take up different amounts of ROM space per character depending on language and alphabet.

1.0 byte per character: American English.
1.1 - 1.3 byte per character: Other languages written with the Latin alphabet.
2.0 - 2.2 byte per character: Other European and Middle Eastern languages except Arabic.
2.6 - 2.9 byte per character: Arabic and South Asiatic languages.
3.0 byte per character: Chinese, Japanese, and Korean.

If you want to convert the input text to a new UTF-8 Unicode text add the **:J** output text command and the **:U** utf command

```
C:\iconedit\IconEdit.exe C:\fonts\MasterFont.c :OC:\texts\Text.cpp
:FC:\fonts\OptimizedFont.c :JC:\texts\OutputText.cpp :U8
```

The output text is as a classic 8-bit file, but the text strings are still Unicode to the compiler. By changing the strings from normal 16-bit Unicode to 8-bit utf the string type is changed from wchar_t to char, or from char16_t to char8_t.

By adding the **:M** match output text string character type, string prefix, and hexadecimal notation command IconEdit will change the code to match the string content

```
C:\iconedit\IconEdit.exe C:\fonts\MasterFont.c :OC:\texts\Text.cpp
:FC:\fonts\OptimizedFont.c :JC:\texts\OutputText.cpp :U8 :My
```

Comments in the text string file are also converted to classic 8-bit so if your comments are not in English add the **:E** code-page type parameter so IconEdit can convert Unicode comments to classic 8-bit.

In this case Windows CodePage 1253 Latin & Greek is used for comments, see the ***IconEditManual.pdf Appendix 10*** for other classic CodePages

```
C:\iconedit\IconEdit.exe C:\fonts\MasterFont.c :OC:\texts\Text.cpp
:FC:\fonts\OptimizedFont.c :JC:\texts\OutputText.cpp :U8 :My :Ew1253
```

The output text strings will be full of hexadecimal numbers, and the comments will only be readable on an editor with the right Windows CodePage.

2: Middle Eastern Languages

Middle eastern alphabets are stored in strings from left to right, but should be shown on the display from right to left, and basic Arabic characters should be substituted with presentation characters, see **Chapter 10** for detailed explanation.

IconEdit can automatically detect and convert middle eastern texts in strings, so all you have to do is add the **:J** output text command and the **:U** string type command for normal Unicode:

```
C:\iconedit\IconEdit.exe C:\fonts\MasterFont.c :OC:\texts\Text.cpp
:FC:\fonts\OptimizedFont.c :JC:\texts\ModifiedText.cpp :Uu
```

The modified text will look all wrong in a normal editor, but correct on an embedded display.

3: South Asian Languages

The glyphs of South Asian alphabets are not fully defined in Unicode, only basic characters and diacritics and they have to be combined at display-time, see **Chapter 10** for detailed explanation.

This means that you can not make a general font for a South Asian language, but must build a font for your particular set of text strings with the **:A** add characters command and its minimum sub commands **:T** Windows font name **:W** normal or bold and **:Y** italics, and generate a modified text to fit the font with the **:J** output text command and its minimum sub command **:U** for string type Unicode:

```
C:\iconedit\IconEdit.exe C:\fonts\MasterFont.c :OC:\texts\Text.cpp
:FC:\fonts\OptimizedFont.c :A :T Times_New_Roman :Wb :Yn
:JC:\texts\ModifiedText.cpp :Uu
```

This will make the optimized font and the modified text a matched pair. The modified text will look all wrong in a normal editor, but correct on an embedded display with this special font.

E: Bulk conversion of Images to C-source code:

Minimum command look for convert all PNG images in a directory to C-source code:

```
C:\iconedit\IconEdit.exe C:\images\*.* :Bpng
```

The new C-source will have the same names as the NAME.png just another type NAME.c.

The converted image will normally match the color mode of the of the original image, but can be converted to other color modes with the **:Z** parameter.

