The Ultimate Oldschool PC Font Pack v2.2: Documentation

ABOUT

The Ultimate Oldschool PC Font Pack started out with the idea of paying tribute to ancient PCs and their bitmapped, pre-GUI typography (if you can call it that). It was inspired by similar efforts that cover other vintage machines: classic system fonts from the Amiga, C64, Apple II, Mac, ZX Spectrum, Atari 8-bit/ST etc. are all celebrated. On the other hand, the IBM PC and its clones seem to get little love... except for that one VGA text mode font (which has been remade numerous times, to varying degrees of success).

This collection is here to remedy that, and to bring you pixel-perfect remakes of various type styles from text-mode era PCs - in modern, multi-platform, Unicode-compatible TrueType form (plus straight bitmap versions).

Although the goal is to make it a complete resource, the main focus is on **hardware/firmware** character sets: the kind that's located in a ROM chip on the system board or graphics card, which is what you'd see by default when working in text (or graphics) mode. Software-loadable fonts are also within the scope of this collection (if associated with a particular machine or display system), so some of these have also made it in.

FREQUENTLY ASKED QUESTIONS

Q. What's with all those weird prefixed versions of each font? Which one(s) do I want/need?

A. The font name starts with two letters which signify the rendering:

- When crisp appearance is important, you'll want bitmap rendering. This is available with the **'Mx'** ('mixed-format') and **'Bm'** (bitmap) variants.
- When perceptual fidelity is important, you'll want the true aspect ratio of the original fonts. This is available with the **'Ac'** variants ('aspect-correct', or maybe 'accurate', take your pick).
- Wherever the above formats don't work (well), you can use the **'Px'** ('pixel-font') variants, which are TrueType fonts with a square-pixel aspect, and should be the most compatible.

The rest of the prefix specifies the character set:

- **'437'** includes all 256 characters available in the IBM PC's original encoding (codepage 437).
- **'Plus'** has a greater selection (some 780 characters in total) to support more scripts and symbols; a lot of them are custom additions, so this was only done for a few selected fonts.

Q. Okay, and the suffixes - "-2x", "-2y"? What are these all about?

A. These indicate how the font is scaled. In many cases, the original video hardware would double or halve the width of the text by modifying the pixel clock that drives the display. For example, take 80-column vs. 40-column text modes: the number of characters per row varies, but each row takes up the same width on the screen. In other words, what changes is the horizontal pixel aspect ratio.

To reproduce this effect, some of these fonts come in additional double-width and/or half-width versions. Generally, the "parent" font (without a suffix) is the one with the pixel aspect closest to square, "-2x" is for double-width (2:1) versions, and "-2y" signifies half-width (1:2). For the lowdown on how those different versions were used originally, see the detailed info page for each font.

Q. If a font already has an expanded 'Plus' charset, isn't the '437' version redundant? Why include it at all?

A. 'Plus' does include every character from '437' and then some, but there are two reasons why the '437' versions are still useful:

- The 'Plus' versions remap a few CP437 characters (Greek/Math symbols) to other code points to resolve ambiguities. The '437' versions always preserve the original encoding 100%.

- With 8-bit encodings, some programs/text engines need these versions to ensure proper rendering. This is especially true in Windows, which fails to provide a "**DOS/OEM**" script option for fonts that include more than that. So if you're working with 8-bit ASCII text which requires this DOS charset, '437' is your best bet.

Q. The TrueType fonts don't look good on my end (badly aliased, too bold, distorted, etc.) - what's the deal?

A. These fonts duplicate the original bitmap characters, with their right-angled pixel outlines, which don't scale smoothly. That means they'll only look good at the original pixel height (or at an integer multiple of it); this size is noted for each font in the font list. E.g. for an 8x16-pixel font, the TrueType version should be displayed at a height of 16 (/32/48/64/...) pixels.

Most current software determines text sizes in points (pt), not pixels. The conversion depends on resolution, so the size you'll want to select may vary with your OS, monitor, and DPI settings: more info below.

Q. Will you add [insert font name]?

A. If it's within the scope of this collection, then sure! In other words, if:

- It's a bitmapped text-mode font (no GUI stuff),
- It originates with some kind of IBM PC-compatible video hardware, and
- It's different from the fonts already included.

It's not hard science, so there are naturally some borderline cases, but without these guidelines the scope would become certifiably insane (it's pretty deep into the rabbit-hole as it is). If you can contribute anything that does meet these criteria, it would be more than welcome - please let me know.

Q. Will you add bold / italics / programming ligatures / etc.?

A. No, there are no plans to do that. Such modern GUI concepts don't square very well with fonts that were originally designed with pure text mode in mind, so they wouldn't really work visually. Most current text renderers can already simulate bold/italic styles by fattening or slanting the glyphs, but these are mostly basic "smear 'n' shear" algorithms that work about as well as you'd imagine.

Exception: certain fonts (very few of them) come from systems that natively supported different text 'weights' simultaneously – usually as an alternative to brightness. This is implemented as Regular and Bold styles of the same font family, and these cases are noted in the font index.

Q. How about raw binary versions of these fonts? For use in real VGA text mode, homebrew hardware projects, etc.?

A. Check out my DOS-based VGA font editor Fontraption[→] - the download includes most of these oldschool PC fonts in raw binary format. The few exceptions are those that are too large for VGA text mode (i.e. more than 8/9 dots wide).

Q. Is every single font here 100% faithful to the original raster typeface?

A. No, not *every* single one. For a start, the 'Plus' versions with the expanded character sets often include many custom glyphs that weren't there originally. Even with the '437' variants, there are cases where some changes were called for: a few of them are *remapped*, since they're based on e.g. Japanese fonts, and the originals only had the lower (ASCII) half of the codepage present. These remaps are clearly noted in the font index (and by "re." in the font name). Others had obvious errors, like a curiously wandering baseline, or misaligned box-drawing characters – these cases are listed under "Diffs & Errata".

I'm all for accuracy, but this isn't purely a preservation project; these fonts are meant to be used, not to just function as museum reproductions. When one of the original charsets has an issue that hampers usability, I tend to fix it and note it as such.

Q. Is this even legal? The original fonts are not yours. Do you have the right to distribute these versions?

A. Short answer: as far as I was able to research, there is nothing illegal or infringing about this collection. The raw bitmap typefaces are not copyrightable, unlike fonts in specific formats such as .fon and TrueType (which qualify as software); at least this is the case in the US, where this site is based. For a longer and far more boring answer, see the legal stuff section below.

