ttfautohint

Werner Lemberg

Version 1.8.4

Contents

1 Introduction

ttfautohint is a library written in C that takes a TrueType font as the input, removes its bytecode instructions (if any), and returns a new font where all glyphs are bytecode hinted using the information given by FreeType's auto-hinting module. The idea is to provide the excellent quality of the auto-hinter on platforms that don't use FreeType.

The library has a central API function, `TTF_autohint`, which is described below (chapter 5).

Bundled with the library there are two front-end programs, `ttfautohint` and `ttfautohintGUI` (chapter 2), being a command line program and an application with a Graphics User Interface (GUI), respectively.

1.1 What exactly are hints?

To cite Wikipedia:

> **Font hinting** (also known as **instructing**) is the use of mathematical instructions to adjust the display of an outline font so that it lines up with a rasterized grid. At low screen resolutions, hinting is critical for producing a clear, legible text. It can be accompanied by antialiasing and (on liquid crystal displays) subpixel rendering for further clarity.

and Apple's TrueType Reference Manual:

> For optimal results, a font instructor should follow these guidelines:
>
> - At small sizes, chance effects should not be allowed to magnify small differences in the original outline design of a glyph.
> - At large sizes, the subtlety of the original design should emerge.

In general, there are three possible ways to hint a glyph.

1. The font contains hints (in the original sense of this word) to guide the rasterizer, telling it which shapes of the glyphs need special consideration. The hinting logic is partly in the font and partly in the rasterizer. More sophisticated rasterizers are able to produce better rendering results.

   This is how PostScript Type 1 and CFF hints work.

2. The font contains exact instructions (also called *bytecode*) on how to move the points of its outlines, depending on the resolution of the output device, and which intentionally distort the (outline) shape to produce a well-rasterized result. The hinting logic is in the font; ideally, all rasterizers simply process these instructions to get the same result on all platforms.

   This is how TrueType hints work.

3. The font gets auto-hinted (at run-time). The hinting logic is completely in the rasterizer. No hints in the font are used or needed; instead, the rasterizer scans and analyzes the glyphs to apply corrections by itself.

This is how FreeType's auto-hinter works; see below (chapter 3) for more.

1.2 What problems can arise with TrueType hinting?

While it is relatively easy to specify PostScript hints (either manually or by an auto-hinter that works at font creation time), creating TrueType hints is far more difficult. There are at least two reasons:

- TrueType instructions form a programming language, operating at a very low level. They are comparable to assembler code, thus lacking all high-level concepts to make programming more comfortable.

  Here an example how such code looks like:

  ```
  SVTCA[0]
  PUSHB[ ]  /* 3 values pushed */
  18 1 0
  CALL[ ]
  PUSHB[ ]  /* 2 values pushed */
  15 4
  MIRP[01001]
  PUSHB[ ]  /* 3 values pushed */
  7 3 0
  CALL[ ]
  ```

  Another major obstacle is the fact that font designers usually aren't programmers.

- It is very time consuming to manually hint glyphs. Given that the number of specialists for True-Type hinting is very limited, hinting a large set of glyphs for a font or font family can become very expensive.

1.3 Why ttfautohint?

The ttfautohint library brings the excellent quality of FreeType rendering to platforms that don't use FreeType, yet require hinting for text to look good – like Microsoft Windows. Roughly speaking, it converts the glyph analysis done by FreeType's auto-hinting module to TrueType bytecode. Internally, the auto-hinter's algorithm resembles PostScript hinting methods; it thus combines all three hinting methods discussed previously (section 1.1).

The simple interface of the front-ends (both on the command line and with the GUI) allows quick hinting of a whole font with a few mouse clicks or a single command on the prompt. As a result, you get better rendering results with web browsers, for example.

Across Windows rendering environments today, fonts processed with ttfautohint look best with ClearType enabled. This is the default for Windows 7 and higher. Good visual results are also seen in recent MacOS X versions and GNU/Linux systems (including Android, ChromeOS, and other mobile operating systems) that use FreeType for rendering glyphs.

1.4 'Smooth' hinting

Fundamentally, there are two approaches to hinting. The older approach, let's call it 'sharp', popular when text was rendered in pure black-and-white, was to make all stems round to full pixels so that in a text line, all stems would be either one pixel or (at a larger point size) two pixels. When grayscale antialiasing came about, this approach actually started harming the rendering rather than helping it, because the horizontal and vertical stems would render very dark but round or diagonal stems would render very light.

So a new approach was developed, let's call it 'fuzzy', where all stems and other elements are equalized so that in grayscale (or ClearType) rendering, they all are of roughly equal color. This means that stems are not rounded to full pixels but in fact to fractions of a pixel. However, with black-and-white renderers, this approach yields poor results because in black-and-white you cannot render a fraction of a pixel, so some stems become one pixel and some become two.

The TrueType auto-hinters in FontForge and FontLab Studio, to name two well-known font editors, take the 'sharp' approach, while the TrueType auto-hinter in ttfautohint takes the 'fuzzy' approach.

In theory, a hybrid approach is possible, using TrueType conditional hints: If the rasterizer is black-and-white, 'sharp' rendering could happen, while if the rasterizer is ClearType, the 'fuzzy' rendering could be used. It is not intended to add black-and-white auto-hinting to ttfautohint. However, it is planned to develop an interface so that ttfautohint can cooperate with font editors, providing this hybrid hinting.

2 `ttfautohint` and `ttfautohintGUI`

On all supported platforms (GNU/Linux, Windows, and Mac OS X), the GUI looks quite similar; the used toolkit is Qt, which in turn uses the platform's native widgets.

Both the GUI and console version share the same features, to be discussed in the next subsection.

Warning: ttfautohint cannot always process a font a second time. If the font contains composite glyphs, and option `-c` (subsection 2.3.12) is used, reprocessing with ttfautohint will fail. For this reason it is strongly recommended to *not* delete the original, unhinted font so that you can always rerun ttfautohint.

2.1 Calling `ttfautohint`

```
ttfautohint [OPTION]... [IN-FILE [OUT-FILE]]
```

The command-line binary, `ttfautohint`, works like a Unix filter, this is, it reads data from standard input if no input file name is given, and it sends its output to standard output if no output file name is specified.
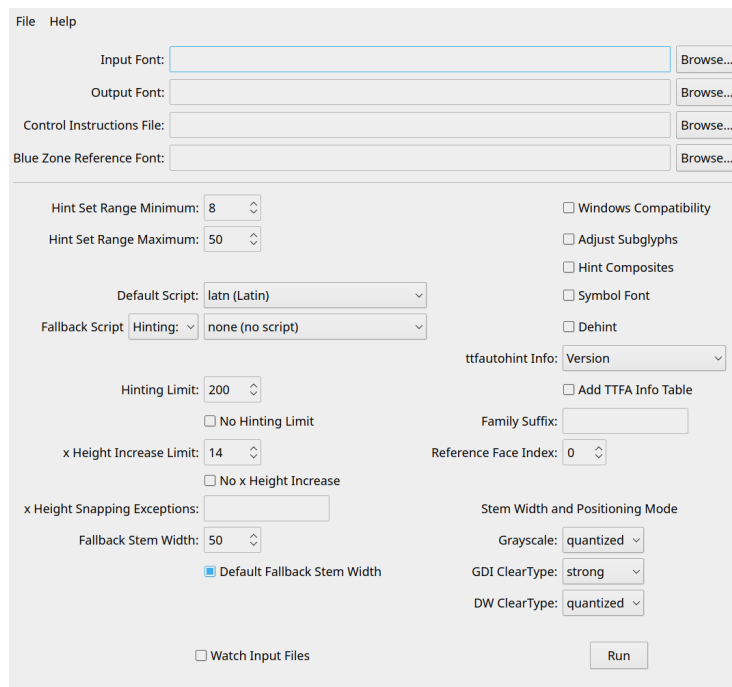
Figure 2.1: `ttfautohintGUI` on GNU/Linux running KDE

A typical call looks like the following.

```
ttfautohint -v -t -m foo-control.txt foo.ttf foo-autohinted.ttf
```

For demonstration purposes, here the same using a pipe and redirection. Note that Windows's default command line interpreter, `cmd.exe`, doesn't support piping with binary files, unfortunately.

```
cat foo.ttf | ttfautohint -v -t -m foo-control.txt > foo-autohinted.ttf
```

2.2 Calling `ttfautohintGUI`

```
ttfautohintGUI [OPTION]...
```

`ttfautohintGUI` doesn't send any output to a console; however, it accepts (almost) the same command line options as `ttfautohint` to provide customized default values of the GUI.

The following command line options are not available in `ttfautohintGUI`: `--control-file` (subsection 2.3.1), `--reference` (subsection 2.3.2), `--reference-index` (subsection 2.3.18). Note, however, that the corresponding functionality can be selected interactively.

Two options, namely `--ttfa-info` and `--debug`, emit information at standard output and standard error, respectively; they are thus not available in `ttfautohintGUI`, too, and there is no corresponding GUI functionality either.

2.3 Options

Long options can be given with one or two dashes, and with and without an equal sign between option and argument. This means that the following forms are acceptable: `-foo=`*bar*, `--foo=`*bar*, `-foo` *bar*, and `--foo` *bar*.

Below, the section title refers to the command's label in the GUI (if applicable), then comes the name of the corresponding long command line option and its short equivalent, followed by a description.

Background and technical details on the meaning of the various options are given afterwards (chapter 3).

2.3.1 Control Instructions File

`--control-file=`*file*, `-m` *file*
> Specify the name of a control instructions file to manually tweak the hinting process. This feature can be used to correct glitches in ttfautohint's hinting algorithm. The syntax used in a control instructions file is given below (chapter 4).
>
> `ttfautohintGUI` doesn't have this command line option.

Uppercase Letters

- AAAAA
- BBBBB
- CCCCC
- DDDDD
- EEEEE
- FFFFF
- GGGGG
- HHHHH
- IIIII
- JJJJJ
- KKKKK
- LLLLL
- MMMMM
- NNNNN
- OOOOO
- PPPPP
- QQQQQ
- RRRRR
- SSSSS
- TTTTT
- UUUUU
- VVVVV
- WWWWW
- XXXXX
- YYYYY
- ZZZZZ

Uppercase Letters

- AAAAA
- BBBBB
- CCCCC
- DDDDD
- EEEEE
- FFFFF
- GGGGG
- HHHHH
- IIIII
- JJJJJ
- KKKKK
- LLLLL
- MMMMM
- NNNNN
- OOOOO
- PPPPP
- QQQQQ
- RRRRR
- SSSSS
- TTTTT
- UUUUU
- VVVVV
- WWWWW
- XXXXX
- YYYYY
- ZZZZZ

Figure 2.2: Fira Regular and Bold (version 4.106), auto-hinted with ttfautohint and displayed at 16px using Internet Explorer 11 under Windows 8.1. The bold series shown on the right side uses the regular variant as the reference font.

2.3.2 Blue Zone Reference Font

--reference=*file*, -R *file*

> Derive all blue zones from the given font, which can either be a normal TrueType font or a TrueType collection – for the latter you can select the face index with a separate option (subsection 2.3.18).
>
> Use this to harmonize font families, avoiding ugly height differences at small sizes.
>
> To make this work the reference font must obviously be similar enough to the font to be hinted; in particular, it must have proper blue zone characters so that ttfautohint can derive blue zones at all.
>
> ttfautohintGUI doesn't have this command line option.

2.3.3 Hint Set Range Minimum, Hint Set Range Maximum

See 'Hint Sets (section 3.5)' for a definition and explanation.

--hinting-range-min=*n*, -l *n*

> The minimum PPEM value (in pixels) at which hint sets are created. The default value for *n* is 8.

`--hinting-range-max=`*n*, `-r` *n*
> The maximum PPEM value (in pixels) at which hint sets are created. The default value for *n* is 50.

Increasing the range given by `-l` and `-r` normally makes the font's bytecode larger.

2.3.4 Default Script

`--default-script=`*s*, `-D` *s*
> Set default script to tag *s*, which is a string consisting of four lowercase characters like `latn` or `dflt`. It is needed to specify the OpenType default script: After applying all features that are handled specially (like small caps or superscript), ttfautohint uses this value for the remaining features. The default value is `latn`. See below (section 3.10) for more details.

2.3.5 Fallback Script

`--fallback-script=`*s*, `-f` *s*
> Set fallback script to tag *s*, which is a string consisting of four characters like `latn` or `dflt`. It gets used for all glyphs that can't be assigned to a script automatically. The default value is `none`. See below (section 3.9) for more details.

`--fallback-scaling`, `-S`
> Use scaling for glyphs covered by the fallback script, not hinting. See below (section 3.9) for more details.

2.3.6 Hinting Limit

`--hinting-limit=`*n*, `-G` *n*
> The *hinting limit* is the PPEM value (in pixels) where hinting gets switched off (using the `INSTCTRL` bytecode instruction, not the `gasp` table data); it does not influence the file size. The default value for *n* is 200, which means that the font is not hinted for PPEM values larger than 200.
>
> Note that hinting in the range 'hinting-range-max' up to 'hinting-limit' uses the hinting configuration for 'hinting-range-max'.
>
> To omit a hinting limit, use `--hinting-limit=0` (or check the 'No Hinting Limit' box in the GUI). Since this causes internal math overflow in the rasterizer for large pixel values (> 1500px approx.) it is strongly recommended to not use this except for testing purposes.

2.3.7 x Height Increase Limit

`--increase-x-height=`*n*, `-x` *n*
> Normally, ttfautohint rounds the x height to the pixel grid, with a slight preference for rounding up (to use the terminology of TrueType's 'Super Round' bytecode instruction, the threshold is 5/8px). If this flag is set, values in the PPEM range 6 to *n* are much more often rounded up (setting the threshold to 13/16px). The default value for *n* is 14. Use this flag to increase the legibility of small sizes if necessary; you might get weird rendering results otherwise for glyphs like 'a' or 'e', depending on the font design.
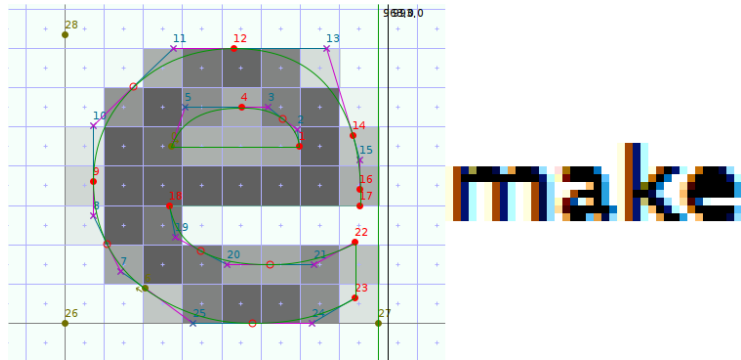
Figure 2.3: At 17px, without option `-x` and `-a qqq`, the hole in glyph 'e' looks very grey in the FontForge snapshot, and the GDI ClearType rendering (which is the default on older Windows versions) fills it completely with black because it uses B/W rendering along the y axis. ttfautohint's 'smooth' stem width algorithm intentionally aligns horizontal lines to non-integer (but still discrete) values to avoid large glyph shape distortions.
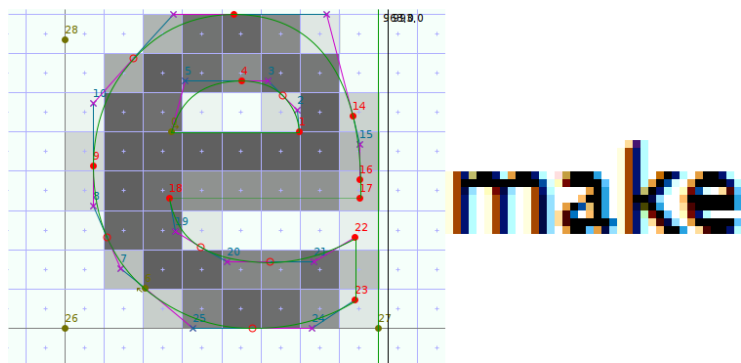
Figure 2.4: The same, this time with option `-x 17` (and `-a qqq`).

To switch off this feature, use `--increase-x-height=0` (or check the 'No x Height Increase' box in the GUI). To switch off rounding the x height to the pixel grid in general, either partially or completely, see 'x Height Snapping Exceptions (subsection 2.3.8)'.

The following FontForge snapshot images use the font 'Mertz Bold' from Vernon Adams.

2.3.8 x Height Snapping Exceptions

`--x-height-snapping-exceptions=`*string*, `-X` *string*
A list of comma separated PPEM values or value ranges at which no x height snapping shall be applied. A value range has the form *value*$_1$-*value*$_2$, meaning *value*$_1$ <= PPEM <= *value*$_2$. *value*$_1$ or *value*$_2$ (or both) can be missing; a missing value is replaced by the beginning or end of the whole interval of valid PPEM values, respectively (6 to 32767). Whitespace is not significant; superfluous commas are ignored, and ranges must be specified in increasing order. For example, the string `"7-9, 11, 13-"` means the values 7, 8, 9, 11, 13, 14, 15, etc. Consequently, if the supplied argument is `"-"`, no x height snapping takes place at all. The default is the empty string (`""`), meaning no snapping exceptions.

Normally, x height snapping means a slight increase in the overall vertical glyph size so that the

height of lowercase glyphs gets aligned to the pixel grid (this is a global feature, affecting *all* glyphs of a font). However, having larger vertical glyph sizes is not always desired, especially if it is not possible to adjust the `usWinAscent` and `usWinDescent` values from the font's `OS/2` table so that they are not too tight. See 'Windows Compatibility ([subsection 2.3.10](#))' for more details.

2.3.9 Fallback Stem Width

`--fallback-stem-width=`*n*, `-H` *n*
> Set the horizontal stem width (hinting) value for all scripts that lack proper standard characters in the font. The value is given in font units and must be a positive integer. If not set, ttfautohint uses a hard-coded default (50 units at 2048 units per EM, and linearly scaled for other UPEM values, for example 24 units at 1000 UPEM).
>
> For symbol fonts, you need option `--fallback-script` too (to set up a script at all).
>
> In the GUI, uncheck the 'Default Fallback Stem Width' box to activate this feature.