This converts the image to 2 bit grey with dither:

```
C:\iconedit\IconEdit.exe C:\images\*.* :Bpng :Z2d
```

This converts only the specified image to 8 bit color with dither:

```
C:\iconedit\IconEdit.exe C:\images\Søpapagøje.png :Bpng :ZBd
```

See the IconEditManual.pdf for details.

Appendix A: Key terms and concepts as used in the IconEdit program

Symbol: A single picture that can be edited either separately or as part of a group. Each symbol has a number that will be its index in the saved file. The insertion and deletion of symbols will change the index of all the higher numbered symbols. The term symbol is also used for the symbol file containing the pixel information of a group or font. A symbol file can contain from 1 to 65536 separate symbols.

Group: A collection of symbols edited and saved as a whole. Typically a group contains symbols representing virtual buttons, arrows and logos.

Character: The combination of a picture or glyph that defines what the character looks like and a Code Point that is a reference to the symbol representing the picture or glyph. The Code Point is the number that represents the character in a text. The character can be edited either separately or as part of a font. The insertion and deletion of characters will NOT change the Code Point pointer to the symbol. The Code Point Character List system will link the numerical index assigned to each character symbol in a font with the Code Point pointer to the character. The resulting lookup table is saved with the font in a separate Code Point Character List file.

Symbol Number & Code Point: A collection of symbols be that characters, buttons, arrows, emoji or logos have a symbol number from 0 to nSymbols-1. Characters and symbols defined in the Unicode Standard have Code Points from 0x000000 to 0x10FFFF. Any character in a font designed in IconEdit has a Code Point that is stored in the Code Point Character List file. Normally IconEdit generates fonts where the Code Point for a character is identical to the Unicode Code Point, but there are exceptions such as Unicode characters moved to the Private Area in Unicode, or fonts that are converted to classical 8 bit standards.

Presentation Characters: In some scripts a character can have several different shapes depending on position in a word or sentence or the character can have indicators for stress or intonation. For scripts written from right to left there is also a small group of mirrored mathematical symbols. The Unicode plane 0 that defines about 50000 characters for living languages has support for only a few scripts that need presentation characters, the rest are supposed to be generated by the display system at time of rendering. This is not a problem for powerful systems such as PCs or mobile phones, but on small embedded systems the overhead is prohibitive. One possible solution is to auto-generate the necessary characters at the time the font is made.

Auto-generated Presentation Characters: Is a combination of one or more Unicode character code-point into a new symbol or glyph that defines what the presentation character looks like. The presentation character can either be made from a basic character from a script followed by one or more diacritics that change the meaning and look of the character, it can be a mirrored symbol, or it can be made from a surrogate pair that points to Code Points outside the 16 bit address room of Unicode plane 0 that defines the first 65536 Code Points in Unicode. In all cases a new Code Point is created in the private area in Unicode and the glyph can be addressed by a 16 bit address in the Unicode plane 0 - thereby making 32 bit addressing unnecessary. The modified Code Points are handled by the Code Point Character List file system, and the combination of Unicode Code Point that the new Code Point represent is saved within the Code Point Character List file.

Font: A collection of characters edited and saved as a whole. Traditionally a font contains 256 characters with an eight bit Code Point from 0x00 to 0xFF, where the characters from 0x20 to 0x7E are the American ASCII signs, numbers and letters, while so-called international letters are located at 0xC0 to 0xFF. There are many conflicting standards for the international letters such as DOS Code Points, Windows Code Points, ISO-8859 and KOI8. An attempt to make only one standard is the 16 bit

Unicode Plane 0 which contains 65536 characters from 0x0000 to 0xFFFF. IconEdit is designed to use both the old 8 bit standards and 16 bit Unicode. A font symbol file can contain from 1 to 65536 characters.

Proportional and Mono-Spaced Fonts: A proportional font is a font where the pixel width of the individual character symbols varies so it follows the visual width of the glyphs. A mono-spaced is a font where all character symbols has the same width so each character takes up the same amount of pixel space. Fonts can be either pure mono-spaced or proportional fonts, or some characters such as numbers or arrows in a proportional font can be converted to mono-space.