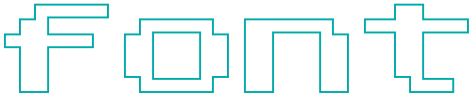
FONT FORMATS / VARIANTS

Fonts in this pack come in a few formats, identified by the prefix on the font's name ("Ac437", "PxPlus", etc.). Here's the gist on what they're all for:

'Px' (pixel outline) TrueType fonts

This is the same TrueType variant used in v1.x of this pack. The outlines in these fonts follow the pixel grid of the original raster characters: each pixel in the original is represented as a square region in the outlined glyph, so the character forms can be replicated nicely on a square-pixel display (i.e. practically every monitor and display device made today).

Raw bitmapped text (original)



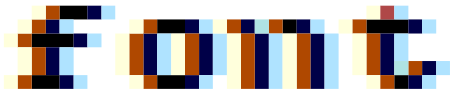
TrueType outline following the pixel glyph shapes

The 'Px' prefix is followed by the character set, either '437' or 'Plus'. **'Px437'** fonts cover the selection of 256 characters established by the original IBM PC, also known as Code Page 437. **'PxPlus'** fonts cover a wider selection of characters to support more scripts and symbols; both of these varieties are 100% compatible with Unicode environments. See Charsets & Encodings below for details.

'Mx' (mixed format) TrueType fonts: outlines + bitmaps

These are basically the same as the 'Px' fonts, but besides the outline forms, they also contain **bitmap** versions of the characters. At the **correct size** (i.e. the same as the original oldschool charset), this provides a 1:1 pixel-based reproduction of the original, without any grayscale or subpixel "smoothing" artifacts that tend to show up in ClearType/FreeType/etc.

Although not widely used, this is actually a standard feature of sfnt containers (the basis of the .ttf format): an outline font can have embedded bitmap 'strikes' tuned for particular sizes, and the renderer is supposed to use them whenever available. The whole mess of rasterizing an outline is bypassed, and you get a perfectly sharp rendering.



In SOME cases, 'Px' fonts may show fringing if the text is anti-aliased

'Mx' fonts will still render as sharp bitmaps (at the right size)

This sounds like the ideal setup for reproducing oldschool bitmap text, so you may ask yourself if it doesn't make the 'Px' variants redundant. If the same format can handle both outlines and bitmaps - and bitmaps are vastly preferable for *our* purpose - then what's the use for the outline-only 'Px' format? Why not just use the 'Mx' fonts everywhere and spare us the disk space and the confusion?

The prosaic answer is that specifications are one thing, and real-world support is another: the bitmap part of the deal just isn't correctly implemented everywhere. Namely, making these fonts compatible with Windows required an ugly hack[→], because Windows ignores the spec *unless* the fonts meet certain undocumented(!) criteria. The upshot is that Windows behaves weirdly with the 'Mx' fonts: they may show up with "@" prefixes in font lists, and more annoyingly, you cannot select the "DOS/OEM" script. This means that they won't display correctly for text that uses a real 8-bit OEM encoding - e.g. CP437 text files from DOS that use non-ASCII characters (.NFO artwork and such). For this, the 'Bm' fonts (8-bit .FON) are more compatible. I haven't tested extensively on Linux, but over there things seem to depend on the rendering engine, the desktop environment, and the individual programs themselves.

In practice, the 'Mx' fonts seem to work just fine in Windows and Linux when the encoding is Unicode, and the full range of CP437 (DOS/OEM) characters is available with the **'Mx437'** fonts. As with the other formats, some fonts also have **'MxPlus'** variants that support a wider range of scripts and symbols.

'Ac' (aspect-correct) TrueType fonts

'Px' and 'Mx' font variants are tuned for **square pixels,** which is what current displays are based on, and is the default assumption of practically all common GUI environments. This makes it easy to

ensure 1:1 reproductions of the oldschool glyph *shapes*, but in most cases, the aspect ratio isn't the same as the original.

That's because the original fonts were shown on CRT or LCD monitors that *didn't* necessarily have square pixels. CRTs were generally 4:3, but the pixel aspect ratio depended on the resolution, which varied with hardware and with the chosen display mode. LCD or plasma panels (for portable machines) came in a dizzying array of shapes and sizes.

So alongside the simple square-pixel versions, these fonts have 'Ac' (aspect-corrected) variants to reproduce the original appearance; both the "square" and "corrected" pixel ratios are listed for each font in the index. Some fonts *did* use square pixel ratios originally, and other cases were close enough that the difference is negligible. For these fonts the square-pixel versions should be good enough, so no 'Ac' variants are provided.

The IBM MDA font shown in square-pixel aspect

...vs. the pixel aspect of an actual IBM MDA display

When the aspect-corrected versions are shown on a current display, the outline of the characters is still rasterized onto a square-pixel grid, so the trade-off is sharpness/quality. On high-DPI monitors, or at the right text sizes (with good subpixel anti-aliasing), the result can still look pretty good – you can try it out on this site if you select a font from the index and play around with the preview options. See Aspect Ratio below for more details about these fonts.

Just like with 'Px' and 'Mx', these fonts come as either **'Ac437'** (covering all CP437 DOS/OEM characters) or **'AcPlus'** (the extended multi-lingual charsets).

'Bm' (bitmap) fonts, .fon and .otf formats

'Bm' fonts contain plain bitmap characters, so they aren't subject to rasterization issues with grayscale or subpixel anti-aliasing; they should always render as 1:1 pixels, as sharp and crisp as they can be. Unfortunately there's no widely used bitmap format which is universally supported everywhere; for now these fonts are available in the following formats:

- **FON (for Windows):** this format only supports 8-bit encodings, so it cannot be used for the wider 'Plus' character sets. Only the '**Bm437**' variants, which follow the US-Latin extended ASCII encoding from DOS, are available in this format.

  The character set of the .FON format (in this case 'DOS/OEM') always seems to be interpreted correctly, with no ifs or buts, whether you're working with Unicode or CP437 text. Some programs may turn up their noses at 'DOS/OEM' TrueType fonts, so these versions can effectively force them to play nice with CP437 data.

- **OTB (for Linux):** OpenType Bitmap fonts, as currently supported by X11, are more or less (but not exactly) the '**Mx**' sfnt fonts minus the TrueType outlines, i.e. only the bitmap characters. Unlike .fon, they aren't limited to 8-bit charsets, so both '**Bm437**' and '**BmPlus**' variants are provided.

  Linux has used other bitmap formats (PCF/BDF) for decades, and those are equally capable, but support for them was recently broken in many places, due to a minor(!) version upgrade of the Pango library unceremoniously dropping FreeType. OTB should have wider support now, but I haven't been able to test these fonts very thoroughly; as of v2.1, they're more of an experimental last-minute addition.