2.3.10 Windows Compatibility

`--windows-compatibility`, `-W`
> This option makes ttfautohint add two artificial blue zones, positioned at the `usWinAscent` and `usWinDescent` values (from the font's `OS/2` table). The idea is to help ttfautohint so that the hinted glyphs stay within this horizontal stripe since older versions of Windows clip everything that lies outside.
>
> There is a general problem with tight values for `usWinAscent` and `usWinDescent`; a good description is given in the [Vertical Metrics How-To](#). Additionally, there is a special problem with tight values if used in combination with ttfautohint because the auto-hinter tends to slightly increase the vertical glyph dimensions at smaller sizes to improve legibility. This enlargement can make the heights and depths of glyphs exceed the range given by `usWinAscent` and `usWinDescent`.
>
> If ttfautohint is part of the font creation tool chain, and the font designer can adjust those two values, a better solution instead of using option `-W` is to reserve some vertical space for 'padding': For the auto-hinter, the difference between a top or bottom outline point before and after hinting is less than 1px, thus a vertical padding of 2px is sufficient. Assuming a minimum hinting size of 6ppem, adding two pixels gives an increase factor of 8÷6 = 1.33. This is near to the default baseline-to-baseline distance used by TeX and other sophisticated text processing applications, namely 1.2×designsize, which gives satisfying results in most cases. It is also near to the factor 1.25 recommended in the abovementioned how-to. For example, if the vertical extension of the largest glyph is 2000 units (assuming that it approximately represents the designsize), the sum of `usWinAscent` and `usWinDescent` could be 1.25×2000 = 2500.
>
> In case ttfautohint is used as an auto-hinting tool for fonts that can be no longer modified to change the metrics, option `-W` in combination with '`-X "-"`' to suppress any vertical enlargement should prevent almost all clipping.

2.3.11 Adjust Subglyphs

`--adjust-subglyphs`, `-p`
> *Adjusting subglyphs* makes a font's original bytecode be applied to all glyphs before it is replaced

with bytecode created by ttfautohint. This makes only sense if your font already has some hints in it that modify the shape even at EM size (normally 2048px); in particular, some CJK fonts need this because the bytecode is used to scale and shift subglyphs (hence the option's long name). For most fonts, however, this is not the case.

2.3.12 Hint Composites

`--composites, -c`
> By default, the components of a composite glyph get hinted separately. If this flag is set, the composite glyph itself gets hinted (and the hints of the components are ignored). Using this flag increases the bytecode size a lot, however, it *might* yield better hinting results – usually, it doesn't.
>
> If this option is used (and a font actually contains composite glyphs), ttfautohint currently cannot reprocess its own output for technical reasons, see below (section 3.7).

2.3.13 Symbol Font

`--symbol, -s`
> Process a font that ttfautohint would refuse otherwise because it can't find a single standard character for any of the supported scripts.
>
> For all scripts that lack proper standard characters, ttfautohint uses a default (hinting) value for the standard stem width instead of deriving it from a script's set of standard characters (for the latin script, one of them is character 'o').
>
> Use this option – usually in combination with the `--fallback-script` (subsection 2.3.5) and/or `--fallback-stem-width` (subsection 2.3.9) option – to hint symbol or dingbat fonts or math glyphs, for example.

2.3.14 Dehint

`--dehint, -d`
> Strip off all hints without generating new hints. Consequently, all other hinting options are ignored. This option is intended for testing purposes.

2.3.15 ttfautohint Info

`--no-info, -n`
> Don't add ttfautohint version and command line information to the version string or strings (with name ID 5) in the font's `name` table. In the GUI, it corresponds to value 'None' in the 'ttfautohint info' combo box.
>
> This option is mutually exclusive with option `-I`.

`--detailed-info, -I`
> Add ttfautohint version and command line information to the version string or strings (with name ID 5) in the font's `name` table. In the GUI, it corresponds to value 'Version and Parameters' in the 'ttfautohint info' combo box.
>
> This option is mutually exclusive with option `-n`.

If neither `-n` nor `-I` is set, the string 'ttfautohint (vNNN)' gets added to the `name` table (with *NNN* the current version); this correponds to value 'Version' in the 'ttfautohint info' combo box.

2.3.16 Add TTFA Info Table

`--ttfa-table`, `-t`
>   Add an SFNT table called `TTFA` to the output font that holds a dump of all parameters; the data resembles the format of the `--debug` option's parameter listing. In particular, it lists all ttfautohint control instructions (which are *not* shown in the `name` table info). This option is mainly for archival purposes so that all information used to create a font is stored in the font itself. Note that such a `TTFA` table gets ignored by all TrueType rendering engines.
>
>   Forthcoming versions of the ttfautohint front-ends will be able to use this data so that a font can be processed another time with exactly the same parameters, thus providing a means for round-tripping fonts.

2.3.17 Family Suffix

`--family-suffix=`*string*, `-F` *string*
>   A string that gets appended to the family name in entries with IDs 1, 4, 6, 16, and 21 in the font's `name` table. Allowed input is ASCII in the range 0x20-0x7E except characters `%()/<>[]{}`.
>
>   Assuming an input family name 'Foo', a full name 'Foo Bold', and a family suffix ' 1', the output family name will be 'Foo 1' and the full name 'Foo 1 Bold'. For the PostScript name in ID 6, ttfautohint uses the suffix with space characters removed (for example 'Foo1Bold').
>
>   This option is mainly for testing purposes, enabling the operating system to simultaneously display several instances of a font that are processed with different ttfautohint parameters.

2.3.18 Reference Face Index

`--reference-index=`*n*, `-Z` *n*
>   Set the face index for the blue zone reference font (subsection 2.3.2) if the font is a TrueType collection (`.ttc`). For normal TrueType fonts, the value is always zero (which is also the default).
>
>   `ttfautohintGUI` doesn't have this command line option.

2.3.19 Stem Width and Positioning Mode

`--stem-width-mode=`*string*, `-a` *string*
>   ttfautohint provides three different algorithms for computing horizontal stem widths and the positioning of blue zones.
>
>   - 'natural': No adjustments to stem widths, discrete blue zone positioning. This is what FreeType uses for its 'light' (auto-)hinting mode. Essentially no glyph shape distortion, low contrast.

- 'quantized': Both stem widths and blue zone positions are slightly quantized to take discrete values. For example, stem values 50, 51, 72, 76, and 100 would become 50, 74, and 100 (or something similar). More glyph shape distortion but increased contrast.

- 'strong: If active, stem widths and blue zones are snapped and positioned to integer pixel values as much as possible. This gives high contrast, but glyph shape distortion can be significant.

These three algorithms are mapped onto three possible rendering targets.

- Grayscale rendering, with or without optimization for subpixel positioning (e.g., Android).

- 'GDI ClearType' rendering: the rasterizer version, as returned by the GETINFO bytecode instruction, is in the range 36 <= version <= 38 and ClearType is enabled (e.g., Windows XP).

- 'DirectWrite ClearType' rendering: the rasterizer version, as returned by the GETINFO bytecode instruction, is >= 39, ClearType is enabled, and subpixel positioning is enabled also (e.g., Internet Explorer 9 running on Windows 7).

GDI ClearType uses a mode similar to B/W rendering along the vertical axis, while DW ClearType applies grayscale rendering. Additionally, only DW ClearType provides subpixel positioning along the x axis. For what it's worth, the rasterizers version 36 and version 38 in Microsoft Windows are two completely different rendering engines.

[Note that the GDI framework on Windows 10 no longer uses B/W rendering along the vertical axis; we consequently treat it as DW ClearType also. We test this by looking at bit 11 of the GETINFO instruction, which was introduced in rasterizer version 40.]

The command line option expects *string* to contain exactly three letters with possible values 'n' for natural, 'q' for quantized, and 's' for strong stem width and positioning mode. The first, second, and third letter correspond to grayscale, GDI ClearType, and DW ClearType rendering, respectively. The default value is qsq, which means that the 'quantized' algorithm gets used for grayscale and DW ClearType, and the 'strong' algorithm for GDI ClearType. For example, to use the 'natural' algorithm for all three rendering targets, use option -a nnn.

In the GUI, simply select the desired stem width algorithm in the three combo boxes.

`--strong-stem-width=`*string*, `-w` *string*

This option is deprecated and has been replaced with the more versatile option `--stem-width-mode` (subsection 2.3.19). The table below shows the corresponding arguments.

-w	-a
""	qqq
g	sqq
G	qsq
D	qqs
gG	ssq
gD	sqs
GD	qss
gGD	sss
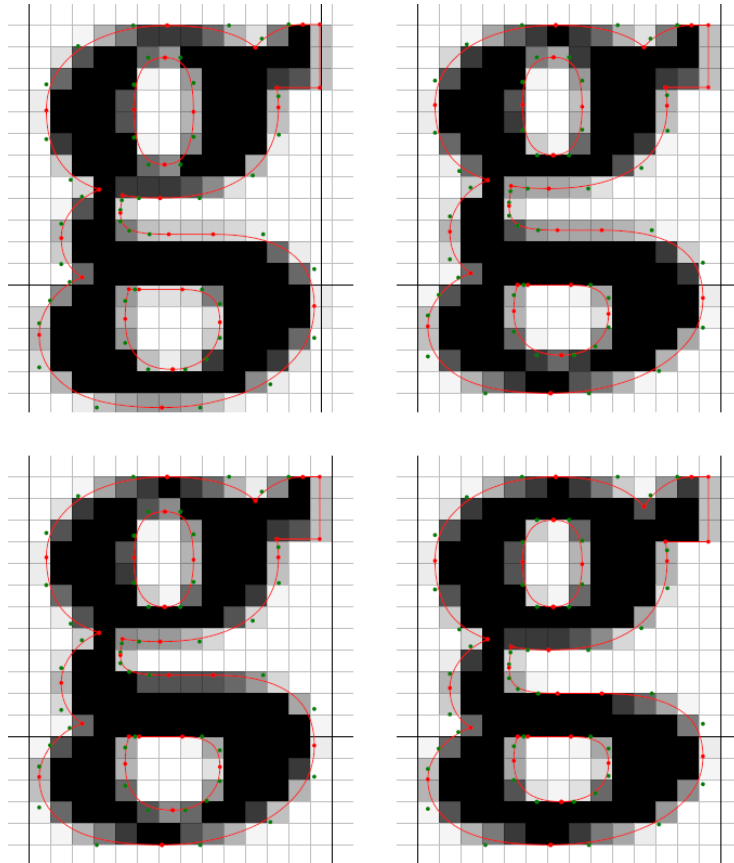
Figure 2.5: This image shows different versions of glyph 'g' of the font Merriweather-Black as displayed with the 'ftgrid' demo program of FreeType. Top left is unhinted, top right is hinted using ttfautohint natural stem width mode. Bottom left and right uses the quantized and strong stem width modes, respectively.

2.3.20 Miscellaneous

Watch input files (ttfautohintGUI only)
> If this checkbox is set, automatically regenerate the output file as soon as an input file (either the font, the control instructions file, or the reference font) gets modified.
>
> Pressing the 'Run' button starts watching. If an error occurs, watching stops and must be restarted with the 'Run' button.

--ignore-restrictions, -i
> By default, fonts that have bit 1 set in the 'fsType' field of the OS/2 table are rejected. If you have a permission of the font's legal owner to modify the font, specify this command line option.
>
> If this option is not set, ttfautohintGUI shows a dialogue to handle such fonts if necessary.

--help, -h
> On the console, print a brief documentation on standard output and exit. This doesn't work with ttfautohintGUI on MS Windows.

--version, -v
> On the console, print version information on standard output and exit. This doesn't work with ttfautohintGUI on MS Windows.

`--ttfa-info`, `-T` (not in `ttfautohintGUI`)

> Print `TTFA` table (subsection 2.3.16) of the input font on standard output if present, then exit.

`--debug` (not in `ttfautohintGUI`)

> Print *a lot* of debugging information on standard error while processing a font (you should redirect stderr to a file).
>
> To reduce the amount of debug data it is recommended to restrict the hinting process to a single PPEM value, e.g.,
>
> ```
> ttfautohint --debug -l 15 -r 15 ... > debug.txt 2>&1
> ```

3 Background and Technical Details

Real-Time Grid Fitting of Typographic Outlines is a scholarly paper that describes FreeType's auto-hinter in some detail. Regarding the described data structures it is slightly out of date, but the algorithm itself hasn't changed in general.

The next few subsections are mainly based on this article, introducing some important concepts. Note that ttfautohint only does hinting along the vertical direction (modifying y coordinates only).

3.1 Segments and Edges

A glyph consists of one or more *contours* (this is, closed curves). For example, glyph 'O' consists of two contours, while glyph 'I' has only one.

A *segment* is a series of consecutive points of a contour (including its Bézier control points) that are approximately aligned along a coordinate axis. A segment has one of three possible directions: left, right, or none (which means neither left nor right), derived from the TrueType outline directions. ttfautohint itself creates segments that contain at least two points. Using control instructions, however, it is possible to create one-point segments, which are useful for fine-tuning the hinting process.

An *edge* corresponds to a single coordinate value (allowing for a small threshold) on the main dimension that collects one or more segments, all pointing into the same direction (either left or right, all others are
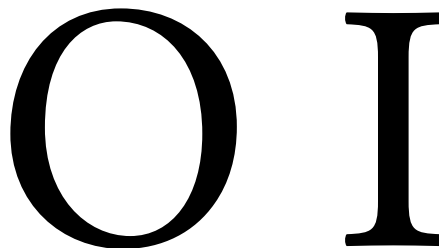
Figure 3.1: The letter 'O' has two contours, an inner and an outer one, while letter 'I' has only an outer contour.
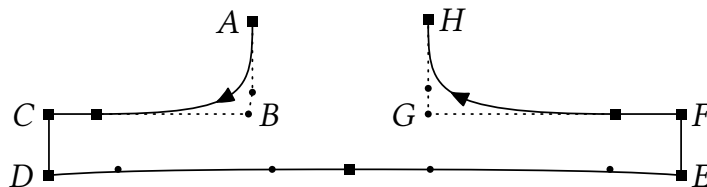
Figure 3.2: A serif. Contour and control points are represented by squares and circles, respectively. The bottom 'line' DE is approximately aligned along the horizontal axis, thus it forms a segment of 7 points. Together with the two other horizontal segments, BC and FG, they form two edges (BC+FG, DE).

ignored). While finding segments is done on the unscaled outline, finding edges is bound to the device resolution. See below (section 3.5) for an example.

In general, segments and edges pointing into different directions 'repel' each other, thus preventing alignment on the same vertical coordinate if they are near. Note that this is a simplification, but it should help understand how to manipulate and/or create segments in control instructions files.

The analysis to find segments and edges is specific to a writing system, see below (section 3.8).

3.2 Feature Analysis

The auto-hinter analyzes a font in two steps. Right now, everything described here happens for the horizontal axis only, providing vertical hinting.

- Global Analysis

  This affects the hinting of all glyphs, trying to give them a uniform appearance.

    – Compute standard horizontal stem width of the font. The value is normally taken from glyphs that resemble letter 'o'.

      If, for a given script, there is no glyph for at least one standard character in the input font, a fallback stem width gets used. See also option `--fallback-stem-width` (subsection 2.3.9).

      The stem width(s) found by the algorithm can be overridden with the `width` (section 4.3) control instruction.

    – Compute blue zones, see below (section 3.3).

  If the stem widths of single glyphs differ by a large value, or if ttfautohint fails to find proper blue zones, hinting becomes quite poor, possibly leading even to severe shape distortions.

Table 3.1: script-specific standard characters of the 'latin' writing system

Script	Standard characters
adlm	'Ɔ', U+1E90C, ADLAM CAPITAL LETTER O
	'᤮', U+1E92E, ADLAM SMALL LETTER O
arab	'ـ', U+0640, ARABIC TATWEEL
	'ل', U+0644, ARABIC LETTER LAM
	'ح', U+062D, ARABIC LETTER HAH
armn	'օ', U+0585, ARMENIAN SMALL LETTER OH
	'Օ', U+0555, ARMENIAN CAPITAL LETTER OH
avst	'𐬚', U+10B1A, AVESTAN LETTER THE
bamu	'ꛁ', U+A6C1, BAMUM LETTER YUQ
	'ꛯ', U+A6EF, BAMUM LETTER KOGHOM
beng	'০', U+09E6, BENGALI DIGIT ZERO
	'৪', U+09EA, BENGALI DIGIT FOUR
buhd	'ᝋ', U+174B, BUHID LETTER MA
	'ᝏ', U+174F, BUHID LETTER WA

Script	Standard characters
cakm	'ᄒ', U+11124, CHAKMA LETTER WAA
	'ᄉ', U+11109, CHAKMA LETTER GAA
	'ᄛ', U+1111B, CHAKMA LETTER PAA
cans	'ᑌ', U+144C, CANADIAN SYLLABICS TE
	'ᓚ', U+14DA, CANADIAN SYLLABICS LA
cari	'Ο', U+102AB, CARIAN LETTER O
	'Ϭ', U+102C9, CARIAN LETTER RR
cher	'Оᴳ', U+13A4, CHEROKEE LETTER U
	'О~', U+13C5, CHEROKEE LETTER NV
	'о~', U+AB95, CHEROKEE SMALL LETTER NV
copt	'Ⲟ', U+2C9E, COPTIC CAPITAL LETTER O
	'ⲟ', U+2C9F, COPTIC SMALL LETTER O
cprt	'𐠅', U+10805, CYPRIOT SYLLABLE JA
	'𐠣', U+10823, CYPRIOT SYLLABLE RA
cyrl	'о', U+043E, CYRILLIC SMALL LETTER O
	'О', U+041E, CYRILLIC CAPITAL LETTER O
deva	'ठ', U+0920, DEVANAGARI LETTER TTHA
	'व', U+0935, DEVANAGARI LETTER VA
	'ट', U+091F, DEVANAGARI LETTER TTA
dsrt	'𐐄', U+10404, DESERET CAPITAL LETTER LONG O
	'𐐬', U+1042C, DESERET SMALL LETTER LONG O
ethi	'ዐ', U+12D0, ETHIOPIC SYLLABLE PHARYNGEAL A
geor	'ი', U+10D8, GEORGIAN LETTER IN
	'ე', U+10D4, GEORGIAN LETTER EN
	'ა', U+10D0, GEORGIAN LETTER AN
	'Ი', U+1C98, GEORGIAN MTAVRULI CAPITAL LETTER IN
geok	'Ⴖ', U+10B6, GEORGIAN CAPITAL LETTER GHAN
	'Ⴑ', U+10B1, GEORGIAN CAPITAL LETTER SAN
	'ⴙ', U+2D19, GEORGIAN SMALL LETTER CHIN
glag	'Ⱅ', U+2C15, GLAGOLITIC CAPITAL LETTER TVRIDO
	'ⱅ', U+2C45, GLAGOLITIC SMALL LETTER TVRIDO
goth	'𐌴', U+10334, GOTHIC LETTER AIHVUS
	'𐌾', U+1033E, GOTHIC LETTER JER
	'𐍃', U+10343, GOTHIC LETTER SAUIL
grek	'ο', U+03BF, GREEK SMALL LETTER OMICRON
	'Ο', U+039F, GREEK CAPITAL LETTER OMICRON
gujr	'ટ', U+0A9F, GUJARATI LETTER TTA
	'૦', U+0AE6, GUJARATI DIGIT ZERO
guru	'ਠ', U+0A20, GURMUKHI LETTER TTHA
	'ਰ', U+0A30, GURMUKHI LETTER RA
	'੦', U+0A66, GURMUKHI DIGIT ZERO
hebr	'ם', U+05DD, HEBREW LETTER FINAL MEM
hmnp	'𞄨', U+1E128, NYIAKENG PUACHUE HMONG LETTER O
kali	'ꤍ', U+A90D, KAYAH LI LETTER NGA
	'꤀', U+A900, KAYAH LI DIGIT ZERO
knda	'೦', U+0CE6, KANNADA DIGIT ZERO