Classic CodePage Fonts with Name and Number: A classic 8 bit font containing 256 characters with Code Points 0x00 to 0xFF. Classic 8 bit fonts can only cover a limited number of languages. Just for the Europe you need between 6 and 11 different CodePages depending on the standard you use. To distinguish between the many CodePages they usually have a name and number such as ISO8859-4 or Windows 1252.

Code Point Character List: A file with a group of Unicode pointer ranges in a look-up table that describes which symbol index in a symbol file is used to represent each character. This makes it possible to save only the characters needed for a particular task in a symbol file without losing the character pointer. A Code Point Character List has the same name as the file with the symbols, just with the extension .cp. A classic 8 bit font containing 256 characters with Code Points 0x00 to 0xFF does not need a Code Point Character List as all necessary information is already in the .sym file.

Default Character: A special Code Point stored in the Code Point Character List file. This character is displayed when an attempt is made to write a character that is not present in the font. The default character can be any value from the font (typically SPACE ‘ ’ or QUESTIONMARK ‘?’), but if an application is sure to never need the default character the value can be left as a normally used character. If the value is outside the font when the Code Point Character List file is saved, the default character is set to the first character in the font.

Palette: A file with 4, 16 or 256 colors or grey-tones. The color or grey-tone of a pixel in a symbol can either be stored in the symbol file directly, or the symbol file may just contain an index into the palette for each pixel. If the number of different colors in a symbol is low, this latter method can result in great savings in memory consumption.

System Palette: A Palette file used for more than one symbol file. There are several reasons for using a system palette common to all the symbols in a project. Some display controllers can only work with palette based symbols, and changing the palette dynamically is either not possible or will result in color flicker with each change. The use of a common palette for all symbols means less memory consumption, as there is only one palette in total instead of one for each symbol. Using one common system palette makes it easier to make and maintain symbols such as virtual buttons with a common look and color scheme. If the color scheme for the whole design has to be changed, you need only open one of the symbol files, change the colors to the new style, and save the symbol and system palette. Because the system palette is common to all the symbols in the project, they now all have the new colors.

DataSet: All the information belonging to a font or group, such as file names, pixel and tool colors, palette size and colors, number of symbols, color mode and number of possible colors, Code Points for symbols, and display and undo information. Undo and Redo operate on the whole data set, and not just the pictures. Save and read files normally only operate the information necessary for recreating the font or group.

Data Set File Name: The data set file name and the name of the symbol or font in the sym file is the same. The *.sym file is a standard "C" file, and the name must be in accordance with the 'C' standard for names. Only ASCII letters, numbers and underscore_ are permitted, and names must not start with a number.

File Mode and Project Mode: IconEdit can operate and save data in two modes that are not mutually exclusive. In file mode IconEdit saves each DataSet in a separate sym format file and files can be reloaded automatically next time the program is started. In project mode IconEdit saves all DataSets in one project file and the current project can be reloaded automatically next time the program is started. Please note that the sym files and the project files are independent storage units for data and can be used for storing alternative versions of the same DataSet.

SYM Format: Standard internal and external data format for IconEdit. All data formats read from a file or pasted from the clipboard are converted to sym format, and if nothing else is specified data is written to disk as two or more of the four files in the sym format: *Name.c* is a header for the other files, *Name.sym* is the pixel information for characters or symbols, *Name.cp* is the Code Points for fonts and *Name.pal* is pixel and/or tool colors.

IEF and IEP Format: IEF is a compressed input/output disk file format for temporary storage of symbol data. It contains the same information as the normal c-sym-cp-pal file quad of the sym data format, but in a much smaller proprietary format. IEP is a container for all the DataSets in a project, it has general header for the project followed by a number of DataSets in IEF format, all in one format for easy exchange of data with other members of a project group or for working on several projects simultaneously.