'Web' (webfonts, .woff format)

Basically these are versions of the 'Px' TrueType fonts, optimized and compressed for web usage. All major current browsers already support plain .ttf fonts natively, so having yet another version may seem like a waste of space, but there are a few things to consider:

- The compression does make the .woff fonts much smaller, i.e. better for remote delivery.

- Sizing is much more straightforward and intuitive in the .woff versions. Thankfully, web usage remains free from the typical OS's insistence on irrelevant units like 'pt' (points), on limiting the sizes you can select, and other such silliness.

  CSS lets you specify any desired font size and line height independently, in either **em** or **px** units; so the .woff versions make the em size always equal to the line height, which can always match the font's original pixel height. If the original font had an 8x14 character size, the web version should simply be used with a font-size *and* a line-height of 14px (which then becomes equal to 1em) – simplicity itself.

So, why make web versions for the 'Px' variants but not for the others? The reasoning is:

- If you want aspect correction (as in the 'Ac' variants), you don't really need a separate font with the correction baked into it — you can always do it yourself with CSS transforms. That's what this site does in the font preview pages.
- If you want bitmap glyphs (as in the 'Mx' or 'Bm' variants)... alas, no can do. The format technically *does* support the 'Mx'-style method of bitmap embedding, but current browsers seem to run all incoming webfonts through something called the OpenType Sanitizer[→], which validates the fonts and gets rid of Bad Things. As of this writing, embedded bitmaps are thrown out with the trash, although there is an open issue[→] for a possible fix. If this gets resolved (either on the browser end or on the Sanitizer end), then bitmapped versions may yet happen.

Summary / tl;dr version

To put the above wall of text a little more succinctly, it works sort of like this:

Variant	Bitmap?	Outline?	Charset	Pixel aspect	Format	Notes
Px437		■	DOS/CP437	Square	ttf	TrueType w/1:1 pixel outlines
PxPlus		■	Extended	Square	ttf	TrueType w/1:1 pixel outlines
Mx437	■	■	DOS/CP437	Square	ttf	TrueType w/embedded bitmaps
MxPlus	■	■	Extended	Square	ttf	TrueType w/embedded bitmaps
Ac437		■	DOS/CP437	Original	ttf	TrueType, aspect-corrected
AcPlus		■	Extended	Original	ttf	TrueType, aspect-corrected
Bm437	■		DOS/CP437	Square	fon/otb	Bitmap-only formats, Win/Linux
BmPlus	■		Extended	Square	otb	Bitmap-only format, Linux
Web437		■	DOS/CP437	Square	woff	Aspect is flexible using CSS
WebPlus		■	Extended	Square	woff	Aspect is flexible using CSS

Confusing? Probably. I *wish* there was a way to roll all of these into a single convenient font format that just worked everywhere, but we're not that lucky, so I've tried to cover as many bases as possible.

ENCODINGS AND CHARACTER SETS

Fonts in this collection come in two character sets, as noted in the font name prefix.

- **'437' (available for all fonts):** This features the classic set of 256 characters established by the original PC, also known as Code Page 437[→] (or "PC ASCII", "Latin-US DOS/OEM", and other catchy names). The character mapping to Unicode aims for maximum compatibility with the original codepage, so in addition to normal usage, these fonts should also let you view CP437 text in its original encoding.
- **'Plus' (for specific fonts only):** On top of the CP437 range, these support extended Latin, Greek, Cyrillic and Hebrew scripts plus a bunch of additional glyphs and Unicode symbols. The extra characters were taken from international versions of the original hardware (if available), or designed to closely follow the existing ones.

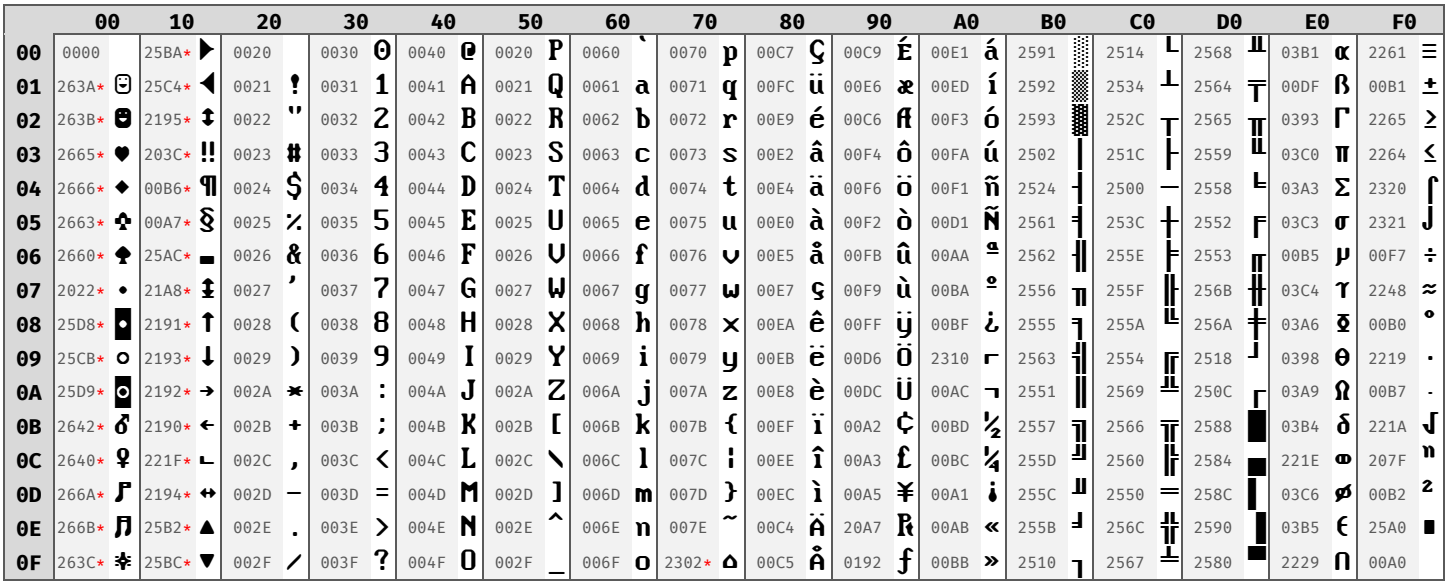
  There are 782 characters in total (more than the Windows Glyph List v4[→] -- in fact the entire WGL4 range is there). A handful of the CP437 glyphs had to be remapped (read below for why), but they're all still around.