Script	Standard characters
	'ಬ', U+0CAC, KANNADA LETTER BA
khmr	'๐', U+17E0, KHMER DIGIT ZERO
lao	'o', U+0ED0, LAO DIGIT ZERO
latn	'o', U+006F, LATIN SMALL LETTER O
	'O', U+004F, LATIN CAPITAL LETTER O
	'0', U+0030, DIGIT ZERO
lisu	'O', U+A4F3, LISU LETTER
medf	'ᴑ', U+16E61, MEDEFAIDRIN SMALL LETTER S
	'ᴐ', U+16E5B, MEDEFAIDRIN CAPITAL LETTER N
	'ᴕ', U+16E6F, MEDEFAIDRIN SMALL LETTER E
mlym	'ഠ', U+0D20, MALAYALAM LETTER TTHA
	'റ', U+0D31, MALAYALAM LETTER RRA
mong	'ᠢ', U+1842, MONGOLIAN LETTER CHI
	'ᠪ', U+182A, MONGOLIAN LETTER BA
mymr	'o', U+101D, MYANMAR LETTER WA
	'c', U+1004, MYANMAR LETTER NGA
	'ဂ', U+1002, MYANMAR LETTER GA
nkoo	'ߋ', U+07CB, NKO LETTER EE
	'߀', U+07C0, NKO DIGIT ZERO
olck	'O', U+1C5B, OL CHIKI LETTER AT
orkh	'O', U+10C17, OLD TURKIC LETTER YENISEI AY
osge	'O', U+104C2, OSAGE CAPITAL LETTER O
	'o', U+104EA, OSAGE SMALL LETTER O
osma	'O', U+10486, OSMANYA LETTER DEEL
	'0', U+104A0, OSMANYA DIGIT ZERO
rohg	'0', HANIFI ROHINGYA DIGIT ZERO
saur	'O', U+A89D, SAURASHTRA LETTER TTHA
	'0', U+A8D0, SAURASHTRA DIGIT ZERO
shaw	'o', U+10474, SHAVIAN LETTER OAK
sinh	'ඨ', U+0DA7, SINHALA LETTER ALPAPRAANA TTAYANNA
sund	'O', U+1BB0, SUNDANESE DIGIT ZERO
taml	'o', U+0BE6, TAMIL DIGIT ZERO
tavt	'ꪒ', U+AA92, TAI VIET LETTER LOW DO
	'ꪫ', U+AAAB, TAI VIET LETTER HIGH VO
telu	'O', U+0C66, TELUGU DIGIT ZERO
	'೧', U+0C67, TELUGU DIGIT ONE
tfng	'O', U+2D54, TIFINAGH LETTER YAR
thai	'า', U+0E32, THAI CHARACTER SARA AA
	'ๅ', U+0E45, THAI CHARACTER LAKKHANGYAO
	'o', U+0E50, THAI DIGIT ZERO
vaii	'ɛ', U+A613, VAI SYMBOL FEENG
	'ꖜ', U+A59C, VAI SYLLABLE BHU
	'ꖴ', U+A5B4, VAI SYLLABLE KU
yezi	'𐺋', U+10E8B, YEZIDI LETTER DAL
	'𐺦', U+10EA6, YEZIDI LETTER EW
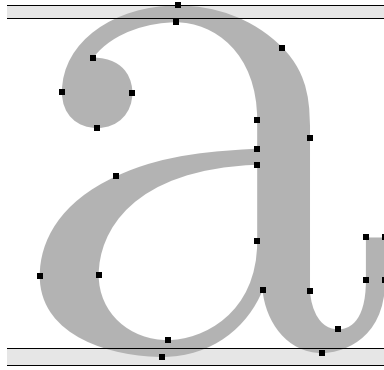
Figure 3.3: Two blue zones relevant to the glyph 'a'. Vertical point coordinates of *all* glyphs within these zones are aligned, provided the blue zone is active (this is, its vertical size is smaller than 3/4 pixels).

Table 3.2: standard characters of the 'latin' writing system, special scripts

Script	Standard characters
khms	'᧡', U+19E1, KHMER SYMBOL MUOY KOET
	'᧪', U+19EA, KHMER SYMBOL DAP KOET
latb	'ₒ', U+2092, LATIN SUBSCRIPT SMALL LETTER O
	'₀', U+2080, SUBSCRIPT ZERO
latp	'ᵒ', U+1D52, MODIFIER LETTER SMALL O
	'ᴼ', U+1D3C, MODIFIER LETTER CAPITAL O
	'⁰', U+2070, SUPERSCRIPT ZERO

- Glyph Analysis

  This is a per-glyph operation.

  - Find segments and edges.
  - Link edges to set up stems and serifs. The abovementioned paper gives more details on what exactly constitutes a stem or a serif and how the algorithm works.

3.3 Blue Zones

Outlines of certain characters are used to determine *blue zones*. This concept is the same as with Type 1 fonts: All glyph points that lie in certain small horizontal zones get aligned vertically.

The tables below show the blue zone characters of all available scripts of the latin writing system; the values are hard-coded in the source code. Since the auto-hinter takes mean values it is not necessary that all characters of a zone are present.

'Round' characters in blue zones (e.g., the top and bottom of 'O' or the bottom of 'g') are used to control overshoot handling.

Blue zones marked with an asterisk are x height blue zones, which are adjusted to be on the pixel grid (to improve rendering at small sizes) by scaling the remaining blue zones before they are adjusted to the grid. See also option `--increase-x-height` ([subsection 2.3.7](#)).

Table 3.3: `adlm` (Adlam) blue zones

ID	Blue zone	Characters
1	top of capital letters	𞤚 𞤝 𞤈 𞤲 𞤵 𞤶
2	bottom of capital letters	𞤂 𞤔
3*	top of small letters	𞤬 𞤐 𞤕 𞤸 𞤫
4	bottom of small letters	𞥊 𞤸 𞤤 𞤵 𞤶 𞤛 𞤔 𞤐

Table 3.4: `arab` (Arabic) blue zones

ID	Blue zone	Characters
1	top of letters with vertical stroke	ظ ط ك ل إ ا
2	bottom of letters	ك ظ ط ث ت
3	glyph joining	ـ

Table 3.5: `armn` (Armenian) blue zones

ID	Blue zone	Characters
1	top of capital letters	Ա Մ Ւ Փ Բ Գ Ր Օ
2	bottom of capital letters	Ւ Ռ Փ Ծ Շ Ս Օ
3	top of ascenders of small letters	եեհ ճ վ փ ֆ փ
4*	top of small letters	ш յ ւ ц q ջ ո o
5	bottom of small letters	Ճ ո ճ ш ե ծ u o
6	bottom of descenders of small letters	բ ը հ լ դ պ փ ց

Table 3.6: `avst` (Avestan) blue zones

ID	Blue zone	Characters
1	top of letters	𐬬 𐬊 𐬯 𐬱
2	bottom of letters	𐬯 𐬱

Table 3.7: `bamu` (Bamum) blue zones

ID	Blue zone	Characters
1	top of letters	ꛓ ꛍ ꛤ ꛂ ꛏ ꛟ ꛉ
2	bottom of letters	ꛕ ꛡ ꛃꛁ ꛑ ꛅ ꛖ ꛌ ꛜ

Table 3.8: `beng` (Bengali) blue zones

ID	Blue zone	Characters
1	baseline (flat glyphs only)	অ ড ত ন ব ভ ল ক
2	top of ascenders	ই ট ঠ টি ঐী টে টৌ
3*	top of baseline	ও এ ড ত ন ব ল ক
4	bottom of base characters	অ ড ত ন ব ভ ল ক

Contrary to scripts like latin, the baseline in Bengali is on the top, and we hint from top to bottom.

Table 3.9: `buhd` (Buhid) blue zones

ID	Blue zone	Characters
1	top of letters	ᝁ ᝆ
2	top of large letters	ᝊ ᝋ ᝉ
3*	top of small letters	ᝁ ᝂ ᝈ ᝎ
4	bottom of letters	ᝎ ᝂ ᝏ ᝈ ᝄ ᝅ ᝆ

Table 3.10: `cakm` (Chakma) blue zones

ID	Blue zone	Characters
1	top of letters	𑄇 𑄈 𑄚 𑄟
2	bottom of letters	𑄉 𑄃 𑄂 𑄟 𑄜
3	bottom of descenders of letters	𑄇 𑄈 𑄉 𑄊 𑄋

Table 3.11: `cans` (Canadian Syllabics) blue zones

ID	Blue zone	Characters
1	top of letters	ᗡ ᖴ ᐁ ᐅ ᐊ ᐳ ᒋ ᒧ
2	bottom of letters	ᒪ ᖵ ᒻ ᐃ ᐆ ᒐ ᗴ
3*	top of small letters	ᑐ ᑕ ᑫ ᑯ ᒥ ᐠ ᐨ ᐣ
4	bottom of small letters	ᐡ ᑯ ᑐ ᐁ ᑐ ᑕ ᑫ ᐣ
5	top of superscript letters	ᵀ ᶻ ᶜ ᶰ ᶟ ᶝ ᶳ
6	bottom of superscript letters	ᶻ ᶠ ᶫ ᐞ ᶳ ᶶ ᶹ

Table 3.12: `cari` (Carian) blue zones

ID	Blue zone	Characters
1	top of letters	𐊠 𐊒 𐊩 𐊨 𐊱 𐊛 𐋎 𐋘
2	bottom of letters	𐊠 𐊡 𐊰 𐊴 𐊒 𐊫 𐊬

25

Table 3.13: `cher` (Cherokee) blue zones

ID	Blue zone	Characters
1	top of capital letters	Ꞽ Ħ Ɛ Z Oꞌ Ꮐ Ꮪ Ꮢ
2	bottom of capital letters	Ꞽ Ħ Ɛ Z Oꞌ Ꮐ Ꮪ Ꮢ
3	top of ascenders of small letters	ħ ꮃ ꮶ Oꮲ � Ꮒ Ꮗ Ꮚ
4*	top of small letters	Ꞽ Ɛ Z Ꮜ Ꮯ Ꮤ Ꮝ ⊝
5	bottom of small letters	Ꞽ Ɛ Z Ꮜ Ꮯ Ꮤ Ꮝ ⊝
6	bottom of descenders of small letters	ꞵ Ꮇ ꭹ Ꭻ

Table 3.14: `copt` (Coptic) blue zones

ID	Blue zone	Characters
1	top of capital letters	Ⲍ Ⲏ Ⲡ Ⲅ Ⲟ Ⲑ Ⲥ Ⳅ
2	bottom of capital letters	Ⳑ Ⳡ Ⳓ Ⲏ Ⲟ Ⲑ Ⳃ Ⲱ
3*	top of small letters	ⲍ ⲏ ⲡ ⲅ ⲟ ⲑ ⲥ ⳅ
4	bottom of small letters	ⳑ ⳡ ⳓ ⲟ ⲑ ⲅ

Table 3.15: `cprt` (Cypriot) blue zones

ID	Blue zone	Characters
1	top of letters	𐠀 𐠅 𐠊 𐠎 𐠏 � 𐠣 𐠦
2	bottom of letters	𐠼 𐠭 𐠶 𐠋 𐠎 𐠑 𐠳
3	top of small letters	𐠷 𐠸 𐠹
4	bottom of small letters	𐠷 𐠸 𐠹

Table 3.16: `cyrl` (Cyrillic) blue zones

ID	Blue zone	Characters
1	top of capital letters	Б В Е П З О С Э
2	bottom of capital letters	Б В Е Ш З О С Э
3*	top of small letters	х п н ш е з о с
4	bottom of small letters	х п н ш е з о с
5	bottom of descenders of small letters	р у ф

Table 3.17: `deva` (Devanagari) blue zones

ID	Blue zone	Characters
1	top of ascenders	ई ऐ ओ औ ि ी ो ौ
2	top of baseline	क म अ आ थ ध भ श
3*	top of baseline (flat glyphs only)	क न म उ छ ट ठ ड

| 4 | bottom of base characters | क न म उ छ ट ठ ड |
| 5 | bottom of descenders | ‍ |

Contrary to scripts like latin, the baseline in Devanagari is on the top, and we hint from top to bottom. Note that some fonts have extreme variation in the height of the round elements in Zone 3; for this reason we also define Zone 1, which must be always present.

Table 3.18: `dsrt` (Deseret) blue zones

ID	Blue zone	Characters
1	top of capital letters	𐐀 𐐂 𐐘 𐐑 �
2	bottom of capital letters	𐐀 𐐆 𐐒 𐐗 𐐛
3*	top of small letters	𐐨 𐐩 𐐸 𐐺 𐑉
4	bottom of small letters	𐐨 𐐩 𐐺 𐑈 𐑉

Table 3.19: `ethi` (Ethiopian) blue zones

ID	Blue zone	Characters
1	top of letters	ሀ ሃ ዘ ፐ ሞ ከ ቀ ሐ
2	bottom of letters	ለ ሐ በ ዘ ሀ ሪ ሐ ጨ

Table 3.20: `geok` (Georgian Khutsuri) blue zones

ID	Blue zone	Characters
1	top of Asomtavruli letters	Ⴑ Ⴣ Ⴠ Ⴎ Ⴈ Ⴓ Ⴊ Ⴚ
2	bottom of Asomtavruli letters	Ⴀ Ⴂ Ⴕ Ⴈ Ⴆ Ⴑ Ⴡ Ⴄ
3*	top of Nuskhuri letters	ⴑ ⴓ ⴌ ⴄ ⴔ ⴁ ⴏ ⴈ
4	bottom of Nuskhuri letters	ⴈ ⴌ ⴈ ⴔ ⴆ ⴄ ⴀ
5	top of ascender Nuskhuri letters	ⴔ ⴑ ⴜ ⴇ ⴈ ⴚ ⴙ ⴞ
6	bottom of Nuskhuri descender letters	ⴄ ⴔ ⴔ ⴇ ⴔ ⴌ ⴓ ⴈ

Georgian Asomtavruli and Nuskhuri form the old ecclesiastical script, Khutsuri. Note that fonts show a great variation in height and depth of ascender and descender letter forms.

Table 3.21: `geor` (Georgian Mkhedruli) blue zones

ID	Blue zone	Characters
1*	top of Mkhedruli letters	გ დ ე ვ თ ი ო ღ
2	bottom of Mkhedruli letters	ა ბ ზ ს შ ძ ხ ჭ
3	top of ascender Mkhedruli letters	ს ხ კ ზ მ შ წ ჩ
4	bottom of descender Mkhedruli letters	ე ვ ჟ ტ უ ფ ქ ყ
5	top of ascender Mtavruli letters	Ⴈ � Ⴈ Ⴑ Ⴁ Ⴈ Ⴏ Ⴡ
6	bottom of descender Mtavruli letters	Ⴂ Ⴓ Ⴖ Ⴙ Ⴧ Ⴔ Ⴭ Ⴓ

Table 3.22: `glag` (Glagolitic) blue zones

ID	Blue zone	Characters
1	top of capital letters	Ⰳ Ⱁ Ⱇ Ⰿ Ⱃ Ⱑ Ⰲ Ⱚ
2	bottom of capital letters	Ⰳ Ⰿ Ⱃ Ⱇ Ⱎ Ⰴ Ⱑ Ⱁ
3*	top of small letters	ⰳ ⱁ ⱇ ⰲ ⱃ ⱑ ⰲ ⱚ
4	bottom of small letters	ⰳ Ⰿ ⱃ ⱇ ⱎ ⰴ ⱑ ⱁ

Table 3.23: `goth` (Gothic) blue zones

ID	Blue zone	Characters
1	top of letters	𐌲 𐌶 𐌸 𐍄 𐌴 𐍃 𐍉 𐌶
2	bottom of letters	𐌶 𐌴 𐍃 𐍉

Table 3.24: `grek` (Greek) blue zones

ID	Blue zone	Characters
1	top of capital letters	Γ Β Ε Ζ Θ Ο Ω
2	bottom of capital letters	Β Δ Ζ Ξ Θ Ο
3	top of 'small beta' like letters	β θ δ ζ λ ξ
4*	top of small letters	α ε ι ο π σ τ ω
5	bottom of small letters	α ε ι ο π σ τ ω
6	bottom of descenders of small letters	β γ η μ ρ φ χ ψ

Table 3.25: `gujr` (Gujarati) blue zones

ID	Blue zone	Characters
1*	top of letters	ત ન ઋ ળ છ ટ ર ૦
2	bottom of letters	ખ ગ ઘ ચ ઇ ઈ ઙ જ
3	top of ascenders	ઈ ઉ ○ि ○ी ली શ્રિ જિ સी
4	bottom of descenders	○ु ○ृ ○ॄ ખુ છૃ છૄ
5	top of Gujarati digits	૦ ૧ ૨ ૩ ૭

Table 3.26: `guru` (Gurmukhi) blue zones

ID	Blue zone	Characters
1	top of ascenders	ਇ ਈ ਉ ੲ ੳ ੳ ○ਿ ○ੀ
2	top of baseline	ਕ ਗ ਙ ਚ ਜ ੜ ਧ ਸ
3*	top of baseline (flat glyphs only)	ਕ ਗ ਙ ਚ ਜ ੜ ਧ ਸ
4	bottom of characters	ਅ ੲ ੳ ਗ ਜ ਠ ਰ ਸ
5	top of Gurmukhi digits	੦ ੧ ੨ ੩ ੯

Table 3.27: `hebr` (Hebrew) blue zones

ID	Blue zone	Characters
1	top of letters	ס מ כ ך ה ד ב
2	bottom of letters	צ ס מ כ ט ב
3	bottom of descenders of letters	ק ך ן ף ץ

Table 3.28: `hmnp` (Nyiakeng Puachue Hmong) blue zones

ID	Blue zone	Characters
1	top of letters	𖬀 𖬂 𖬄 𖬆 𖬈
2	bottom of letters	𖬀 𖬂 𖬄 𖬆 𖬈

Table 3.29: `kali` (Kayah Li) blue zones

ID	Blue zone	Characters
1*	top of letters	ꤊ ꤛ ꤢ ꤕ ꤏ
2	bottom of letters	ꤚ ꤊ ꤢ ꤕ ꤜ
3	top of ascending letters	ꤕ ꤘ
4	bottom of descending letters	ꤢ ꤒ ꤖ
5	bottom of large descending letters	ꤢ ꤒ ꤖ

Table 3.30: `khmr` (Khmer) blue zones

ID	Blue zone	Characters
1*	top of letters	ខ ទ ន � ◌ា
2	top of subscript cluster components	្ក ្ខ ្ច ្ឆ
3	bottom of letters	ខ ឃ ច ប ម ឃ ច
4	bottom of descenders	្រ ្រ៉ ្ឆ ្រ៉
5	bottom of large descenders	្រៃ ្ងួ ្កៀ ្ថៀ ្ញៀ ្លៀ

Table 3.31: `khms` (Khmer Symbols) blue zones

ID	Blue zone	Characters
1*	top of symbols for waxing	᧠ ᧡ ᧢ ᧣
2	bottom of symbols for waning	᧪ ᧫

Khmer symbols are used for lunar dates.