Save, Open, Import and Export: All these operations have to do with reading and writing disk files. Import and export imply a data transformation with loss and/or modification of information. One example of this is reading a text or 'C' file, finding all the different characters in the text, ordering the characters alphabetically and drawing the characters in accordance with the Master Font. The text is "lost" but a font with all the characters in the text is gained. Open and save operations may be in different file formats, but with the same information as in the standard sym format. One example is saving a font as a bdf file. The white space around the characters is removed, and the Code Point Character List information is stored with each character in the bdf file instead of storing it as a separate Code Point Character List file, but no information about the font is lost, and it can still be read back in IconEdit as a sym DataSet.

Master Font: A combination of TrueType and OpenType vector fonts installed in Windows and any font opened in IconEdit. The master font is for drawing new fonts or text on symbols based on existing fonts in Windows or fonts stored in *.sym or *.bdf disk files.

TrueType & OpenType, ClearType, XP Standard and Smoothing: TrueType and OpenType vector fonts are the most used font file standards in all Windows versions from XP to 10. **TrueType** & **OpenType** are vector formats, and before the characters can be displayed on the screen the vectors have to be rendered to pixels to make the characters visible. Windows have 3 different font and character rendering modes, **ClearType** does smoothing of all characters regardless of size or shape, **XP Standard** does smoothing of small and large characters but renders intermediate size characters as black & white, finally all smoothing can be turned **Off** to render all size characters as black & white.

Glyph: In printing technology, a glyph (from a Greek word meaning carving) is a graphic symbol that provides the appearance or form for a character. A glyph can be an alphabetic or numeric figure or some other symbol that pictures an encoded character.

ASCII Standard: Standard Latin 7-bit characters designed for use in the USA. Only characters 0x20 … 0x7E are assigned symbols to make them visible. The rest are modem control codes, most of witch are now obsolete.

Unicode Standard: Standard for assigning symbols to 8-bit, 16-bit or 32-bit characters world wide. The standard pages and planes, there are 256 characters in each page, and 256 pages in each plane, and there are a total of 17 planes. The very first page (№ 0) on the first plane contains Northern American and Western European characters, and can be addressed by 8-bit Code Points. The first plane (№ 0) contains almost all living languages, and can be addressed by 16-bit Code Points. The higher planes (№ 1 to 17) are normally addressed by 32-bit Code Points though only 21 bit is necessary. The second plane (№ 1) primarily contains historic scripts, and the third plane (№ 2) primarily contains rare Chinese, Japanese and Korean Ideograms. All of these are supported in various degrees by Windows, the higher the Windows version number, the better the support. For information about the present stage of Unicode implementation see: www.unicode.org.

Unicode Diacritic or Accent: In printing technology, a diacritic or accent is a graphic symbol that is added to a basic letter to change the sound-values of the letters to which they are added. The Unicode Standard defines about 700 diacritics or accents.

Unicode Pre-composed Character: In printing technology, a pre-composed character is a glyph consisting of both a basic letter and a diacritic or accent so the combination can be treated as one unit. The Unicode Standard defines about 500 pre-composed Latin characters.

Unicode Presentation Character: In printing technology, a presentation character is an alternate glyph for a basic letter. Presentation characters are used in connection with Arabic scripts where the shape of a letter depends on its position in a word. The Unicode Standard defines about 700 Arabic presentation characters.

Unicode Combined Character or Ligature: In printing technology, a combined character or a ligature is a glyph consisting of both one or more basic letter and one or more diacritic or accent so the combination can be treated as one unit. The Unicode Standard defines a few ligatures for Latin and Arabic. Southern Asiatic scripts have a very large number of ligatures, Unicode does not define them directly as glyphs, but IconEdit can use Windows rendering engine for TrueType and OpenType fonts to generate over 4000 of them to fit a given text.