The '437' Character Set

In most cases, code page 437 was the original character set of the source fonts. This pack's converted versions are still standard Unicode fonts - they just don't include a whole lot of the Unicode range.

CP437 can't really be mapped to Unicode in a simple 1:1 manner. The culprits are characters **00h-1Fh** and **7Fh,** which can be interpreted either as control codes or as graphical symbols. Thus there are two widely used mappings: the standard IBM/MS map[→] (which does the former), and Unicode's "IBMGRAPH" map[→] (which does the latter). Trouble is, software that expects one of them may not always play nice with the other one.

As a solution, these fonts cover both bases in one mapping: the ambiguous characters are duplicated so that your program will find them at either placement. Windows detects the fonts as "OEM/DOS", and you can use them in any program or environment that understands this charset (including the Command Prompt). The same will be true on other platforms, as long as your software is properly configured -- RTFM, YMMV, etc.



	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
00	0000	25BA*	0020	0030	0040	0020	0060	0070	00C7	00C9	00E1	2591	2514	2568	03B1	2261
01	263A*	25C4*	0021	0031	0041	0021	0061	0071	00FC	00E6	00ED	2592	2534	2564	00DF	00B1
02	263B*	2195*	0022	0032	0042	0022	0062	0072	00E9	00C6	00F3	2593	252C	2565	0393	2265
03	2665*	203C*	0023	0033	0043	0023	0063	0073	00E2	00F4	00FA	2502	251C	2559	03C0	2264
04	2666*	00B6*	0024	0034	0044	0024	0064	0074	00E4	00F6	00F1	2524	2500	2558	03A3	2320
05	2663*	00A7*	0025	0035	0045	0025	0065	0075	00E0	00F2	00D1	2561	253C	2552	03C3	2321
06	2660*	25AC*	0026	0036	0046	0026	0066	0076	00E5	00FB	00AA	2562	255E	2553	00B5	00F7
07	2022*	21A8*	0027	0037	0047	0027	0067	0077	00E7	00F9	00BA	2556	255F	256B	03C4	2248
08	25D8*	2191*	0028	0038	0048	0028	0068	0078	00EA	00FF	00BF	2555	255A	256A	03A6	00B0
09	25CB*	2193*	0029	0039	0049	0029	0069	0079	00EB	00D6	2310	2563	2554	2518	0398	2219
0A	25D9*	2192*	002A	003A	004A	002A	006A	007A	00E8	00DC	00AC	2551	2569	250C	03A9	00B7
0B	2642*	2190*	002B	003B	004B	002B	006B	007B	00EF	00A2	00BD	2557	2566	2588	03B4	221A
0C	2640*	221F*	002C	003C	004C	002C	006C	007C	00EE	00A3	00BC	255D	2560	2584	221E	207F
0D	266A*	2194*	002D	003D	004D	002D	006D	007D	00EC	00A5	00A1	255C	2550	258C	03C6	00B2
0E	266B*	25B2*	002E	003E	004E	002E	006E	007E	00C4	20A7	00AB	255B	256C	2590	03B5	25A0
0F	263C*	25BC*	002F	003F	004F	002F	006F	2302*	00C5	0192	00BB	2510	2567	2580	2229	00A0

Codepage 437 (value = column + row) mapped to Unicode; all values are hexadecimal.
* = these characters are also duplicated at the Control Code points (Unicode value = CP437 value).

The 'Plus' Character Set

A few selected fonts also come in multi-lingual versions with an expanded character set. These 'Plus' versions don't strictly duplicate the original hardware/software, which almost always used sets of 256 characters. Rather, they combine characters from various sources.

In some cases, I've had access to multiple code pages of the original fonts - e.g. from non-US versions of CGA/MDA boards, from multi-lingual ROMs (like the Amstrad or Tandy 1000 sets), or from IBM's Adapter Interface font data; these were all pulled into the 'Plus' versions. Mostly however, they had to be fleshed out with custom additions, and for those extra glyphs I've tried to match the original font's style as closely as I could.

The full 'Plus' charset (along with the supported Unicode ranges):

A few things that may (arguably...) be useful to know:

- **Alternate number forms:** there's a flat-top '3' ʒ mapped to U+01B7 'Latin capital letter Ezh', which is more easily distinguished from the Cyrillic letter Ze **З**. Also, two alternative zeroes (dotted and slashed) are mapped to U+2299 **⊙** 'circled dot operator' and U+2300 **⌀** 'diameter symbol'.

- **Cursor shapes:** Unicode character U+2581 **▁** 'lower one eighth block' can be used to mimic the classic text-mode cursor appearance. U+2584 **▄** 'lower half block' and U+2588 **█** 'full block' could also stand in for those respective cursor forms.

- **Remapped codepage 437 glyphs:** The 'PxPlus' fonts still include all the original glyphs from the CP437 (DOS/OEM) versions. However, they also include a full Greek alphabet, which takes over the code points that Unicode assigns to the CP437 Greek/Math characters, and the new glyphs don't necessarily look the same.
  Instead of just dropping the CP437 originals, I tried to retain them at remapped code points for the sake of completeness. Here are the changes:

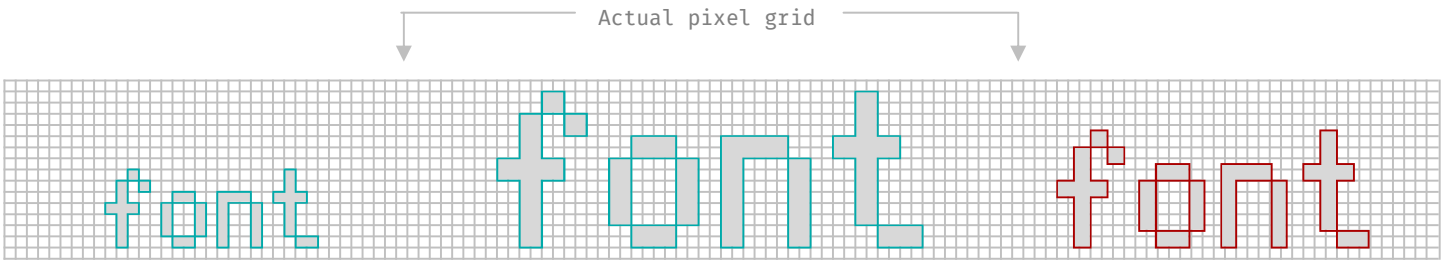
CP437 glyph	Canonical Unicode value	Modified 'PxPlus' mapping
α (E0h)	α (03B1/Greek small Alpha)	ɑ (0251/Latin small Alpha)
ß (E1h)*	ß (00DF/Latin small Sharp S)	ϐ (03D0/Greek Beta symbol)
Γ (E2h)	Γ (0393/Greek capital Gamma)	ᴦ (1D26/Small capital Gamma)
π (E3h)	π (03C0/Greek small Pi)	ᴨ (1D28/Small capital Pi)
Σ (E4h)	Σ (03A3/Greek capital Sigma)	∑ (2211/N-ary Summation)
σ (E5h)	σ (03C3/Greek small Sigma)	ơ (01A1/Small o with horn)
τ (E7h)	τ (03C4/Greek small Tau)	ᴛ (1D1B/Small capital T)
Φ (E8h)	Φ (03A6/Greek capital Phi)	ɸ (0278/Latin small Phi)
Θ (E9h)	Θ (0398/Greek capital Theta)	ϴ (03F4/Capital Theta symbol)
Ω (EAh)	Ω (03A9/Greek capital Omega)	Ω (2126/Ohm symbol)
δ (EBh)	δ (03B4/Greek small Delta)	ẟ (1E9F/Latin small Delta)
ø (EDh)	φ (03C6 Greek small Phi)	∅ (2205/Empty set)
∈ (EEh)	ε (03B5/Greek small Epsilon)	ϵ (2208/Element of)