Table 3.32: `knda` (Kannada) blue zones

ID	Blue zone	Characters
1	top of letters	ಇ ಊ ಐ ಣ ಸಾ ನಾ ದಾ ಠಾ
2	bottom of letters	ಅ ಉ ఎ ల 0 ౨ ౬ ౭

Table 3.33: `lao` (Lao) blue zones

ID	Blue zone	Characters
1*	top of letters	໅ ຓ ຖ ມ ລ ວ ຣ
2	bottom of letters	໅ ອ ບ ຍ ຣ ຣ ວ ຢ
3	top of ascenders	ປ ຢ ຟ ຝ
4	top of large ascenders	ໃ ໄ ໄ
5	bottom of descenders	ຬ � ງ ຽ ໆ ໆ

Table 3.34: `latb` (Latin Subscripts) blue zones

ID	Blue zone	Characters
1	top of capital characters	0 3 5 7 8
2	bottom of capital characters	0 1 2 3 8
3	top of 'small f' like characters	i j h k l
4*	top of small characters	a e o x n s v u r
5	bottom of small characters	a e o x n s v u r
6	bottom of descenders of small characters	β γ ρ φ p

Subscript latin characters are similar to normal latin characters.

Table 3.35: `latn` (Latin) blue zones

ID	Blue zone	Characters
1	top of capital letters	T H E Z O C Q S
2	bottom of capital letters	H E Z L O C U S
3	top of 'small f' like letters	f i j k d b h
4*	top of small letters	u v x z o e s c
5	bottom of small letters	n r x z o e s c
6	bottom of descenders of small letters	p q g j y

Table 3.36: `latp` (Latin Superscripts) blue zones

ID	Blue zone	Characters
1	top of capital characters	0 3 5 7 T H E O
2	bottom of capital characters	0 1 2 3 E L O U

ID	Blue zone	Characters
3	top of 'small f' like characters	b d k h j f i
4*	top of small characters	e o r s x c z
5	bottom of small characters	e o r s x c z
6	bottom of descenders of small characters	p y g

Superscript latin characters are similar to normal latin characters.

Table 3.37: `lisu` (Lisu) blue zones

ID	Blue zone	Characters
1	top of letters	ㄴ ㅐ ㅌ ㄱ ㄷ ㄲ O
2	bottom of letters	ㅗ ㄹ ㅓ ㄴ ㅇ S O U

Table 3.38: `medf` (Medefaidrin) blue zones

ID	Blue zone	Characters
1	top of capital letters	℘ ⅃ ℣ ℥ ℇ ℗ ⅃
2	bottom of capital letters	℘ ⅃ ℣ ℥ ℇ ℗ ℈ ℨ
3	top of ascenders of small letters	℔ ℀ ℓ ℊ ℱ ℨ
4*	top of small letters	℧ ℧ ℘ ℙ ℨ ⅃
5	bottom of small letters	℧ ℧ ℘ ℨ ♡ ℵ
6	bottom of descenders of small letters	ℨ ℈ ℘
7	top of Medefaidrin digits	θ ℥ ↑ ▽ ℋ

Table 3.39: `mlym` (Malayalam) blue zones

ID	Blue zone	Characters
1	top of letters	൭ S O ഠ ച പ ഖ ള
2	bottom of letters	S O ൡ ൫ ൢ ച ഇ ല

Table 3.40: `mong` (Mongolian) blue zones

ID	Blue zone	Characters
1	top of base stem	ᠪ ᠷ ᠠ ᠼ ᠥᠥ ᠷ᠊ ᠥᠷ
2	bottom of base stem	᠌

Table 3.41: mymr (Myanmar) blue zones

ID	Blue zone	Characters
1*	top of letters	၁ ဂ င ၃ ၀ ၁ ၊ ။
2	bottom of letters	င ဎ ၃ ၀ ၆ ၀ ၊ ။
3	top of ascenders of characters	သြ ြ ရှ ေါ် ၆ ၁ ံ
3	bottom of descenders of letters	၃ � ၃ သြ ၆ ၂ ၅ ၉

Table 3.42: nkoo (N'Ko) blue zones

ID	Blue zone	Characters
1	top of letters	ߕ ߔ ߖ ߗ ߦ ߥ ߒ ߛ
2	bottom of letters	߀ ߡ ߝ ߒ ߛ
3*	top of small letters	ߚ ߘ ߏ
4	bottom of small letters	ߎ ߚ ߘ ߏ

Table 3.43: olck (Ol Chiki) blue zones

ID	Blue zone	Characters
1	top of letters	᱐ ᱜ ᱝ ᱬ ᱫ ᱶ
2	bottom of letters	᱐ ᱜ ᱝ ᱬ ᱫ ᱶ

Table 3.44: orkh (Old Turkic) blue zones

ID	Blue zone	Characters
1	top of letters	𐰞 𐰏 𐰆
2	bottom of letters	𐰁 𐰞 𐰔 𐰆

Table 3.45: osge (Osage) blue zones

ID	Blue zone	Characters
1	top of capital letters	𐒷 𐒼 𐒿 𐓀 𐓂 𐓏 𐓐
2	bottom of capital letters	𐓄 𐒼 𐓂 𐓋 𐓎 𐓆
3	bottom of descenders of capital letters	𐒹 𐒻 𐒽
4*	top of small letters	𐓷 𐓶 𐓭 𐓮 𐓴 𐓪 𐓨
5	bottom of small letters	𐓚 𐓘 𐓭 𐓷 𐓞 𐓪 𐓶
6	top of ascenders of small letters	𐓸 𐓹 𐓻 𐓽 𐓧
7	bottom of descenders of small letters	𐓹 𐓺 𐓻

32

Table 3.46: `osma` (Osmanya) blue zones

ID	Blue zone	Characters
1	top of letters	Ο ૧ Ⴖ ૩ ૬ Ⴍ Ⴙ
2	bottom of letters	Ծ ૫ Ⴍ ૩ ૪ ૩ Ⴑ

Table 3.47: `rohg` (Hanifi Rohingya) blue zones

ID	Blue zone	Characters
1	top of letters with vertical stroke	ꤺ ꤻ ꤹ ꤳ ꤶ
2	bottom of letters	ꤺ ꤳ ꤶ ꤲ ꤴ
3	glyph joining	-

Table 3.48: `saur` (Saurashtra) blue zones

ID	Blue zone	Characters
1	top of letters	ꢳ ꢴ ꢼ ꢪ ꢡ ꢐ Ο ꢯ
2	bottom of letters	ꢪ ꢼ ꢝ ꢳ ꢯ

Table 3.49: `shaw` (Shavian) blue zones

ID	Blue zone	Characters
1	top of letters	𐑕 𐑝
2	bottom of letters	𐑖 𐑘 𐑙 𐑚 𐑛
3	bottom of descenders of letters	𐑑 𐑜
4*	top of small letters	𐑦 𐑧 𐑐 𐑩 𐑪 𐑫
5	bottom of small letters	𐑐 𐑫 𐑭

Table 3.50: `sinh` (Sinhala) blue zones

ID	Blue zone	Characters
1	top of letters	ඉ ක ස ද ප ය ල ඏ
2	bottom of letters	එ ඔ ස ජ ට ර ධ ර
3	bottom of descenders of letters	ද ද උ ල තු තු ඔ ද

Table 3.51: `sund` (Sundanese) blue zones

ID	Blue zone	Characters
1	top of letters	ᮙ ᮊ ᮒ ᮕ Ο ᮓ
2	bottom of letters	ᮜ ᮚ ᮒ ᮕ Ο ᮓ ᮔ ᮮ
3	bottom of descenders of letters	ᮤ ᮨ

Table 3.52: `taml` (Tamil) blue zones

ID	Blue zone	Characters
1	top of letters	உ ஓ ஓ ற ஈ க ங ச
2	bottom of letters	க ச ல �揃 உ ங ட ப

Table 3.53: `tavt` (Tai Viet) blue zones

ID	Blue zone	Characters
1	top of letters	ꪊ ꪑ ꪔ ꪣ ꪦ
2	bottom of letters	ꪉ ꪎ ꪠ

Table 3.54: `telu` (Telugu) blue zones

ID	Blue zone	Characters
1	top of letters	ఇ ఇ ఙ ఞ ఱ ౯
2	bottom of letters	అ క చ ౮ ౽ ౨ ౬

Table 3.55: `tfng` (Tifinagh) blue zones

ID	Blue zone	Characters
1	top of letters	ⵔ ⵔ ⵛ ⵞ ⵝ ⵀ ⵗ ⵛ
2	bottom of letters	ⵔ ⵔ ⵛ ⵞ ⵝ ⵀ ⵗ ⵛ

Table 3.56: `thai` (Thai) blue zones

ID	Blue zone	Characters
1*	top of letters	บ เ แ อ ก า
2	bottom of letters	บ ป ษ �ๆ อ ฮ
3	top of ascenders of letters	ป ฝ ฟ
4	top of large ascenders of letters	โ ใ ไ
5	bottom of descenders of letters	ฎ ฏ ฤ ฦ
6	bottom of large descenders of letters	ญ ฐ
7	top of Thai digits	๐ ๑ ๓

Table 3.57: `vaii` (Vai) blue zones

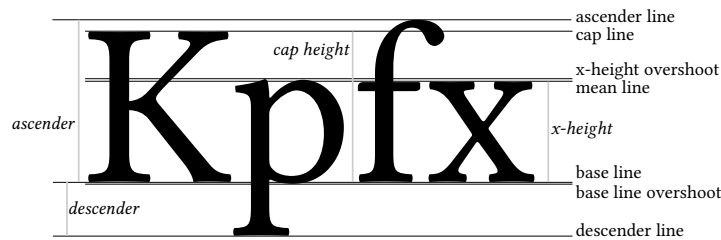| ID	Blue zone	Characters
1	top of letters	ꔀ ꘓ ꖴ ꕙ ꕯ ꔷ ꕪ ꖡ
2	bottom of letters	ꔀ ꘓ ꖴ ꖖ ꕯ ꔷ ꕪ ꖝ

Figure 3.4: This image shows the relevant glyph terms for vertical blue zone positions.

Table 3.58: `yezi` (Yezidi) blue zones

ID	Blue zone	Characters
1*	top of letters	ᐯ ᒎ ᐣ ⊥ ᐃ ᐁ ▢
2	bottom of letters	ᐯ ⊥ ᐃ ▢ ⊓ ᙭
3	top of ascenders of letters	ᛁ ᒐ ᒣ
4	bottom of descenders of letters	ᒎ ᐣ ᒐ ∥ ┰ ᐃ ᕍ

3.4 Grid Fitting

Aligning outlines along the grid lines is called *grid fitting*. It doesn't necessarily mean that the outlines are positioned *exactly* on the grid, however, especially if you want a smooth appearance at different sizes. This is the central routine of the auto-hinter; its actions are highly dependent on the used writing system. Currently, only one writing system is available (latin), providing support for scripts like Latin or Greek.

- Align edges linked to blue zones.

- Fit edges to the pixel grid.

- Align serif edges.

- Handle remaining 'strong' points. Such points are not part of an edge but are still important for defining the shape. This roughly corresponds to the IP TrueType instruction.

- Everything else (the 'weak' points) is handled with an 'IUP' instruction.

The following images illustrate the hinting process, using glyph 'a' from the freely available font 'Ubuntu Book'. The manual hints were added by Dalton Maag Ltd, the used application to create the hinting debug snapshots was FontForge.

3.5 Hint Sets

In ttfautohint terminology, a *hint set* is the *optimal* configuration for a given PPEM (pixel per EM) value.

In the range given by the `--hinting-range-min` and `--hinting-range-max` options, ttfautohint creates hint sets for every PPEM value. For each glyph, ttfautohint automatically determines whether a new
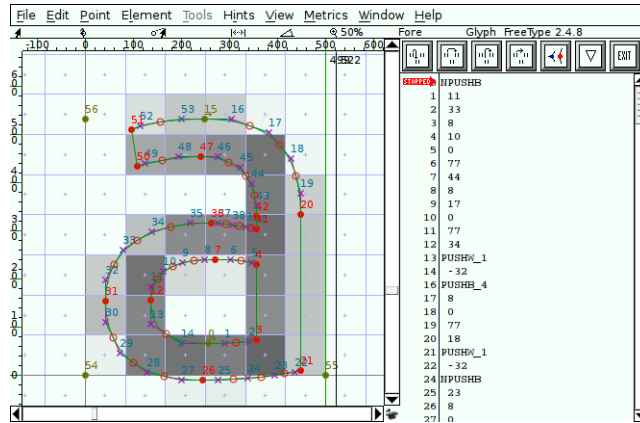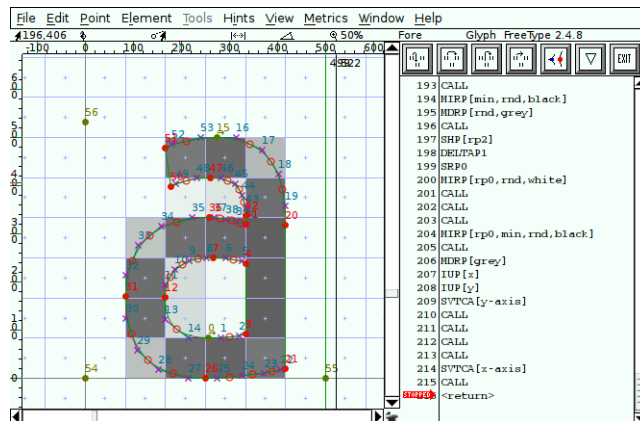
Figure 3.5: Before hinting.

Figure 3.6: After hinting, using manual hints.

set should be emitted for a PPEM value if it finds that it differs from a previous one. For some glyphs it is possible that one set covers, say, the range 8px-1000px, while other glyphs need 10 or more such sets.

In the PPEM range below `--hinting-range-min`, ttfautohint always uses just one set, in the PPEM range between `--hinting-range-max` and `--hinting-limit`, it also uses just one set.

One of the hinting configuration parameters is the decision which segments form an edge. For example, let us assume that two segments get aligned on a single horizontal edge at 11px, while two edges are used at 12px. This change makes ttfautohint emit a new hint set to accomodate this situation. The next images illustrate this, using a Cyrillic letter (glyph 'afii10108') from the 'Ubuntu book' font, processed with ttfautohint.

Obviously, the more hint sets get emitted, the larger the bytecode ttfautohint adds to the output font. To find a good value n for `--hinting-range-max`, some experimentation is necessary since n depends on the glyph shapes in the input font. If the value is too low, the hint set created for the PPEM value n (this hint set gets used for all larger PPEM values) might distort the outlines too much in the PPEM range given by n and the value set by `--hinting-limit` (at which hinting gets switched off). If the value is too high, the font size increases due to more hint sets without any noticeable hinting effects.