Unicode Scripts & Symbols: A set of built-in predefined CodePoints for creating fonts with single or multiple Unicode ranges directly from the names of the ranges. There are 2 groups of ranges. The first group is plane 0 ranges, which can be addressed by 16-bit Code Points; these are written as “Name”. The ranges are ordered alphabetically and have names in accordance with the Unicode standard for the Basic Multilingual Plane as described at www.Unicode.org. The second group is plane 1 ranges, which can be addressed by either one 32-bit or two consecutive16-bit Code Points; these are written as “NAME”.

Language & Region Filters: A set of built-in predefined Code Points for creating single or multiple language Unicode fonts directly from the names of the languages. The language and region filters fall in 3 groups. The first group usually contains more than just the characters to write a language. Numbers, common signs, characters from other languages or alphabets and currency symbols are also included. The second group is coverage of regions as defined by the ISO-8859 standard; these are marked by “ISO”. The third group is minimum requirement for a particular language; these are marked by “Basic”. Languages that are commonly written with more than one alphabet have filters for each alphabet. Extra user filters can be created by opening font files, Code Point files or text files.

Alpha Blending, Anti-Aliasing and Smoothing: The mixing of foreground and background colors of a pixel to make contours appear smoother. The pixel is given a color somewhere between the foreground and the background color to make it appear that the foreground figure does not fully cover the pixel, and the background therefore is partly visible. The alpha value is the weight of each foreground pixel's RGB value when it blended with the background.

On Off Transparency: Single color on off transparency is a method for drawing symbols of any shape without having to use the 32bit per pixel semitransparent data format. One of the possible colors in the data format is marked as fully transparent, and can be used to mark the unused pixels around the main part of the symbol to make them invisible. It is referred to in this manual as the **transparent color**. For a pixel to be considered transparent, the pixel color and the transparent color needs an exact color match, any other pixel color however much it resembles the transparent color will still make the pixel fully opaque. Single color on off transparency works with all non transparent data formats from 24 bit RGB color to 2 bit palette can make it possible to reduce memory consumption significantly.

Basic Color: A basic color is the color you normally see when you watch an object. But imagine a stained glass window or a pair of sunglasses; they have a color, but also a degree of transparency. In computer graphics a semi transparent pixel has 4 values: Alpha, Red, Green and Blue. The A value is the opacity (max value = fully opaque, zero = fully transparent), and the R,G,B values are the color you see, and it is referred to in this manual as the **basic color**.

Grey Tone: A grey tone is not a color in the normal meaning of the word. It is a degree of lightness as seen on a black and white photograph or TV. This degree of lightness is referred to in this manual as a **grey tone**. The **grey tone** and the **grey tone palette** data formats are for drawing black and white symbols and characters. The grey tone data format has fixed grey values and is primary for use on screens that has build-in gamma correction, and therefore has a linear relation between input values and lightness of the display. The grey tone palette data format has an additional palette where the grey values can be adjusted to compensate for un-linear relation between input values and lightness on screens that does not have build-in gamma correction.

Intensity Level: An intensity level does not have a color at all. It is a degree of opacity applied to a basic tool color (the rendering color) when it should be drawn on a background. This degree of opacity is referred to in this manual as an **intensity level**. Typical uses for intensity level are anti aliased characters or figures where smoothness of the edges is simulated by mixing the tool color with the background color to make it appear that the figure only covers part of the pixel. Other uses are watermarks and semi transparent copyright notes on color photographs. The **intensity level** data format can be edited in 2 different ways, as a grey symbol or character where black is full opacity and white is full transparency, or as a semi transparent format with a chosen rendering color. The saved data format is independent the editing method, it is the same in both cases.

Dither: Dithering is used to create the illusion of color depth in images on displays with a limited color resolution or a small palette. In a dithered image, colors that are not available on the display or in the palette are approximated by a diffusion of colored pixels from within the available colors.

Technical Support

You do not have to wait until you find an error to contact us. If you think something is missing, or just impractical, please feel free to send an email to:

DanMagic

Bregnerødvej 61
DK 3460 Birkerød
Denmark
Email John@DanMagic.dk