* inconsistent: looks like a Beta in some fonts (**ϐ**/03B2) and like a Sharp S in others (**ß**/00DF). Both of those have their own code points, so the new mapping simply preserves the original (whatever it looks like).

DISPLAY CONSIDERATIONS

Size Matters

Outlined TrueType fonts are by definition scalable, so in theory they can be used at any size. But to reproduce oldschool raster characters, the outlines follow the contours of the original pixel glyphs, so the rendering will be optimal only if the outline snaps to the pixel grid of your target display; otherwise you *will* get fugly scaling artifacts. To make a long story short, they'll only look best at their native pixel sizes, or at integer multiples thereof.



With the bitmap ('Bm') fonts this is less tricky, since there are no outlines - they're always rendered on grids of whole pixels, so the correct sizes are easy to find and the result should always appear sharp. The mixed-format ('Mx') fonts also include embedded bitmaps, but these bitmaps are only used when the native size is selected: other sizes will display as outlines.

It *should* be easy to determine this native pixel size, because it's clearly spelled out for each font in the index. Problem is, most current GUI environments make it more difficult than it should be – sizes may have scaling factors applied, may not be easily selectable, and are often not measured in pixels at all.

This doesn't matter very much for displays with ultra-high DPI resolutions (say 200 or higher), because they make scaling artifacts much less noticeable so you can often get away with arbitrary sizes, but most current monitors aren't like that.

Pixels, Points, and DPI Resolutions

One obstacle is that most operating systems tend to measure text sizes in points (**pt**), a unit used in print and typesetting, rather than pixels, the physical unit that display devices operate with. This oddity is with us today largely for historical reasons: computer GUIs were becoming popular right when Desktop Publishing was all the rage, and the assumption was that if you bothered with the sizes of on-screen text at all, it was because you wanted a hard copy on dead trees.

That's where this particular unit came from, but there was a more general reason to separate type sizes from the physical characteristics of the monitor: the concept of **device independence** mandates that things should always have a consistent appearance, regardless of your output device. A swell idea in theory, but in practice little attention was paid to mapping logical to physical units. Monitor resolutions kept varying, but Windows settled on the assumption of **96** DPI (where 1pt = 0.75px), Mac OS chose **72** DPI (1pt = 1px), and others went their own ways.

Decades later, we still have to deal with GUI scaling that never actually matches the monitor's physical resolution, with applications that may or may not be aware of DPI adjustments, with mobile devices that make their own assumptions, and a whole bunch of fun problems like that. None of it really helps with our use case (finding the optimal sizes for pixel-outline fonts), which is naturally marginal and was never really considered.

The Bottom Line: Optimal Sizes

In a nutshell, you'll have to experiment, but a good guideline would go something like this:

- Find out how your OS converts points to pixels for your particular display settings. For **most** fonts this should be enough, and you can simply select the size that corresponds to the original bitmap font's height in pixels.
- For other fonts the desired size will differ a little bit. Sometimes an integer pixel size results in a **fractional** point size - e.g. in Windows at 96 DPI, 19 pixels = 14.25 points; Windows text rendering seems to work at 0.5pt increments, so you can't quite get there. Even when Windows can *render* the right size (e.g. 14 pixels = 10.5 points), you can't always *select* it, because the default font dialogs simply don't accept fractional numbers.

  To get around these hassles, font metrics for certain sizes have been tweaked, so that the size you want to select might differ from the naive pixel-to-point conversion.
- Basically: if the original charset's pixel height divides by **4**, you're good – just select the corresponding point/pixel size. If it doesn't, **round up** to the next multiple of 4, and treat the font as if it were that size.

  For example, if the original oldschool character size is 8x**14**, round the height up to **16** pixels, and select the size that corresponds to that (at 96 DPI, that's **12** pt). For 8x**19**, round up to **20** pixels (=**15** pt at 96 DPI). This holds true even when you're selecting the size in pixels rather than points.

All of the above applies to .ttf fonts only. Bitmap fonts (.fon) naturally come with the right pixel sizes built-in; web fonts (.woff) can always be sized flexibly using any supported CSS unit, which includes pixels, so these metric adjustments weren't necessary.

Aspect ratio

As discussed somewhere above, modern displays have square pixels as a rule. However, the original fonts were (mostly) used in various **non-square** pixel resolutions, so 1:1 pixel reproductions do not preserve the aspect ratio: most of these fonts will be somewhat squashed vertically, compared to their appearance on original hardware.

Version 2.0 includes aspect-corrected ('Ac') versions of the TrueType fonts, each according to its original pixel aspect ratio (both the "square" and the "corrected" ratios are noted for each font in the index). The outlines in these fonts similarly follow the pixel grids of the original characters, but here they aren't square, so in general the rendering on a modern display won't be 100% crisp. With good sub-pixel anti-aliasing however (such as ClearType on Windows, *if* it's tuned properly) the result can still be quite good.

Only the .ttf fonts have aspect-corrected versions, since:

- Aspect correction doesn't really work with .FON bitmap fonts. Technically the format *does* support it, because it was created way back when pixels still weren't uniformly square, and allows you to specify separate vertical and horizontal resolutions. The results are pretty horrible however – the renderer seems to interpret this as a green light to distort the fonts out of recognition if the desired aspect ratio doesn't precisely fit the physical pixel grid. So, no go with bitmaps.
- The web fonts (.woff) really don't need separate aspect corrected versions, since you can always do it yourself using CSS.