Similar arguments hold for `--hinting-range-min` except that there is no lower limit at which hinting is switched off.
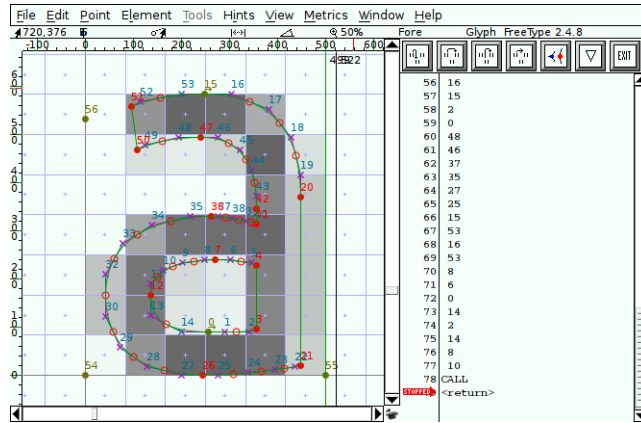
36

Figure 3.7: After hinting, using ttfautohint. Note that the hinting process doesn't change horizontal positions.
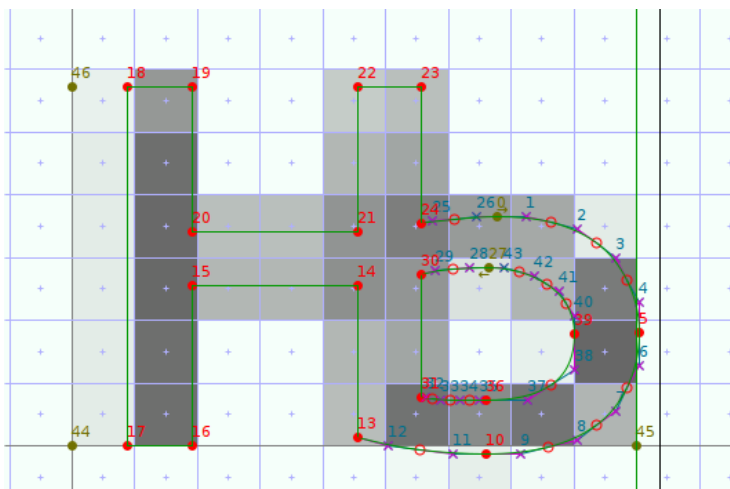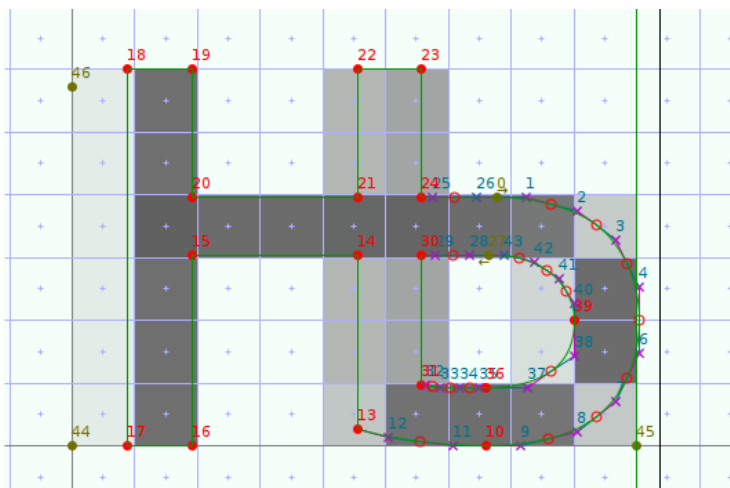

Figure 3.8: Before hinting, size 11px.

Figure 3.9: After hinting, size 11px. Segments 43-27-28 and 14-15 are aligned on a single edge, as are segments 26-0-1 and 20-21.
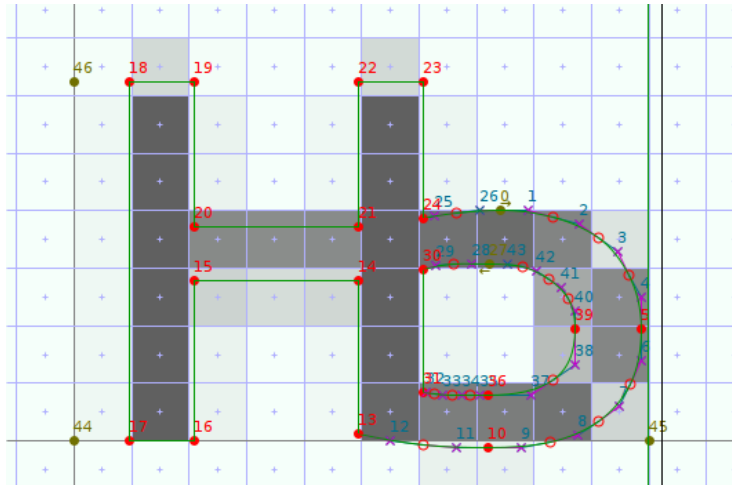
Figure 3.10: Before hinting, size 12px.
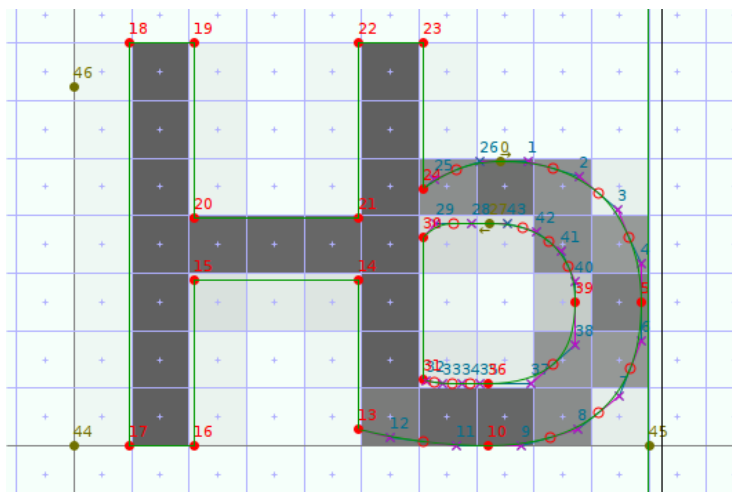
Figure 3.11: After hinting, size 12px. The segments are not aligned. While segments 43-27-28 and 20-21 now have almost the same horizontal position, they don't form an edge because the outlines passing through the segments point into different directions.

An example. Let's assume that we have a hinting range 10 <= ppem <= 100, and the hinting limit is set to 250. For a given glyph, ttfautohint finds out that four hint sets must be computed to exactly cover this hinting range: 10-15, 16-40, 41-80, and 81-100. For PPEM values below 10ppem, the hint set covering 10-15ppem is used, for PPEM values larger than 100 the hint set covering 81-100ppem is used. For PPEM values larger than 250, no hinting gets applied.

3.6 Composite Glyphs

The ttfautohint library (and programs) supports two solutions for handling composite glyphs, to be controlled with option --composites (subsection 2.3.12). This section contains some general information, then covers the case where the option is off, while the next section describes how ttfautohint behaves if this option is activated.

Figure 3.12: Here, the subscript glyphs are composites each having a single element that is shifted down. If option `--composites` is not used, subglyphs are hinted before they are glued together (possibly applying scaling and shifting). Because the `ROUND_XY_TO_GRID` flag isn't set, the vertical translation doesn't align the subglyph to the pixel grid, causing severe distortions.

Figure 3.13: The same as before, but with `ROUND_XY_TO_GRID` set. Now the subscript glyphs look identical to the superscripts.

Figure 3.14: For comparison purposes, here the result *with* option `--composites` (and no `ROUND_XY_TO_GRID`). The composite glyphs as a whole get hinted; consequently, the subscript glyphs get separate blue zones. At the displayed size of 16ppem the vertical positions of the subscript blue zones are rounded differently if compared to the superscript zones, thus the smaller glyph height.

Regardless of the `--composites` option, ttfautohint performs a scan over all composite glyphs to assure that components of a composite glyph inherit its style, as described later (section 3.10). However, components that are shifted vertically will be skipped. For example, if the glyph 'Agrave' uses a shifted 'grave' accent glyph, the accent is ignored. On the other hand, if there is a glyph 'agrave' that uses the same 'grave' glyph vertically unshifted, 'grave' does inherit the style.

If `--composites` is off, components are hinted separately, then put together. Separate hinting implies that the current style's blue zones are applied to all subglyphs in its original, unshifted positions. In case you want to shift components vertically, it is *mandatory* to set bit 2 (value 4), `ROUND_XY_TO_GRID`, in the flag variable of the composite glyph description to get visually pleasing results, as the images below demonstrate.

3.7 The '.ttfautohint' Glyph

If option `--composites` (subsection 2.3.12) is used, ttfautohint doesn't hint subglyphs of composite glyphs separately. Instead, it hints the whole glyph, this is, composites get recursively expanded internally so that they form simple glyphs, then hints are applied – this is the normal working mode of FreeType's auto-hinter.

One problem, however, must be solved: Hinting for subglyphs (which usually are used as normal glyphs also) must be deactivated so that nothing but the final bytecode of the composite gets executed.

The trick used by ttfautohint is to prepend a composite element called '.ttfautohint', a dummy glyph with a single point, and which has a single job: Its bytecode increases a variable (to be more precise, it is a

CVT register called `cvtl_is_subglyph` in the source code), indicating that we are within a composite glyph. The final bytecode of the composite glyph eventually decrements this variable again.

As an example, let's consider composite glyph 'Agrave' ('À'), which has the subglyph 'A' as the base and 'grave' as its accent. After processing with ttfautohint it consists of three components: '.ttfautohint', 'A', and 'grave' (in this order).

Bytecode of	Action
.ttfautohint	increase `cvtl_is_subglyph` (now: 1)
A	do nothing because `cvtl_is_subglyph` > 0
grave	do nothing because `cvtl_is_subglyph` > 0
Agrave	decrease `cvtl_is_subglyph` (now: 0)
	apply hints because `cvtl_is_subglyph` == 0

Some technical details (which you might skip): All glyph point indices get adjusted since each '.ttfautohint' subglyph shifts all following indices by one. This must be done for both the bytecode and one subformat of OpenType's `GPOS` anchor tables.

While this approach works fine on all tested platforms, there is one single drawback: Direct rendering of the '.ttfautohint' subglyph (this is, rendering as a stand-alone glyph) disables proper hinting of all glyphs in the font! Under normal circumstances this never happens because '.ttfautohint' doesn't have an entry in the font's `cmap` table. (However, some test and demo programs like FreeType's `ftview` application or other glyph viewers that are able to bypass the `cmap` table might be affected.)

3.8 Writing Systems

In FreeType terminology, a writing system is a set of functions that provides auto-hinting for certain scripts. Right now, only two writing systems from FreeType's auto-hinter are available in ttfautohint: 'dummy' and 'latin'. The former handles the 'no-script' case; details to 'latin' follow in the next section.

3.9 Scripts

ttfautohint needs to know which script should be used to hint a specific glyph. To do so, it checks a glyph's Unicode character code whether it belongs to a given script.

See 'Character Ranges (chapter 8)' for a complete list of all handled scripts and its ranges. This list is auto-generated from a source code file, covering the 'latin' writing system. It also covers some non-latin scripts (in the Unicode sense) that have similar typographical properties.

In ttfautohint, scripts are identified by four-character tags (if there are less characters, spaces are appended). The value `none` indicates 'no script'.

Each script is represented by two tables to handle 'base' and 'non-base' characters. For ttfautohint, a non-base character is something that should not be affected by blue zones, regardless of whether this is a spacing or no-spacing glyph. In other words, non-base characters are hinted using a script's default stem width without applying blue zones.

Right now, there are two pseudo-scripts that are used as fallbacks: `latb` and `latp`, used for latin subscript and superscript characters, respectively. Its main usage is support of phonetic alphabets like the IPA, which intermix those characters with normal characters sitting on the baseline, and which are not specially handled in corresponding OpenType features like `sups`.

If a glyph's character code is not covered by a script range, it is handled by a *fallback script*. By default, the fallback script is `none`, which indicates handling by the 'latin' writing system without applying script-specific blue zones (but aligning stems to the grid if possible). The fallback script can be changed; see option `--fallback-script` (subsection 2.3.5).

The user can also select whether uncovered glyphs are either hinted (which is the default) or scaled only with the fallback script's scaling parameters. This can be controlled with option `--fallback-scaling` (subsection 2.3.5). Note that fallback scaling only makes sense if the fallback script has x height blue zones, e.g., `cyrl` or `latn`.

As a special case, specifying `none` as a fallback script and switching on fallback scaling ('`-f none -S`'), no hinting is applied at all to uncovered glyphs – using `none` always implies a scaling factor of 1.

3.10 OpenType Features

(Please read the OpenType specification for details on *features*, `GSUB`, and `GPOS` tables, and how they relate to scripts.)

For modern OpenType fonts, character ranges are not sufficient to handle scripts.

- Due to glyph substitution in the font (as specified in a font's `GSUB` table), which handles ligatures and similar typographic features, there is no longer a one-to-one mapping from an input Unicode character to a glyph index. Some ligatures, like 'fi', actually do have Unicode values for historical reasons, but most of them don't. While it is possible to map ligature glyphs into Unicode's Private Use Area (PUA), code values from this area are arbitrary by definition and thus unusable for ttfautohint.

- Some features like `sups` (for handling superscript) completely change the appearance and even vertical position of the affected glyphs. Obviously, the blue zones for 'normal' glyphs no longer fit, thus the auto-hinter puts them into a separate group (called *style* in FreeType speak), having its own set of blue zones.

Table 3.60: OpenType features handled specially by ttfautohint

Feature tag	Description
c2cp	petite capitals from capitals
c2sc	small capitals from capitals
ordn	ordinals
pcap	petite capitals
sinf	scientific inferiors
smcp	small capitals
subs	subscript
sups	superscript
titl	titling

There are two conditions to get a valid style for a feature in a given script.

1. One of the script's standard characters must be available in the feature.

2. The feature must provide characters to form at least one blue zone; see above (section 3.3).

An additional complication is that features from the above table might use data not only from the GSUB but also from the GPOS table, containing information for glyph positioning. For example, the sups feature for superscripts might use the same glyphs as the subs feature for subscripts, simply moved up. ttfautohint skips such vertically shifted glyphs (except for accessing standard characters) because glyph positioning happens after hinting. Continuing our example, the sups feature wouldn't form a style, contrary to subs, which holds the unshifted glyphs.

The remaining OpenType features of a script are not handled specially; the affected glyphs are simply hinted together with the 'normal' glyphs of the script.

Note that a font might still contain some features not covered yet: OpenType has the concept of a *default script*; its data gets used for all scripts that aren't explicitly handled in a font. By default, ttfautohint unifies all affected glyphs from default script features with the latn script. This can be changed with option --default-script (subsection 2.3.4), if necessary.

ttfautohint uses the HarfBuzz library for handling OpenType features.

3.11 SFNT Tables

ttfautohint touches almost all SFNT tables within a TrueType or OpenType font. Note that only Open-Type fonts with TrueType outlines are supported. OpenType fonts with a CFF or CFF2 table (this is, with PostScript outlines) won't work.

- glyf: All hints in the table are replaced with new ones. If option --composites (subsection 2.3.12) is used, one glyph gets added (namely the '.ttfautohint' glyph) and all composites get an additional component.

- cvt, prep, and fpgm: These tables get replaced with data necessary for the new hinting bytecode.

- gasp: Set up to always use grayscale rendering, for all sizes, with grid-fitting for standard hinting, and symmetric grid-fitting and symmetric smoothing for horizontal subpixel hinting (ClearType).

- DSIG: If it exists, it gets replaced with a dummy version. ttfautohint can't digitally sign a font; you have to do that afterwards.

- name: The 'version' entries are modified to add information about the parameters that have been used for calling ttfautohint. This can be controlled with the --no-info (subsection 2.3.15) option. Family name entries might also be affected by option --family-suffix (subsection 2.3.17).

- GPOS, hmtx, loca, head, maxp, post: Updated to fit the additional '.ttfautohint' glyph, the additional subglyphs in composites, and the new hinting bytecode.

- LTSH, hdmx: Since ttfautohint doesn't do any horizontal hinting, those tables are superfluous and thus removed.

- VDMX: Removed, since it depends on the original bytecode, which ttfautohint removes. A font editor might recompute the necessary data later on.

3.12 Problems

3.12.1 Interaction With FreeType

Some versions of FreeType have an experimental extension for handling subpixel hinting; it is off by default and can be activated by setting the macro `TT_CONFIG_OPTION_SUBPIXEL_HINTING` to value 2 or 3 at compile time. This code has been contributed mainly by Infinality, being a subset of his original patch. Many GNU/Linux distributions activate this code, or provide packages to activate it. [Note that starting with FreeType version 2.7 the Infinality extension is still available but superseded by a less aggressive subpixel hinting mode, to which the explanations in this section don't apply.]

This extension changes the behaviour of many bytecode instructions to get better rendering results. However, not all changes are global; some of them are specific to certain fonts. For example, it contains font-specific improvements for the 'DejaVu Sans' font family. The list of affected fonts is hard-coded; it can be found in FreeType's source code file `ttsubpix.c`.

If you are going to process such specially-handled fonts with ttfautohint, serious rendering problems might show up. Since ttfautohint (intentionally) doesn't change the font name in the `name` table, the Infinality extension has no chance to recognize that the hints are different. All such problems vanish if the font gets renamed in its `name` table (the name of the font file itself doesn't matter), for example, by using option `--family-suffix` (subsection 2.3.17).

3.12.2 Incorrect Unicode Character Map

Fonts with an incorrect Unicode `cmap` table will not be properly hinted by ttfautohint. Especially older fonts do cheat; for example, there exist Hebrew fonts that map its glyphs to character codes 'A', 'B', etc., to make them work with non-localized versions of Windows 98, say.

Since ttfautohint needs to find both standard and blue zone characters, it relies on correct Unicode values. If you want to handle such fonts, please fix their `cmap` tables accordingly before running ttfautohint.

3.12.3 Irregular Glyph Heights

The central concept of ttfautohint's hinting algorithm as discussed above (section 3.1) is to identify horizontal segments at extremum positions, especially for blue zones. If such a segment is missing, it cannot be associated with a blue zone, possibly leading to irregular heights for the particular glyph.

Normally, a segment has a horizontal length of at least 20 font units (assuming 2048 units per EM)[1]. Using a Control Instructions File (subsection 2.3.1), however, it is possible to define additional segments at arbitrary points that help overcome this restriction, making it possible to fix (most of) such problems.

3.12.4 Diagonals

ttfautohint doesn't handle diagonal lines specially. For thin outlines, this might lead to strokes that look too thick at smaller sizes. A font designer might compensate this to a certain amount by slightly reducing

---

[1]To be more precise, the sum of the height and length of a segment must be at least 20 font units, and the height multiplied by 14 must not exceed the length. Thus (19,1) is also a valid minimum (length,height) pair, while (18,2) isn't. The value 20 is heuristic and hard-coded, as is the value 14 (corresponding to a slope of approx. 4.1°).

the stroke width of diagonal lines. However, in many cases the sub-optimal appearance of a stroke with borders that don't exactly fit the pixel grid is not the outline itself but an incorrect gamma value of the monitor: People tend to not properly adjust it, and the default values of most operating systems are too low, causing too much darkening of such strokes. It is thus of vital importance to compare ttfautohint's results with similar fonts to exclude any systematic effect not related to the outlines themselves.

