

RasterFontGen

# Description

RasterFontGen is a tool for creating “rasterized” TrueType fonts. Use RasterFontGen to create .rasterfont files that you can subsequently load into your game and render using the RasterFont API. The .rasterfont format was specifically developed with the Scorpio Devkit Font Panel in mind. In particular, .rasterfont supports a lightweight text rendering system that is appropriate for rendering text on the CPU.

# Building RasterFontGen

RasterFontGen is provided as source code. To build the tool, simply open RasterFontGen.sln and build from Visual Studio.

# Using RasterFontGen

To get the complete set of command line options use:

>RasterFontGen.exe /?

RasterFontGen supports the following features:

* Works with any TrueType font, including symbol fonts. Note that the fonts must be installed on the PC where you run the tool.
* Can specify different font heights (in pixels.)
* Multiple code ranges are supported so that you can flexibly specify ranges of Unicode characters

The height option is used to specify the height in “logical units” (think pixels) and has nothing to do with “printer’s points” i.e. if you specify –h 16 you get a font of height 16 pixels. This means that the distance from the tallest ascender to the deepest descender, considered over all supported glyphs in the font, will be 16 pixels.

If you don’t provide any value for the typeface, the tool will use the system font selection algorithm to find a font that works for the other provided arguments. If you misspell the typeface name, the tool will use the system font selection algorithm to find a font that works for the other provided parameters. If you don’t get the typeface you want, then chances are you didn’t provide the exact string needed to specify that typeface. Here are some recommendations to help you specify the typeface name:

* Go to the Fonts folder in Control Panel to determine what fonts you have installed. Use this as the guide to choose the correct name for the font (i.e. the value to provide to the -tf argument.)
* Note that the Fonts folder has separate icons representing single fonts as well as font families. You should drill down until you open the “proof page” for the particular font face you are looking for. The proof page will show the text “The quick brown fox jumped over the lazy dog.” On this page, you will find the typeface name you should use. (e.g. use “Algerian” rather than “Algerian Regular”.)

If you don’t specify a code range for the font, then RasterFontGen will generate all glyphs supported by the specified font. Typically you will want to use one or more -cr: specifications to limit the number of glyphs that are generated in the resulting .rasterfont file. Note that the -cs flag is used as a hint to the system font selection algorithm and should not be used to limit the code range(s) for the set of generated glyphs.

The tool will provide a warning when attempting to generate a glyph for an unsupported code point. Heed these warnings if you are trying to find a font that supports a specific code point. Otherwise, these warnings can be safely ignored.

# Examples:

The following example creates a .rasterfont file from Lucida Console at a height of 16 logical units (pixels.) It will also include all glyphs supported by the font (quite a few!):

>RasterFontGen -tf “Lucida Console” -h 16 –of LC.rasterfont

The following example creates a .rasterfont file using specific code ranges (specified using -cr) from a symbol font:

>RasterFontGen –tf “Segoe Xbox MDL 2 Assets” –h 16 –of Symbols16.rasterfont -cr:0xE3AF-0xE3B2 -cr:0xE48B-0xE48C

The following example creates a .rasterfont file using the Algerian Regular font face which covers the ANSI range of code points (and probably some more glyphs):

>RasterFontGen -tf "Algerian" -cr:0x00-0xFF -h 16 -of AR.rasterfont

# Implementation notes

The command line for the tool provides more options than the .rasterfont format supports. This is because the command lines exposes options corresponding to all the fields of the LOGFONT structure. This design was chosen for maximum flexibility at the expense of causing a little bit of confusion. Chances are that you won’t have any use for the more arcane options anyway.

The .rasterfont format currently only supports 1-bit-per-pixel format. The command line options support both a quality option (-q) as well as a pixel depth option (-d) These options only affect how each glyph is rendered (via GetGlyphOutline) prior to being “flattened” into the .rasterfont format. So you are likely to see no affect/improvement using these flags.

Disclaimer: Don’t expect to create a super high-quality text rendering system based on the .rasterfont format. The .rasterfont format is only intended for lightweight text rendering on the CPU and is specifically targeted for use on the Scorpio Devkit front panel display. Within the intended scope, I think you’ll be highly satisfied with the results.