So how was the "correct" aspect ratio derived for each font? Knowing the original resolution ("SAR" - 'storage aspect ratio'), and the intended aspect of the display area ("DAR" – 'display aspect

ratio'), you can derive the "PAR" (pixel aspect ratio): PAR = DAR/SAR. Whenever this ratio is too unwieldy, I've chosen to round it to a close enough number which is easier to work with.

- **Example:** standard VGA text mode runs at **720x400** on a **4:3** monitor, so the pixel aspect ratio is (4:3)/(720:400) = 20:27 = 0.740740...; for practical reasons I've rounded this to 3:4 = 0.75.

Certain fonts *were* originally used in square-pixel resolutions, and for others the pixel aspect was close enough to square that the difference is barely noticeable. For these fonts aspect-corrected versions aren't included.

- **Example:** 40-column text mode on the Philips :YES runs at **320x256** on **4:3** monitors, so the pixel aspect ratio is (4:3)/(320:256) = 16:15 = 1.0666...; close enough to 1, so it's treated as square.

Some of the original fonts were seen in more than one display mode, so the resolutions and pixel ratios could vary. To avoid going completely overboard, in these cases I simply selected the most common pixel aspect as the "correct" one.

(Incidentally: a pixel aspect ratio like "3:4" means you can still get an ideal display on a square-pixel monitor, if you render the text at 4x its original pixel height: the rectangular blocks enclosed by the outline will always be 3 pixels wide by 4 tall, so you shouldn't get unseemly pixel-fraction artifacts. Of course, for most normal usage this isn't very practical.)

Anti-Aliasing

Current operating systems usually apply some sort of anti-aliasing ("smoothing") to scalable text. With a proper algorithm (hinted sub-pixel AA, not grayscale) and when configured correctly, this does improve legibility; I wouldn't really recommend disabling it completely, although if that works for you then by all means go ahead and turn it off. In the specific case of TrueType fonts based on pixel outlines, however, it mostly hinders rather than helps. In some cases you may get a sort of smearing/fringing effect on the TrueType fonts, even if you pick the right size to perfectly align them with your monitor's pixel grid.

In recent **Windows** versions this happens even with ClearType, which usually produces sharper rendering than other algorithms. For a nice crisp appearance, you should stick to the font's "native" pixel size and use either the '**Bm**' (bitmap) or '**Mx**' (mixed-format) variants, whichever plays nicer with your application.

In most **Linux** distros you should be able to use fontconfig[→] to control the anti-aliasing of text, and the advantage here is that it can be enabled or disabled on a per-font basis: you can keep your smoothed-out text, and turn off AA only for pixel-based fonts (like the ones in this collection). If your application or rendering engine isn't handling the '**Mx**' fonts properly, this sort of fine-tuning can compensate for that.

There's also the newer .otb format that provides bitmap-only 'Bm' versions for Linux, but only preliminary testing has been done so far so I can't be sure how well that works. If it does, great.

On the **Mac**, at least on macOS >=10.14, AA is grayscale-only and you cannot turn it off globally. Embedded-bitmap rendering doesn't seem to be supported, so the 'Mx' fonts won't help out there. As icing on the cake, "font smoothing" (enabled by default!) doesn't do what it says on the tin: it simply makes all fonts artificially bolder[→], which is generally a bad idea, and even more so with fonts like these.

Specific Mac applications (e.g. iTerm2, MacVim) *can* be configured to disable anti-aliasing, so that's your best bet wherever it is supported – as long as you're using the 'Px' fonts, it'll look just fine. For all other cases, just ensure that "font smoothing" is off. In any event, the aforementioned advice about native/optimal font sizes still holds true. (Thanks to kerframil for these findings.)

MISCELLANEOUS USAGE NOTES

Windows Console / Command Prompt / Terminal

At least in **Windows 7** and earlier, only bitmap (raster) fonts can be freely selected for use in the Windows console. TrueType fonts require a registry edit – to add them to the list, you'll have to start regedit.exe and open this key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Console\TrueTypeFont
```

You'll probably see two fonts defined (Lucida Console and Consolas), as "0" and "00" respectively. To enable more TrueType fonts, add a new String Value with one more 0 in its name (the first one you add will be **000**, then **0000** and so on). In the Value field simply enter the font's name, e.g. **MxPlus IBM VGA 9x16**. Hit OK and exit the editor. The font will now be available in the Properties dialog for console windows.

In **Windows 10** this no longer seems to be the case. Monospaced TrueType fonts *are* selectable for console windows; in fact, as of this writing, "v2" of the Console Host (activated with the 'ForceV2' registry setting[→]) does not support bitmap (raster) fonts at all.

There is also the newer Windows Terminal[→]; I haven't tested it too extensively, but Microsoft's own docs showcase a font from this pack[→], so apparently it'll work just fine. :) The JSON settings to specify the font face and size can be seen in the linked page.

Web usage

For embedding these fonts in web pages, the .woff format is recommended. You simply need to define a CSS @font-face rule as follows, and then specify the defined 'font-family' as usual:

```
@font-face {
  font-family: 'IBM VGA 8x16';
  src: url(WebPlus_IBM_VGA_8x16.woff) format('woff');
  font-weight: normal;
  font-style: normal;
}
```

'Web437' fonts only include the 256 characters present in DOS code page 437; if your web document contains characters outside this range, they will usually be displayed using a fallback font. To control this, you can specify a second @font-face rule for one of the 'Plus' fonts – preferably one with the **same size** as your main font – and then set it as your fallback font like so:

```
.vga8 {
  font-family: 'Verite 8x16', 'IBM VGA 8x16', monospace;
  font-size: 16px;
  line-height: 16px;
}
```

In this example 'Verite 8x16' contains only the CP437 characters. Non-CP437 characters like the Euro symbol '**€**' will be taken from the 'IBM VGA 8x16' font, since it's specified as the second priority, and contains '€' as part of the 'Plus' charset. If the character doesn't exist in the 'Plus' font either, the browser will move on to the third priority specified (your default monospaced font).

The webfonts implement only the square-pixel variants - if you want aspect correction, you can simply use a **CSS transform.** As a general rule, you should always do your scaling on the horizontal axis, to take advantage of subpixel anti-aliasing where available. For instance, 'IBM MDA' has a 2:3 pixel aspect, so you'd want to do it like this:

```
.mda-scaled {
  font-family: 'IBM MDA', monospace;
  font-size: 14px;
  line-height: 14px;
  transform: scaleX(0.6667);
}
```

As for sizes and line heights, for best results you should always use the same pixel height as the original typeface (or an integer multiple). It's still a good idea to keep in mind that in CSS, a pixel is not a pixel[→]; but that distinction applies mainly to high-DPI screens and devices – where it doesn't matter very much if you stray from the optimal font size, since scaling artifacts are much less apparent. On most monitors with a normal resolution, where size does matter, a CSS pixel generally *is* a physical pixel.

Working with CP437/DOS Text Files

Text-based material which originates with DOS can be difficult to handle properly in a current OS, especially if it uses non-ASCII/graphical characters. This is usually the case with .NFO/ASCII/ANSI artwork[→], for instance. To display and edit it correctly, two things have to happen:

- Your viewer/editor needs to support the encoding (code page 437).
- The font you're using should have the ability to display it.

On Windows the 'Px437', 'Ac437' and 'Bm437' fonts in this pack fulfill the latter condition; they're recognized as supporting the "**DOS/OEM**" charset. The 'Mx' (mixed outline/bitmap) fonts also include 'Mx437' versions, but disappointingly enough, the same trick[→] which forces Windows to display them properly also prevents it from recognizing them as DOS/OEM fonts. Unless and until Microsoft chooses to support embedded bitmaps without requiring this sort of insanity, they can't be used for CP437 text.

If your software doesn't support the CP437 encoding, or doesn't know how to map it to something like UTF-8, you can use a converter: a nice one can be found at CodeLobe[→].

On the web things are even more problematic, since most browsers don't support CP437[→]. In fact WHATWG's encoding standard decrees that user agents *must* not[→] support it, as it's missing from

their list of kosher encodings. A particularly silly decision when other DOS charsets – even less common ones! – coexist happily enough in the approved list.

Luckily, solutions exist: I can recommend the RetroTxt browser extension[→], which not only converts CP437 text to UTF-8 for display, but comes with a big bunch of tweakable options (and includes a few of these oldschool PC fonts built-in).

Older Windows versions (95/98/ME/NT4)

As far as I know, .ttf fonts from this package won't work in Windows versions earlier than Windows 2000. The reason is that they actually use OpenType tables (like most TrueType outline fonts in the wild these days), and OpenType wasn't supported out of the box before Windows 2000.

If your retro system is running one of these Windows versions, it's *possible* that ATM Light[→] from Adobe will make these .ttf fonts usable, but I haven't tested this myself.

DIFFS & ERRATA

This section lists instances where the versions in the Font Pack differ from the original raw bitmap designs. This doesn't cover custom additions, like in the remapped and 'Plus' versions, which are noted separately. Items here represent only errors and obvious visual issues with the original glyphs, which I've tried to spot and correct, in the interest of usability.

- **Acer VGA *x8:** fixed misaligned box drawing character (#190/0xBE)
- **DG One Alt** (and **-2y**)**:** fixed misaligned box drawing character (#208/0xD0)
- **DOS/V re. PRC16/19:** fixed erratic baseline for 'p', 'q' and several characters in the #128-#165 block
- **Master 512*:** changed character #90/0x60 from a duplicate of '£' (#156/0xA3) to a backquote (`) as standard
- **NEC MultiSpeed** (and **-2x**): fixed misaligned box drawing character (#189/#0xBD)
- **Paradise132 7x9:** fixed misaligned characters 'g','p','q'
- **SeequaCm** (and **-2y**)**:** fixed surplus floating pixel over block character (#220/0xDC)
- **Tandy2K** (and **-2x**)**:** fixed misaligned box drawing character (#208/0xD0)
- **Trident 8x11:** fixed inconsistent baseline (characters #128-#172)

VERSION HISTORY

v2.2 (2020-11-21)

- **20** new fonts added:
  - **PC-compatible computers**: NEC MultiSpeed (4 fonts), Schneider EuroPC (3 fonts), Tandy 1000 Video I -2x (1 font), Tandy 1000 Video II -2x (1 font)
  - **PC semi-compatibles**: DEC Rainbow 100 (4 fonts), HP 150 Touchscreen (1 font), Robotron A7100 (1 font), Tandy 2000 graphics modes (3 fonts), Zenith Z-100 (2 fonts)

- Fixes/changes: in all 'Plus' fonts, the characters **ď**, **ť**, **ľ**, **Ľ** now consistently use the modified form of the háček/caron diacritic (as used in Czech/Slovak)

v2.1 (2020-10-30)

- **72** new fonts added:
  - **PC-compatible computers**: Acer 710 Mono (1 font), HP 100LX (6 fonts), Olivetti MS-DOS (4 fonts), Sanyo MBC-55x (2 fonts), Sanyo MBC-775 (2 fonts), Sharp PC-3000 (4 fonts), TeleVideo Tele-PC (2 fonts)
  - **PC video hardware**: CL Eagle II (2 fonts), CL Eagle III (2 fonts), CL Stingray (4 fonts), InteGraphics VGA (2 fonts), Phoenix VGA (6 fonts), Sigma Designs RealMagic (6 fonts), STB AutoEGA (2 fonts), Early Trident/TVGA8800CS (7 fonts), Tseng EVA-480 ET2000 (2 fonts), Various/ACM VGA (6 fonts)
  - **PC semi-compatibles**: Acorn BBC Master 512 (6 fonts), Atari Portfolio (1 font), Philips :YES G-2x (1 font), Texas Instruments PC/PPC (1 font), Wang Professional Computer (3 fonts)

- New extended charsets ('Plus' versions) for existing fonts: Cordata PPC-21, ToshibaSat 8x8, ToshibaSat 9x8

- Format updates:
  - **'Web'** (.woff webfonts) now have separate '437'/'Plus' variants, like the rest ('Plus' remaps some of the Greek/Math glyphs in CP437, so it's not exactly a superset of the latter)
  - **'Bm'** (bitmap) fonts are now available for Linux too, in the .otb format – both '437'/'Plus' charsets. See the format notes about these.

- Removed fonts:
  - Olivetti MxVGA: superseded by **CL Stingray** (the VGA chipset/firmware used in these laptops)
  - DG One Alt, -2y: merged with **DG One, -2y** as Bold variants of these font families

- Fixed a few minor errors in the following fonts:
  - **Cordata PPC-400** 'Plus' variants (characters 'ē', 'Ų', 'ų')
  - **ToshibaSat 8x14** and **9x14** 'Plus' variants (character 'Є')
  - **PhoenixEGA*** (character '☺')

v2.0 (2020-07-12)

- **131** new font families added, from **52** different sources (hardware/firmware/software)