3.13 Extending ttfautohint with new scripts

Right now, adding new scripts to ttfautohint only works on the source code level, this is, you have to patch the C source code.

The process itself isn't very complicated; it is demonstrated best by example. The following commits in ttfautohint add Ethiopian and Armenian, respectively.

https://repo.or.cz/ttfautohint.git/commitdiff/d14c7c07
https://repo.or.cz/ttfautohint.git/commitdiff/b5022cd9

It shows that you have to do the following steps.

- Add blue zone character data to the file `lib/tablue.dat`.

- Add the proper Unicode ranges to `lib/taranges.c`, following the structure of similar entries.

- Similarly, the files `lib/tastyles.h` and `lib/ttfautohint-script.h` must be updated. The latter holds the information on the used default character or characters; it also references the corresponding script tag `HB_SCRIPT_XXX` as used by the HarfBuzz library.

If there are any questions, please contact the FreeType mailing list for help. Note that the script data in ttfautohint are hold in sync with FreeType's auto-hinter.

4 Control Instructions

An entry in a control instructions file has various syntax forms, which are discussed here. Brackets indicate optional elements.

4.1 Common Syntax Elements

font-idx gives the index of the font in a TrueType Collection, starting with value 0. If missing, it is set to zero. For normal TrueType fonts, only value zero is valid. A font index can be specified in decimal, octal, or hexadecimal format, the latter two indicated by the prefixes 0 and 0x, respectively.

glyph-id is either a glyph's name as listed in the font's `post` SFNT table or a glyph index. A glyph name consists of characters from the set 'A-Za-z0-9._' only and does not start with a digit or period, with the exceptions of the names '.notdef' and '.null'. A glyph index starts with value 0 can be specified in decimal, octal, or hexadecimal format, the latter two indicated by the prefixes 0 and 0x, respectively. Glyph names are internally converted to glyph indices.

points are number ranges, see 'x Height Snapping Exceptions ([subsection 2.3.8](#))' for the syntax.

Similar to the Bourne shell (`sh` or `bash`), a comment starts with character '#'; the rest of the line is ignored. An empty line is ignored also. Both the newline character and ';' can be used as a separator between exception entries. A trailing '\' at the end of a line continues the current line on the next one.

A control instructions file is parsed line by line; later entries override earlier entries (in case there is something to override).

4.2 Style Adjustments

This syntax form makes it possible to override the style assignment algorithm of ttfautohint; see 'Scripts ([section 3.9](#))' and 'OpenType Features ([section 3.10](#))' for more details.

> [*font-idx*] *script feature* @ *glyph-ids*

script is a four-letter name[1] of one of the scripts supported by ttfautohint. *feature* is one of the four-letter names of features supported by ttfautohint.

The elements of *glyph-ids* are a list of comma separated *glyph-id* values or value ranges. Note that is not necessary that elements are specified in increasing order.

Assuming that a font contains superscript digits 'zero.sups' to 'nine.sups' together with the glyphs 'a.sups' and 'o.sups', use a line

---

[1]The notable exception is the tag 'lao', which originally has a trailing space as the fourth character. However, ttfautohint ignores the space.

```
cyrl sups @ zero.sups-nine.sups, a.sups, o.sups
```

to add those glyphs to the style handling Cyrillic superscript glyphs. However, it is still necessary that the selected script contains proper Blue Zone characters (section 3.3), otherwise those glyphs aren't handled at all.

Use the --debug command line option to see how ttfautohint assigns glyph indices of a font to styles.

4.3 Stem Width Adjustments

Use the following syntax form to adjust stem width values for a given style, overriding ttfautohint's algorithm; see 'Scripts (section 3.9)' and 'OpenType Features (section 3.10)' for more details. This adjustment doesn't change the glyph shapes; it only influences the hinting process.

[*font-idx*] *script feature* w[idth] *stem-widths*

script and *feature* are the same as with style adjustments; see above. However, *script* can additionally be the wildcard character '*', which indicates 'any script'. Parameter width can be abbreviated as 'w'.

stem-widths is an unsorted list of comma separated integer stem width values (in font units); the first value gives the style's default stem width.

The effect of this adjustment depends on the selected stem width algorithm (subsection 2.3.19). For smooth stem width positioning, only the first (i.e., the default) value is used; it gets a higher preference than other discrete stem width choices. For strong stem width positioning, the stem widths are snapped to the provided values (if not differing too much) before rounding to an integer pixel value. For natural stem width positioning, this adjustment is ignored.

A typical example is to reduce the default stem width of an extra-bold font, which gets better hinted if a stem width of, say, 100 is used instead of the default value 150. Let's also assume that the font further contains latin subscript and superscript characters that are hinted best with a stem width set to 80 font units. We can achieve this with the following lines in a control instructions file.

```
* dflt width 100
latb dflt width 80
latp dflt width 80
```

Without the adjustment chances are very high that the 'eyes' in glyphs 'e' or 'a' of extra-bold fonts are filled at smaller PPEM values.

Use the --debug command line option to see how ttfautohint assigns stem widths to styles by default.

4.4 Glyph Adjustments

The following syntax forms allows adjustments of a glyph's hinting process.
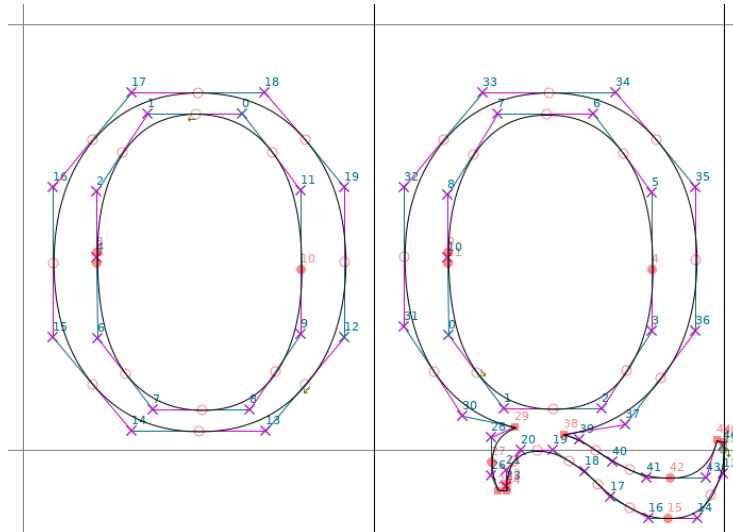
Figure 4.1: The outlines of glyphs 'O' and 'Q', as displayed in FontForge. They are sufficiently similar to expect that ttfautohint hints them equally. However, this is not the case.

4.4.1 Change Direction of Points, Artificial Segments

[*font-idx*] *glyph-id* l[eft]|r[ight] *points* [(*left-offset* , *right-offset*)]

The mutually exclusive parameters left and right (which can be abbreviated as 'l' and 'r', respectively) indicate that the following points have left or right 'out' direction, respectively, overriding ttfautohint's algorithm for setting point directions. The 'out direction' of a point is the direction of the outline *leaving* the point (or passing the control point). If the specified direction is identical to what ttfautohint computes, nothing special happens. Otherwise, a one-point segment with the specified direction gets created, see above (section 3.1). By default, its length is zero. Setting *left-offset* and *right-offset*, you can change the segment's horizontal start and end position relative to the point position. *left-offset* and *right-offset* are integers measured in font units.

The following five images, displaying glyphs 'O' and 'Q' from the font Halant-Regular, demonstrate how to use direction changes.

4.4.2 Unset Direction of Points

[*font-idx*] *glyph-id* n[odir] *points*

Parameter nodir (or 'n') sets the 'out' direction of the following points to 'no direction', this is, neither left nor right. If the specified direction is identical to what ttfautohint computes, nothing special happens. Otherwise, ttfautohint no longer considers those points as part of horizontal segments, thus treating them as 'weak' (section 3.4) points.

Modifying or adding segments doesn't directly modify the outlines; it only influences the hinting process.
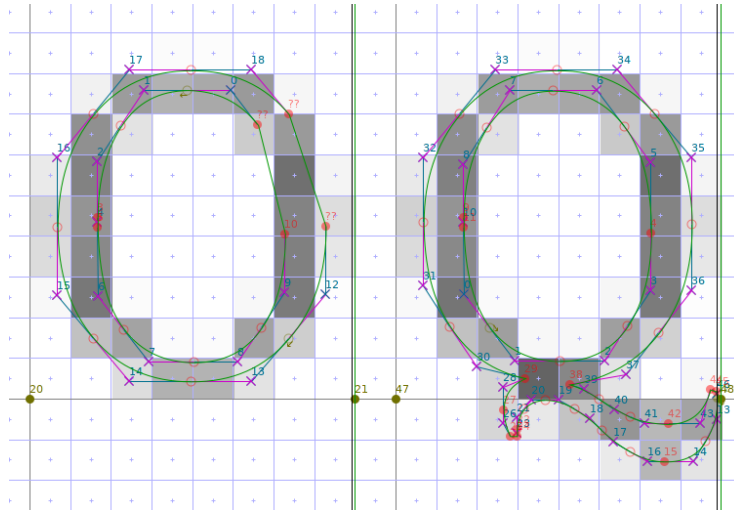
Figure 4.2: The same glyphs, shown at 12px before hinting. [Please ignore the outline distortion in the upper right of glyph 'O'; this is a bug in FontForge while running the TrueType debugger.]
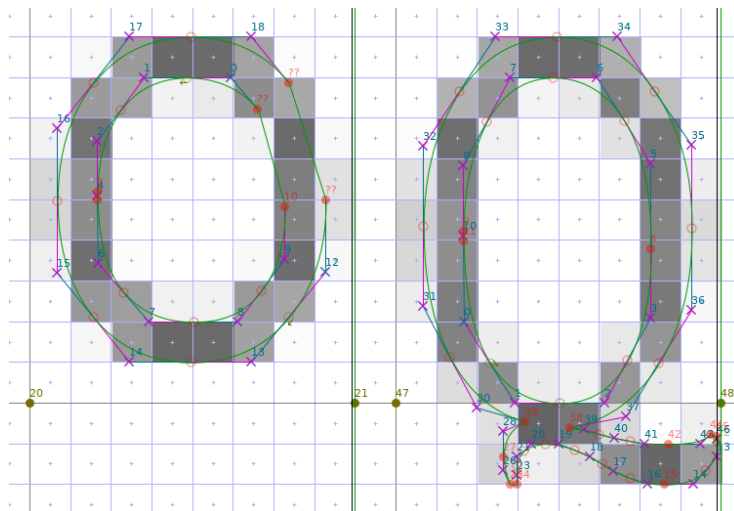
Figure 4.3: Using only ttfautohint's '-a sss' parameter to force strong stem width and positioning, the hinting of glyph 'Q' is really bad, making the glyph vertically two pixels larger! Reason is that this glyph doesn't contain a horizontal segment at the baseline blue zone ($y = 1$; this corresponds to the segment 13-14 in the 'O' glyph). Normally, segment 1-2 would form a 'stem' with the baseline segment (as segment 7-8 does in glyph 'O'). Instead, it forms a stem with segment 19-20, which gets moved down ($y = -1$) because the whole glyph appears to be stretched.

4.4.3 Delta Exceptions

[*font-idx*] *glyph-id* t[ouch]|p[oint] *points* [x[shift] *x-shift*] [y[shift] *y-shift*] @ *ppems*

The mutually exclusive parameters `touch` and `point` (which can be abbreviated as 't' and 'p', respectively) make ttfautohint apply delta exceptions for the given points, shifting them by the given values. Delta exceptions entered with `touch` are applied before the final 'IUP' (*interpolate untouched points*) instructions in a glyph's bytecode, exceptions entered with `point` after 'IUP' (please consult Greg Hitch-
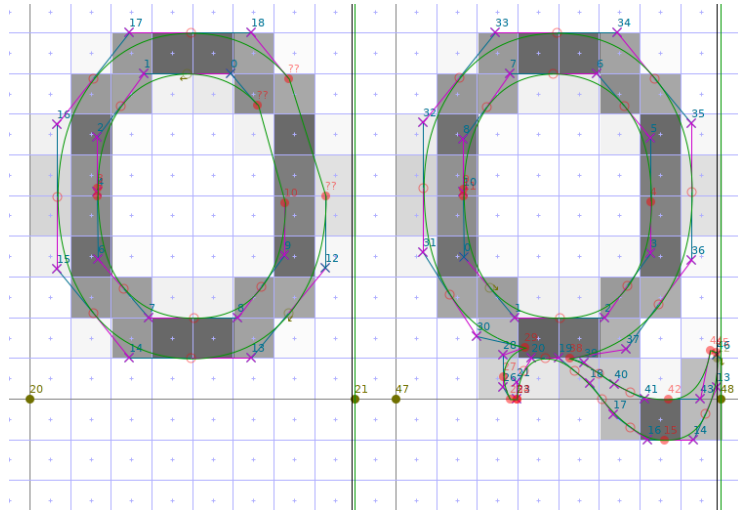
Figure 4.4: To fix the problem, we change the direction of point 38 to 'left' by writing a line 'Q left 38' (without the quotes) to a control description file `Halant-Regular.txt`. Adding option '-m Halant-Regular.txt' to ttfautohint, we get the shown image as a result, which is much better: Segment 1-2 now properly forms a stem with our artificial one-point segment 38, and the 'O'-like shape is properly positioned. However, there is still room for improvement: Segment 19-20 is also positioned at the baseline, making the connection between the 'O' shape and the tail too thin.

cock's ClearType Whitepaper for more on pre-IUP and post-IUP delta hints). Additionally, the `touch` parameter makes the bytecode *touch* the affected points; such points are no longer affected by 'IUP' at all. Note that in ClearType mode all deltas along the x axis are discarded, and deltas along the y axis are only executed for touched points. As a consequence, vertical delta exceptions entered with `point` should not be used in ClearType mode.[2]

ppems, similar to *points*, are number ranges, see 'x Height Snapping Exceptions (subsection 2.3.8)' for the syntax.

x-shift and *y-shift* represent real numbers that get rounded to multiples of 1/8 pixels. The entries for xshift ('x') and yshift ('y') are optional; if missing, the corresponding value is set to zero. If both values are zero, the delta exception entry is ignored as a whole.

Values for *x-shift* and *y-shift* must be in the range [−1.0;1.0]. Values for *ppems* must be in the range [6;53]. Values for *points* are limited by the number of points in the glyph.

Note that only character '.' is recognized as a decimal point, and a thousands separator is not accepted.

As an example for delta instructions, let's assume that you want to shift points 2, 3, and 4 in glyph 'Aacute' at PPEM sizes 12 and 13 by a vertical amount of 0.25 pixels. This corresponds to the line

```
Aacute  touch 2-4  yshift 0.25  @ 12, 13
```

in a control instructions file. Since we use `touch` and not `point`, points 2, 3, and 4 are no longer subject to the final 'IUP' instruction, which interpolates weak, untouched point positions between strong, touched ones, cf. the description here.

---

[2]Unfortunately, there is a bug in FreeType prior to version 2.5.4 (released in December 2014) that completely disables vertical delta exceptions if subpixel hinting is activated. For this reason you should expect that the `touch` parameter fails on older GNU/Linux distributions.
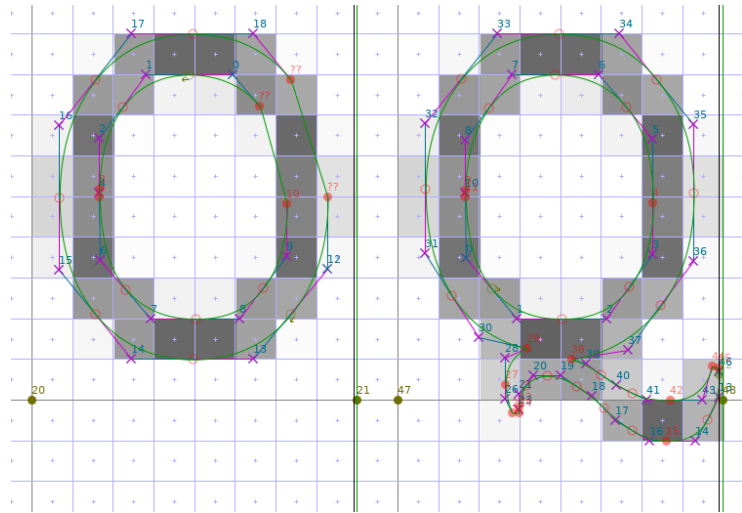
Figure 4.5: By giving the one-point segment 38 a horizontal width, we can prevent that segment 19-20 gets positioned at the baseline: Replace the line in the previous image description with 'Q left 38 (-70,20)', making the segment extend 70 font units to the left and 20 to the right of point 38. The exact offset values don't matter; it's only important to start left of point 19. Another solution to the problem is to artificially change the direction of segment 19-20 by adding a second line 'Q right 19-20' to the control instructions file; for our 'Q' glyph, this produces almost exactly the same hinting results. Note that such direction changes only influence the hinting process; an outline's direction won't be changed at all.

5 The ttfautohint API

This section documents the public functions of the ttfautohint library together with its callback functions. All information has been directly extracted from the `ttfautohint.h` header file.

5.1 Preprocessor Macros, Typedefs, and Enums

Some default values.

```
#define TA_HINTING_RANGE_MIN 8
#define TA_HINTING_RANGE_MAX 50
#define TA_HINTING_LIMIT 200
#define TA_INCREASE_X_HEIGHT 14
```

An error type.

```
typedef int TA_Error;
```

An enum type for stem width algorithm selection.

```
enum
{
  TA_STEM_WIDTH_MODE_NATURAL = -1,
  TA_STEM_WIDTH_MODE_QUANTIZED = 0,
  TA_STEM_WIDTH_MODE_STRONG = 1
};
```

5.2 Function Pointer: `TA_Alloc_Func`

A pointer to a function provided by the calling application to allocate memory. The ttfautohint library uses this for allocating the buffer given by the `out-buffer` field of `TTF_autohint` (section 5.8) and for allocating the `str` buffer in the `TA_Info_Func` (section 5.6) callback.

The signature is identical to standard C's `malloc` function (in header file `stdlib.h`).

```
typedef void *
(*TA_Alloc_Func)(size_t size);
```

5.3 Function Pointer: `TA_Free_Func`

A pointer to a function provided by the calling application to free memory allocated with `TA_Alloc_Func` (section 5.2). The ttfautohint library calls this for deallocating the `str` buffers in the `TA_Info_Func` (section 5.6) callback after they have been used.

The signature is identical to standard C's `free` function (in header file `stdlib.h`).

```
typedef void
(*TA_Free_Func)(void *ptr);
```

5.4 Callback: `TA_Progress_Func`

A callback function to get progress information. *curr_idx* gives the currently processed glyph index; if it is negative, an error has occurred. *num_glyphs* holds the total number of glyphs in the font (this value can't be larger than 65535).

curr_sfnt gives the current subfont within a TrueType Collection (TTC), and *num_sfnts* the total number of subfonts.

If the return value is non-zero, `TTF_autohint` aborts with `TA_Err_Canceled`. Use this for a 'Cancel' button or similar features in interactive use.

progress_data is a void pointer to user-supplied data.

```
typedef int
(*TA_Progress_Func)(long curr_idx,
                    long num_glyphs,
                    long curr_sfnt,
                    long num_sfnts,
                    void* progress_data);
```

5.5 Callback: `TA_Error_Func`

A callback function to get error information.

error is the value `TTF_autohint` returns. See file `ttfautohint-errors.h` for a list. Error codes not in this list are directly taken from FreeType; see the FreeType header file `fterrdef.h` for more.

error_string, if non-NULL, is a pointer to an error message that represents *error*.

The next three parameters help identify the origin of text string parsing errors. *linenum*, if non-zero, contains the line number. *line*, if non-NULL, is a pointer to the input line that can't be processed. *errpos*, if non-NULL, holds a pointer to the position in *line* where the problem occurs.

error_data is a void pointer to user-supplied data.

```
typedef void
(*TA_Error_Func)(TA_Error error,
                 const char* error_string,
                 unsigned int linenum,
                 const char* line,
                 const char* errpos,
                 void* error_data);
```

5.6 Callback: `TA_Info_Func`

A callback function to access or modify strings in the `name` table; it is called in a loop that iterates over all `name` table entries. If defined, `TA_Info_Post_Func` (section 5.7) gets executed after this loop so that the collected data can be written back to the `name` table.

platform_id, *encoding_id*, *language_id*, and *name_id* are the identifiers of a `name` table entry pointed to by *str* with a length pointed to by *str_len* (in bytes; the string has no trailing NULL byte). Please refer to the OpenType specification of the name table for a detailed description of the various parameters, in particular which encoding is used for a given platform and encoding ID.

The string *str* is allocated with the function specified by the `alloc-func` field of `TTF_autohint` (section 5.8); the application should reallocate the data if necessary, ensuring that the string length doesn't exceed 0xFFFF.

info_data is a void pointer to user-supplied data.

If an error occurs, return a non-zero value and don't modify *str* and *str_len* (such errors are handled as non-fatal).

```
typedef int
(*TA_Info_Func)(unsigned short platform_id,
                unsigned short encoding_id,
                unsigned short language_id,
                unsigned short name_id,
                unsigned short* str_len,
                unsigned char** str,
                void* info_data);
```

5.7 Callback: `TA_Info_Post_Func`

A callback function, giving the application the possibility to access or modify strings in the `name` table after `TA_Info_Func` (section 5.6) has iterated over all `name` table entries.

It is expected that `TA_Info_Func` stores pointers to the `name` table entries it wants to access or modify; the only parameter is thus *info_data*, which is a void pointer to the user-supplied data already provided to `TA_Info_Func`. Obviously, calling `TA_Info_Post_Func` with `TA_Info_Func` undefined has no effect.

The `name` table strings are allocated with the function specified by the `alloc-func` field of `TTF_autohint` (section 5.8); the application should reallocate the data if necessary, ensuring that no string length exceeds 0xFFFF.

If an error occurs, return a non-zero value and don't modify the affected string and string length (such errors are handled as non-fatal).

```
typedef int
(*TA_Info_Post_Func)(void* info_data);
```

5.8 Function: `TTF_autohint`

Read a TrueType font, remove existing bytecode (in the SFNT tables `prep`, `fpgm`, `cvt`, and `glyf`), and write a new TrueType font with new bytecode based on the autohinting of the FreeType library, optionally using a reference font to derive blue zones.

It expects a format string *options* and a variable number of arguments, depending on the fields in *options*. The fields are comma separated; whitespace within the format string is not significant, a trailing comma is ignored. Fields are parsed from left to right; if a field occurs multiple times, the last field's argument wins. The same is true for fields that are mutually exclusive. Depending on the field, zero or one argument is expected.

Note that fields marked as 'not implemented yet' are subject to change.

5.8.1 Memory Management

The next two fields are necessary on some platforms if ttfautohint is compiled as a shared library, and the application uses a different runtime library. This can happen, for example, on the MS Windows platform if your program is written in Python and communicates via the 'ctypes' interface with the ttfautohint DLL.

alloc-func
> A pointer of type `TA_Alloc_Func` (section 5.2) specifying a memory allocation function. It gets used to allocate the buffer given by the `out-buffer` field and the data exposed by the `TA_Info_Func` (section 5.6) callback. If not set or set to NULL, or if `out-buffer` is not set or set to NULL, standard C's `malloc` function is used.

free-func
> A pointer of type `TA_Free_Func` (section 5.3) specifying a memory deallocation function. It gets called to free the data exposed by the `TA_Info_Func` (section 5.6) callback after it has been used. If not set or set to NULL, or if `out_buffer` is not set or set to NULL, standard C's `free` function is used.

5.8.2 I/O

in-file
> A pointer of type `FILE*` to the data stream of the input font, opened for binary reading. Mutually exclusive with `in-buffer`.

in-buffer
> A pointer of type `const char*` to a buffer that contains the input font. Needs `in-buffer-len`. Mutually exclusive with `in-file`.

in-buffer-len
> A value of type `size_t`, giving the length of the input buffer. Needs `in-buffer`.

out-file
> A pointer of type `FILE*` to the data stream of the output font, opened for binary writing. Mutually exclusive with `out-buffer`.

`out-buffer`

A pointer of type `char**` to a buffer that contains the output font. Needs `out-buffer-len`. Mutually exclusive with `out-file`. The application should deallocate the memory with the function given by `free-func`.

`out-buffer-len`

A pointer of type `size_t*` to a value giving the length of the output buffer. Needs `out-buffer`.

`control-file`

A pointer of type `FILE*` to the data stream of control instructions. Mutually exclusive with `control-buffer`.

See 'Control Instructions (chapter 4)' for the syntax used in such a file or buffer.

`control-buffer`

A pointer of type `const char*` to a buffer that contains control instructions. Needs `control-buffer-len`. Mutually exclusive with `control-file`.

`control-buffer-len`

A value of type `size_t`, giving the length of the control instructions buffer. Needs `control-buffer`.

`reference-file`

A pointer of type `FILE*` to the data stream of the reference font, opened for binary reading. Mutually exclusive with `reference-buffer`.

`reference-buffer`

A pointer of type `const char*` to a buffer that contains the reference font. Needs `reference-buffer-len`. Mutually exclusive with `reference-file`.

`reference-buffer-len`

A value of type `size_t`, giving the length of the reference buffer. Needs `reference-buffer`.

`reference-index`

The face index to be used in the reference font. The default value is 0.

`reference-name`

A string that specifies the name of the reference font. It is only used to emit a sensible value for the `TTFA` table if `TTFA-info` is set.

5.8.3 Messages and Callbacks

`progress-callback`

A pointer of type `TA_Progress_Func` (section 5.4), specifying a callback function for progress reports. This function gets called after a single glyph has been processed. If this field is not set or set to NULL, no progress callback function is used.

`progress-callback-data`

A pointer of type `void*` to user data that is passed to the progress callback function.

`error-string`

A pointer of type `unsigned char**` to a string (in UTF-8 encoding) that verbally describes the error code. You must not change the returned value.

`error-callback`

> A pointer of type `TA_Error_Func` ([section 5.5](#)), specifying a callback function for error messages. This function gets called right before `TTF_autohint` exits. If this field is not set or set to NULL, no error callback function is used.
>
> Use it as a more sophisticated alternative to `error-string`.

`error-callback-data`

> A point of type `void*` to user data that is passed to the error callback function.

`info-callback`

> A pointer of type `TA_Info_Func` ([section 5.6](#)), specifying a callback function for manipulating the `name` table. This function gets called for each `name` table entry. If not set or set to NULL, `TA_Info_Func` is not called.

`info-post-callback`

> A pointer of type `TA_Info_Post_Func` ([section 5.7](#)), specifying a callback function for manipulating the `name` table. It is called after the function specified with `info-callback` has iterated over all `name` table entries. If not set or set to NULL, `TA_Info_Post_Func` is not called.

`info-callback-data`

> A pointer of type `void*` to user data that is passed to the info callback functions.

`debug`

> If this integer is set to 1, lots of debugging information is print to stderr. The default value is 0.

5.8.4 General Hinting Options

`hinting-range-min`

> An integer (which must be larger than or equal to 2) giving the lowest PPEM value used for autohinting. If this field is not set, it defaults to `TA_HINTING_RANGE_MIN` ([section 5.1](#)).

`hinting-range-max`

> An integer (which must be larger than or equal to the value of `hinting-range-min`) giving the highest PPEM value used for autohinting. If this field is not set, it defaults to `TA_HINTING_RANGE_MAX` ([section 5.1](#)).

`hinting-limit`

> An integer (which must be larger than or equal to the value of `hinting-range-max`) that gives the largest PPEM value at which hinting is applied. For larger values, hinting is switched off. If this field is not set, it defaults to `TA_HINTING_LIMIT` ([section 5.1](#)). If it is set to 0, no hinting limit is added to the bytecode.

`hint-composites`

> If this integer is set to 1, composite glyphs get separate hints. This implies adding a special glyph to the font called '.ttfautohint' ([section 3.7](#)). Setting it to 0 (which is the default), the hints of the composite glyphs' components are used. Adding hints for composite glyphs increases the size of the resulting bytecode a lot, but it might deliver better hinting results. However, this depends on the processed font and must be checked by inspection.

`adjust-subglyphs`

> An integer (1 for 'on' and 0 for 'off', which is the default) to specify whether native TrueType hinting of the *input font* shall be applied to all glyphs before passing them to the (internal) autohinter.

The used resolution is the em-size in font units; for most fonts this is 2048ppem. Use this only if the old hints move or scale subglyphs independently of the output resolution, for example some exotic CJK fonts.

`pre-hinting` is a deprecated alias name for this option.

5.8.5 Hinting Algorithms

ttfautohint provides three different algorithms for computing horizontal stem widths and the positioning of blue zones.

- `TA_STEM_WIDTH_MODE_NATURAL`: No adjustments to stem widths, discrete blue zone positioning. This is what FreeType uses for its 'light' (auto-)hinting mode.

- `TA_STEM_WIDTH_MODE_QUANTIZED`: Both stem widths and blue zone positions are slightly quantized to take discrete values. For example, stem values 50, 51, 72, 76, and 100 would become 50, 74, and 100 (or something similar).

- `TA_STEM_WIDTH_MODE_STRONG`: If active, stem widths and blue zones are snapped and positioned to integer pixel values as much as possible.

These values are arguments to ttfautohint's three different hinting mode options.

`gray-stem-width-mode`
   Specify the stem width algorithm for grayscale rendering. Possible integer values are `TA_STEM_WIDTH_MODE_NATURAL`, `TA_STEM_WIDTH_MODE_QUANTIZED` (the default), and `TA_STEM_WIDTH_MODE_STRONG`, as discussed above.

`gdi-cleartype-stem-width-mode`
   Specify the stem width algorithm for GDI ClearType rendering, this is, the rasterizer version (as returned by the GETINFO bytecode instruction) is in the range 36 <= version < 38 and ClearType is enabled. Possible integer values are `TA_STEM_WIDTH_MODE_NATURAL`, `TA_STEM_WIDTH_MODE_QUANTIZED`, and `TA_STEM_WIDTH_MODE_STRONG` (the default), as discussed above.

`dw-cleartype-stem-width-mode`
   Specify the stem width algorithm for DW ClearType rendering, this is, the rasterizer version (as returned by the GETINFO bytecode instruction) is >= 38, ClearType is enabled, and subpixel positioning is enabled also. Possible integer values are `TA_STEM_WIDTH_MODE_NATURAL`, `TA_STEM_WIDTH_MODE_QUANTIZED` (the default), and `TA_STEM_WIDTH_MODE_STRONG`, as discussed above.

`increase-x-height`
   An integer. For PPEM values in the range 6 <= PPEM <= `increase-x-height`, round up the font's x height much more often than normally (to use the terminology of TrueType's 'Super Round' bytecode instruction, the threshold gets increased from 5/8px to 13/16px). If it is set to 0, this feature is switched off. If this field is not set, it defaults to `TA_INCREASE_X_HEIGHT` (section 5.1). Use this flag to improve the legibility of small font sizes if necessary.

`x-height-snapping-exceptions`
   A pointer of type `const char*` to a null-terminated string that gives a list of comma separated PPEM values or value ranges at which no x height snapping shall be applied. A value range has the form $value_1$–$value_2$, meaning $value_1$ <= PPEM <= $value_2$. $value_1$ or $value_2$ (or both) can be missing; a missing value is replaced by the beginning or end of the whole interval of valid PPEM

values, respectively. Whitespace is not significant; superfluous commas are ignored, and ranges must be specified in increasing order. For example, the string `"3, 5-7, 9-"` means the values 3, 5, 6, 7, 9, 10, 11, 12, etc. Consequently, if the supplied argument is `"-"`, no x height snapping takes place at all. The default is the empty string (`""`), meaning no snapping exceptions.

`windows-compatibility`

> If this integer is set to 1, two artificial blue zones are used, positioned at the `usWinAscent` and `usWinDescent` values (from the font's `OS/2` table). The idea is to help ttfautohint so that the hinted glyphs stay within this horizontal stripe since Windows clips everything falling outside. The default is 0.

`gray-strong-stem-width`

> Deprecated. The argument values 0 and 1 of this field correspond to the argument values `TA_STEM_WIDTH_MODE_QUANTIZED` and `TA_STEM_WIDTH_MODE_STRONG` of the field 'gray-stem-width-mode', respectively.

`gdi-cleartype-strong-stem-width`

> Deprecated. The argument values 0 and 1 of this field correspond to the argument values `TA_STEM_WIDTH_MODE_QUANTIZED` and `TA_STEM_WIDTH_MODE_STRONG` of the field 'gdi-cleartype-stem-width-mode', respectively.

`dw-cleartype-strong-stem-width`

> Deprecated. The argument values 0 and 1 of this field correspond to the argument values `TA_STEM_WIDTH_MODE_QUANTIZED` and `TA_STEM_WIDTH_MODE_STRONG` of the field 'dw-cleartype-stem-width-mode', respectively.

5.8.6 Scripts

`default-script`

> A string consisting of four lowercase characters that specifies the default script for OpenType features. After applying all features that are handled specially, use this value for the remaining features. The default value is `"latn"`; if set to `"none"`, no script is used. Valid values can be found in the header file `ttfautohint-scripts.h`.

`fallback-script`

> A string consisting of four lowercase characters, specifying the default script for glyphs that can't be mapped to a script automatically. By default, such glyphs are hinted; if option `fallback-scaling` is set, they are scaled only instead. Valid values can be found in the header file `ttfautohint-scripts.h`.
>
> Default value is `"none"`, which means hinting without using a script's blue zones if `fallback-scaling` isn't set. If `fallback_scaling` is set, value `"none"` implies no hinting for unmapped glyphs.

`fallback-scaling`

> Set this integer to 1 if glyphs handled by the fallback script should be scaled only with the fallback script's scaling value, instead of being hinted with the fallback script's hinting parameters.

`symbol`

> Set this integer to 1 if you want to process a font that ttfautohint would refuse otherwise because it can't find a single standard character for any of the supported scripts. ttfautohint then uses a default (hinting) value for the standard stem width instead of deriving it from a script's set of standard characters (for the latin script, one of them is character 'o'). The default value of this option is 0.

`fallback-stem-width`

> Set the horizontal stem width (hinting) value for all scripts that lack proper standard characters. The value is given in font units and must be a positive integer. If not set, or the value is zero, ttfautohint uses a hard-coded default (50 units at 2048 units per EM, and linearly scaled for other UPEM values, for example 24 units at 1000 UPEM).
>
> For symbol fonts (i.e., option `symbol` is given), `fallback-stem-width` has an effect only if `fallback-script` is set also.

5.8.7 Miscellaneous

`ignore-restrictions`

> If the font has set bit 1 in the 'fsType' field of the `OS/2` table, the ttfautohint library refuses to process the font since a permission to do that is required from the font's legal owner. In case you have such a permission you might set the integer argument to value 1 to make ttfautohint handle the font. The default value is 0.

`TTFA-info`

> If set to 1, ttfautohint creates an SFNT table called `TTFA` and fills it with information on the parameters used while calling `TTF_autohint`. The format of the output data resembles the information at the very beginning of the dump emitted by option `debug`. The default value is 0.
>
> Main use of this option is for font editing purposes. For example, after a font editor has added some glyphs, a front-end to `TTF_autohint` can parse `TTFA` and feed the parameters into another call of `TTF_autohint`. The new glyphs are then hinted while hints of the old glyphs stay unchanged.
>
> If this option is not set, and the font to be processed contains a `TTFA` table, it gets removed.
>
> Note that such a `TTFA` table gets ignored by all font rendering engines. In TrueType Collections, the `TTFA` table is added to the first subfont.

`dehint`

> If set to 1, remove all hints from the font. All other hinting options are ignored.

`epoch`

> An integer of type `unsigned long long`, defined as the number of seconds (excluding leap seconds) since 01 Jan 1970 00:00:00 UTC. If set, or if the value is not equal to `ULLONG_MAX`, this epoch gets used instead of the current date and time for the 'modification time' field in the TTF header. Use this to get reproducible builds.

5.8.8 Remarks

- Obviously, it is necessary to have an input and an output data stream. All other options are optional.