- 3 new font variants:
  - **'Mx'** – Mixed-Format: TrueType versions with embedded bitmaps (for sharper rendering)
  - **'Ac'** - Aspect-Correct TrueType versions
  - **'Web'** - .woff web fonts

- These fonts have been renamed, to keep things more consistent and less confusing ("*" = wildcard):
  - AmstradPC1512* renamed **Amstrad PC*** (other Amstrad PC models use the same font)
  - IBM EGA8/EGA9/VGA8/VGA9 now specify the full size (**IBM EGA 8x14** etc.), as more 8/9-dot sizes have been added
  - IBM PS/2thin* renamed **IBM Model3x Alt*** (not all PS/2 models have it)
  - IBM ISO* renamed **IBM DOS ISO***
  - Kaypro2K* renamed **Kaypro2K G*** (as it's specific to graphics modes)
  - TandyOld* renamed **Tandy1K-I** (there's 'Tandy2K' as well now, and the video chipset in the early 1000 series was officially named 'Tandy Video I')
  - TandyNew* renamed **Tandy1K-II** (likewise)
  - For all Tandy1K fonts, TV* also becomes **200L***, and 225* becomes **225L***
  - VGA SquarePx renamed **AST Premium Exec** (other square-pixel VGA fonts have been added, and the name should indicate the origin)

- These fonts have been removed, or rather replaced with better/fuller versions:
  - AMI BIOS* fonts are superseded by **AMI EGA 8x8**
  - ITT BIOS* fonts are superseded by **ITT Xtra***
  - ToshibaLCD 8x8 is superseded by **ToshibaTxL1 8x8**
  - ToshibaLCD 8x16 is superseded by **ToshibaTxL2 8x16**

- Bugfix: corrected all font styles to "Regular" across the board
- Adjusted the Ascent metric for certain fonts taller than 16 pixels, to make size selection more consistent

If you're upgrading from v1.x: Due to the above changes, I **strongly recommend** uninstalling all older "Bm" and "Px" fonts before you install any from v2.0. Otherwise you'll end up with duplicates and an inconsistent set.

v1.01 (2020-04-22)

- Bugfix release: fixed single-character errors in the fonts 'ITT BIOS'/'-2y' (Px437/Bm437), 'IBM BIOS'/'-2x'/'-2y' (PxPlus), and 'IBM CGA'/'-2y' (PxPlus)

v1.0 (2016-01-06)

- Initial release

CREDITS & ACKNOWLEDGEMENTS

Font adaptations, documentation, website: **VileR**

Thanks to:

- John Elliot[→], MinusZeroDegrees[→], **ripsaw8080**, Ardent Tool of Capitalism[→], Great Hierophant[→], **NewRisingSun** - for helpful information

- **alecv, alexanrs, Robbert van der Bijl,** Rich Cini[→], John Elliot[→], **fs5500, Yossi Gil, Bill Hart, James Howard, jesolo, Jo22, keropi,** MCbx[→], Olivrea[→], **per, Thomas Schneider, SomeGuy, Svenska,** Trixter[→], T-Squared[→], **Foone Turing,** VGA Museum[→] - for providing/verifying sources and ROM dumps for several charsets, off real hardware and localized video cards

- **Rebecca G. Bettencourt/Kreative Korp**[→] - for responding to my inquiries with a surprise open-source release of the awesome Bits'n'Picas vectorizer
- **Felix Rosén, erkcan, kerframil** - for testing and bug reports

Tools used:

- ROM charset extraction: **Binxelview**[→] by **Brad Smith**, **Crystal Tile 2**[→] by **angel-team**
- Bitmap font editing: **Fony 1.4.7**[→] by **hukka**
- Bitmap-to-outline vectorization: **Bits'n'Picas**[→] by **Kreative Korp**
- TrueType font editing, fine-tuning, re-encoding etc.: **FontForge**[→] by **George Williams and the FontForge Project**
- Windows .TTF testing/viewing: **SIL ViewGlyph v1.81.0**[→] by **Bob Hallissy/SIL International**
- DOS screen font manipulation and dumping: **fntcol16**[→] by **Yossi Gil**, **Fontraption**[→] by yours truly
- DOS .CPI file manipulation:- **CPIED 1.3c**[→] by **Balthasar Szczepański**

CONTACT

I can be reached at: **email** - viler/AT/int10h/DOT/org
www - http://int10h.org[→]
blog - https://int10h.org/blog[→]

If you have an interesting machine or video card, which fits the scope of this project, and it has a non-generic text mode font (i.e. it doesn't just duplicate the IBM charsets), please contact me - contributions to this collection are always appreciated.

LEGAL STUFF

In addition, there are a couple of points that should be made quite clear. Out of all the comments I've received about this collection, a small minority of people have assumed a priori that its legal status is dubious at best, that the original binary charsets are protected by copyrights (which are being infringed), and/or that I act in bad faith by taking credit for my work on it.

It should be noted that I've done quite a bit of research into this issue, and everything I've turned up supports the conclusion that there's nothing illegal about this work. US law (which happens to have this site under its jurisdiction) does not consider typefaces to be copyrightable material, and making derivative versions does not infringe on anyone's intellectual property. If I was re-distributing the original programs which render these fonts onto a computer screen - such as firmware/driver code, or TrueType files that were the *original distribution format* - that would've been a different story. However, all fonts here were originally simple bitmap data; I am providing new software to reproduce these old glyphs in different (and sometimes expanded) formats.

There's enough material online to consult on this subject if you wish (the comp.fonts FAQ[→] is a good starting point). It's probably safe to consider the lack of past precedents, too. The most commonly-seen of these fonts are the ones from IBM, which have been duplicated bit-for-bit by countless manufacturers in their own hardware - from Hercules to nVidia - and sold with this hardware practically everywhere on the planet. I couldn't find information about a single lawsuit resulting from that (as opposed to, say, companies that infringed on IBM's BIOS code). By itself, lack of precedent is not a

sufficient argument, but it does support the one already made - and precedents for NON-infringement certainly exist (e.g. Eltra Corp. v. Ringer).

As for bad-faith arguments ("you just ripped a bunch of fonts and repackaged them!"), rest assured that there's more to it than that. Creating this collection has involved a lot of hard research, conversion and optimization work to ensure that the originals are represented both faithfully and usefully, something that is far from trivial; there are also my own additions in the 'Plus' versions and the remapped charsets. This work surely merits *some* credit, and that is the only credit I claim.

- VileR