- `hinting-range-min` and `hinting-range-max` specify the range for which the autohinter generates optimized hinting code. If a PPEM value is smaller than the value of `hinting-range-min`, hinting still takes place but the configuration created for `hinting-range-min` is used. The analogous action is taken for `hinting-range-max`, only limited by the value given with `hinting-limit`. The font's gasp table is set up to always use grayscale rendering with grid-fitting for standard hinting, and symmetric grid-fitting and symmetric smoothing for horizontal subpixel hinting (ClearType).

- ttfautohint can process its own output a second time only if option `hint-composites` is not set (or if the font doesn't contain composite glyphs at all). This limitation might change in the future.

```
TA_LIB_EXPORT TA_Error
TTF_autohint(const char* options,
             ...);
```

5.9 Macros: `TTFAUTOHINT_MAJOR`, `TTFAUTOHINT_MINOR`, `TTFAUTOHINT_REVISION`

These three macros give the major, minor, and revision number of the library, respectively. See function `TTF_autohint_version` (section 5.11) for more details.

```
#define TTFAUTOHINT_MAJOR %TTFAUTOHINT_MAJOR%
#define TTFAUTOHINT_MINOR %TTFAUTOHINT_MINOR%
#define TTFAUTOHINT_REVISION %TTFAUTOHINT_REVISION%
```

5.10 Macro: `TTFAUTOHINT_VERSION`

This macro holds the ttfautohint version string.

For tarball releases it has the form *X.Y*[.*Z*], with *X*, *Y*, and *Z* the major, minor, and revision numbers, respectively. If the revision number is zero, it is omitted. Examples: `2.7`, `2.7.1`.

If compiling directly from the git repository, ttfautohint's bootstrap script derives the version number from the `git describe` output, appending the number of commits after a tag together with a shortened commit ID. Example: `2.7.1.23-379b`.

See function `TTF_autohint_version` (section 5.11) for more details.

```
#define TTFAUTOHINT_VERSION "%TTFAUTOHINT_VERSION%"
```

5.11 Function: `TTF_autohint_version`

Return the ttfautohint version triplet. This function is useful when dynamically linking to the library, since the macros `TTFAUTOHINT_MAJOR`, `TTFAUTOHINT_MINOR`, and `TTFAUTOHINT_PATCH` cannot be used in that case.

The returned integer triplet is *not* the same as the shared library's version triplet (which the dynamic linker uses to resolve runtime dependencies). For example, hypothetical ttfautohint versions 2.27 and 2.27.1 might both correspond to shared library version 1.0.38 in case there are only, say, documentation or packaging differences that don't affect the library code.

If the pkg-config configuration file `ttfautohint.pc` is installed in a place that pkg-config can find, a call to

```
    pkg-config ttfautohint --modversion
```

returns the shared library version.

```
TA_LIB_EXPORT void
TTF_autohint_version(int *major,
                     int *minor,
                     int *revision);
```

5.12 Function: `TTF_autohint_version_string`

Return the ttfautohint version string. This function is useful when dynamically linking to the library, since the macro `TTFAUTOHINT_VERSION` cannot be used in that case.

```
TA_LIB_EXPORT const char*
TTF_autohint_version_string(void);
```

6 Compilation and Installation

Please read the files `INSTALL` and `INSTALL.git` (both part of the source code bundle) for instructions how to compile the ttfautohint library together with its front-ends using a POSIX compatible shell and compiler.

6.1 Unix-like Platforms

The generic instructions should work just fine. Since ttfautohint depends on Qt version 4 or newer, FreeType version 2.4.5 or newer, and HarfBuzz version 1.3.0 or newer, you should install packages for these libraries (called 'libqt4' or similar, 'libfreetype6'[1], and 'libharfbuzz0' or similar) together with its development bundles (called 'libqt4-devel', 'freetype2-devel', and 'harfbuzz-devel' or similar) before running ttfautohint's `configure` script.

6.2 MS Windows

Precompiled binaries `ttfautohint.exe` and `ttfautohintGUI.exe` are available, being statically linked to Qt, FreeType, and HarfBuzz. This means that the two programs are not dependent on any other program-specific DLL, and you can move them to any place you like.

Hints for compilation with the MinGW environment are given in `INSTALL.git`.

6.3 Mac OS X

Right now, only a precompiled binary `ttfautohint` is offered; a ready-to-run app bundle for the GUI version is not yet available; however, ttfautohint is part of Homebrew, making compilation and installation very simple.

Detailed instructions to compile both `ttfautohint` and `ttfautohintGUI` can be found on ttfautohint's homepage.

---

[1] The number '6' indicates the version of the shared library of FreeType, which is not directly related to the source code version of FreeType.

7 Authors

Copyright © 2011-2021 by Werner Lemberg.
Portions Copyright © 2011-2017 by Dave Crossland.
Portions Copyright © 2014 by Adam Twardoch.

This file is part of the ttfautohint library, and may only be used, modified, and distributed under the terms given in `COPYING`. By continuing to use, modify, or distribute this file you indicate that you have read `COPYING` and understand and accept it fully.

The file `COPYING` mentioned in the previous paragraph is distributed with the ttfautohint library.

8 Character Ranges

For the FreeType auto-hinter (and thus ttfautohint), a 'non-base character' is something that should not be affected by blue zones, regardless of whether this is a spacing or no-spacing glyph.

Table 8.1: `adlm` base characters

Character range	Description
0x1E900 - 0x1E95F	Adlam

Table 8.2: `adlm` non-base characters

Character range
0x1D944 - 0x1E94A

Table 8.3: `arab` base characters

Character range	Description
0x0600 - 0x06FF	Arabic
0x0750 - 0x07FF	Arabic Supplement
0x08A0 - 0x08FF	Arabic Extended-A
0xFB50 - 0xFDFF	Arabic Presentation Forms-A
0xFE70 - 0xFEFF	Arabic Presentation Forms-B
0x1EE00 - 0x1EEFF	Arabic Mathematical Alphabetic Symbols

Table 8.4: `arab` non-base characters

Character range
0x0600 - 0x0605
0x0610 - 0x061A
0x064B - 0x065F
0x0670 - 0x0670
0x06D6 - 0x06DC
0x06DF - 0x06E4
0x06E7 - 0x06E8
0x06EA - 0x06ED
0x08D3 - 0x08FF
0xFBB2 - 0xFBC1

Character range
0xFE70 - 0xFE70
0xFE72 - 0xFE72
0xFE74 - 0xFE74
0xFE76 - 0xFE76
0xFE78 - 0xFE78
0xFE7A - 0xFE7A
0xFE7C - 0xFE7C
0xFE7E - 0xFE7E

Table 8.5: `armn` base characters

Character range	Description
0x0530 - 0x058F	Armenian
0xFB13 - 0xFB17	Alphab. Present. Forms (Armenian)

Table 8.6: `armn` non-base characters

Character range
0x0559 - 0x055F

Table 8.7: `avst` base characters

Character range	Description
0x10B00 - 0x10B3F	Avestan

Table 8.8: `avst` non-base characters

Character range
0x10B39 - 0x10B3F

Table 8.9: `bamu` base characters

Character range	Description
0xA6A0 - 0xA6FF	Bamum

Table 8.10: `bamu` non-base characters

Character range
0xA6F0 - 0xA6F1

Table 8.11: `beng` base characters

Character range	Description
0x0980 - 0x09FF	Bengali

Table 8.12: `beng` non-base characters

Character range
0x0981 - 0x0981
0x09BC - 0x09BC
0x09C1 - 0x09C4
0x09CD - 0x09CD
0x09E2 - 0x09E3
0x09FE - 0x09FE

Table 8.13: `buhd` base characters

Character range	Description
0x1740 - 0x175F	Buhid

Table 8.14: `buhd` non-base characters

Character range
0x1752 - 0x1753

Table 8.15: `cakm` base characters

Character range	Description
0x11100 - 0x1114F	Chakma

Table 8.16: `cakm` non-base characters

Character range
0x11100 - 0x11102

```
0x11127 - 0x11134
0x11146 - 0x11146
```

Table 8.17: `cans` base characters

Character range	Description
0x1400 - 0x167F	Unified Canadian Aboriginal Syllabics
0x18B0 - 0x18FF	Unified Canadian Aboriginal Syllabics Extended

Table 8.18: `cari` base characters

Character range	Description
0x102A0 - 0x102DF	Carian

Table 8.19: `cher` base characters

Character range	Description
0x13A0 - 0x13FF	Cherokee
0xAB70 - 0xABBF	Cherokee Supplement

Table 8.20: `copt` base characters

Character range	Description
0x2C80 - 0x2CFF	Coptic

Table 8.21: `copt` non-base characters

Character range
0x2CEF - 0x2CF1

Table 8.22: `cprt` base characters

Character range	Description
0x10800 - 0x1083F	Cypriot

Table 8.23: `cyrl` base characters

Character range	Description
0x0400 - 0x04FF	Cyrillic
0x0500 - 0x052F	Cyrillic Supplement
0x2DE0 - 0x2DFF	Cyrillic Extended-A
0xA640 - 0xA69F	Cyrillic Extended-B

Table 8.24: `cyrl` non-base characters

Character range
0x0483 - 0x0489
0x2DE0 - 0x2DFF
0xA66F - 0xA67F
0xA69E - 0xA69F

There are some characters in the Devanagari Unicode block that are generic to Indic scripts; we omit them so that their presence doesn't trigger Devanagari.

Table 8.25: `deva` base characters

Character range	Description
0x0900 - 0x093B	Devanagari
0x093D - 0x0950	... continued
0x0953 - 0x0963	... continued
0x0966 - 0x097F	... continued
0x20B9 - 0x20B9	(new) Rupee sign
0xA8E0 - 0xA8FF	Devanagari Extended

Table 8.26: `deva` non-base characters

Character range
0x0900 - 0x0902
0x093A - 0x093A
0x0941 - 0x0948
0x094D - 0x094D
0x0953 - 0x0957
0x0962 - 0x0963
0xA8E0 - 0xA8F1
0xA8FF - 0xA8FF

Table 8.27: `dsrt` base characters

Character range	Description
0x10400 - 0x1044F	Deseret

Table 8.28: `ethi` base characters

Character range	Description
0x1200 - 0x137F	Ethiopic

Character range	Description
0x1380 - 0x139F	Ethiopic Supplement
0x2D80 - 0x2DDF	Ethiopic Extended
0xAB00 - 0xAB2F	Ethiopic Extended-A

Table 8.29: `ethi` non-base characters

Character range
0x135D - 0x135F

Table 8.30: `geor` base characters

Character range	Description
0x10D0 - 0x10FF	Georgian (Mkhedruli)
0x1C90 - 0x1CBF	Georgian Extended (Mtavruli)

Table 8.31: `geok` base characters

Character range	Description
0x10A0 - 0x10CD	Georgian (Asomtavruli)
0x2D00 - 0x2D2D	Georgian Supplement (Nuskhuri)

Table 8.32: `glag` base characters

Character range	Description
0x2C00 - 0x2C5F	Glagolitic
0x1E000 - 0x1E02F	Glagolitic Supplement

Table 8.33: `glag` non-base characters

Character range
0x1E000 - 0x1E02F

Table 8.34: `goth` base characters

Character range	Description
0x10330 - 0x1034F	Gothic

Table 8.35: grek base characters

Character range	Description
0x0370 - 0x03FF	Greek and Coptic
0x1F00 - 0x1FFF	Greek Extended

Table 8.36: grek non-base characters

Character range
0x037A - 0x037A
0x0384 - 0x0385
0x1FBD - 0x1FC1
0x1FCD - 0x1FCF
0x1FDD - 0x1FDF
0x1FED - 0x1FEF
0x1FFD - 0x1FFE

Table 8.37: gujr base characters

Character range	Description
0x0A80 - 0x0AFF	Gujarati

Table 8.38: gujr non-base characters

Character range
0x0A81 - 0x0A82
0x0ABC - 0x0ABC
0x0AC1 - 0x0AC8
0x0ACD - 0x0ACD
0x0AE2 - 0x0AE3
0x0AFA - 0x0AFF

Table 8.39: guru base characters

Character range	Description
0x0A00 - 0x0A7F	Gurmukhi

Table 8.40: `guru` non-base characters

Character range
0x0A01 - 0x0A02
0x0A3C - 0x0A3C
0x0A41 - 0x0A51
0x0A70 - 0x0A71
0x0A75 - 0x0A75

Table 8.41: `hebr` base characters

Character range	Description
0x0590 - 0x05FF	Hebrew
0xFB1D - 0xFB4F	Alphab. Present. Forms (Hebrew)

Table 8.42: `hebr` non-base characters

Character range
0x0591 - 0x05BF
0x05C1 - 0x05C2
0x05C4 - 0x05C5
0x05C7 - 0x05C7
0xFB1E - 0xFB1E

Table 8.43: `hmnp` base characters

Character range	Description
0x1E100 - 0x1E14F	Nyiakeng Puachue Hmong

Table 8.44: `hmnp` non-base characters

Character range
0x1E130 - 0x1E13B

Table 8.45: `kali` base characters

Character range	Description
0xA900 - 0xA92F	Kayah Li

Table 8.46: `kali` non-base characters

Character range
0xA926 - 0xA92D

Table 8.47: `knda` base characters

Character range	Description
0x0C80 - 0x0CFF	Kannada

Table 8.48: `knda` non-base characters

Character range
0x0C81 - 0x0C81
0x0CBC - 0x0CBC
0x0CBF - 0x0CBF
0x0CC6 - 0x0CC6
0x0CCC - 0x0CCD
0x0CE2 - 0x0CE3

Table 8.49: `khmr` base characters

Character range	Description
0x1780 - 0x17FF	Khmer

Table 8.50: `khmr` non-base characters

Character range
0x17B7 - 0x17BD
0x17C6 - 0x17C6
0x17C9 - 0x17D3
0x17DD - 0x17DD

Table 8.51: `khms` base characters

Character range	Description
0x19E0 - 0x19FF	Khmer Symbols

Table 8.52: `lao` base characters

Character range	Description
0x0E80 - 0x0EFF	Lao

Table 8.53: `lao` non-base characters

Character range
0x0EB1 - 0x0EB1
0x0EB4 - 0x0EBC
0x0EC8 - 0x0ECD

Table 8.54: `latn` base characters

Character range	Description
0x0020 - 0x007F	Basic Latin (no control chars)
0x00A0 - 0x00A9	Latin-1 Supplement (no control chars)
0x00AB - 0x00B1	... continued
0x00B4 - 0x00B8	... continued
0x00BB - 0x00FF	... continued
0x0100 - 0x017F	Latin Extended-A
0x0180 - 0x024F	Latin Extended-B
0x0250 - 0x02AF	IPA Extensions
0x02B9 - 0x02DF	Spacing Modifier Letters
0x02E5 - 0x02FF	... continued
0x0300 - 0x036F	Combining Diacritical Marks
0x1AB0 - 0x1ABE	Combining Diacritical Marks Extended
0x1D00 - 0x1D2B	Phonetic Extensions
0x1D6B - 0x1D77	... continued
0x1D79 - 0x1D7F	... continued
0x1D80 - 0x1D9A	Phonetic Extensions Supplement
0x1DC0 - 0x1DFF	Combining Diacritical Marks Supplement
0x1E00 - 0x1EFF	Latin Extended Additional
0x2000 - 0x206F	General Punctuation
0x20A0 - 0x20B8	Currency Symbols ...
0x20BA - 0x20CF	... except new Rupee sign
0x2150 - 0x218F	Number Forms
0x2C60 - 0x2C7B	Latin Extended-C
0x2C7E - 0x2C7F	... continued
0x2E00 - 0x2E7F	Supplemental Punctuation
0xA720 - 0xA76F	Latin Extended-D
0xA771 - 0xA7F7	... continued
0xA7FA - 0xA7FF	... continued
0xAB30 - 0xAB5B	Latin Extended-E
0xAB60 - 0xAB6F	... continued

Character range	Description
0xFB00 - 0xFB06	Alphab. Present. Forms (Latin Ligs)
0x1D400 - 0x1D7FF	Mathematical Alphanumeric Symbols

Table 8.55: `latn` non-base characters

Character range
0x005E - 0x0060
0x007E - 0x007E
0x00A8 - 0x00A9
0x00AE - 0x00B0
0x00B4 - 0x00B4
0x00B8 - 0x00B8
0x00BC - 0x00BE
0x02B9 - 0x02DF
0x02E5 - 0x02FF
0x0300 - 0x036F
0x1AB0 - 0x1ABE
0x1DC0 - 0x1DFF
0x2017 - 0x2017
0x203E - 0x203E
0xA788 - 0xA788
0xA7F8 - 0xA7FA

Table 8.56: `latb` base characters

Character range	Description
0x1D62 - 0x1D6A	some small subscript letters
0x2080 - 0x209C	subscript digits and letters
0x2C7C - 0x2C7C	latin subscript small letter j

Table 8.57: `latp` base characters

Character range	Description
0x00AA - 0x00AA	feminine ordinal indicator
0x00B2 - 0x00B3	superscript two and three
0x00B9 - 0x00BA	superscript one, masc. ord. indic.
0x02B0 - 0x02B8	some latin superscript mod. letters
0x02E0 - 0x02E4	some IPA modifier letters
0x1D2C - 0x1D61	latin superscript modifier letters
0x1D78 - 0x1D78	modifier letter cyrillic en
0x1D9B - 0x1DBF	more modifier letters
0x2070 - 0x207F	superscript digits and letters

Character range	Description
0x2C7D - 0x2C7D	modifier letter capital v
0xA770 - 0xA770	modifier letter us
0xA7F8 - 0xA7F9	more modifier letters
0xAB5C - 0xAB5F	more modifier letters

Table 8.58: `lisu` base characters

Character range	Description
0xA4D0 - 0xA4FF	Lisu

Table 8.59: `mlym` base characters

Character range	Description
0x0D00 - 0x0D7F	Malayalam

Table 8.60: `mlym` non-base characters

Character range
0x0D00 - 0x0D01
0x0D3B - 0x0D3C
0x0D4D - 0x0D4E
0x0D62 - 0x0D63

Table 8.61: `medf` base characters

Character range	Description
0x16E40 - 0x16E9F	Medefaidrin

Table: `medf` non-base characters

Character range

Table 8.62: `mong` base characters

Character range	Description
0x1800 - 0x18AF	Mongolian
0x11660 - 0x1167F	Mongolian Supplement

Table 8.63: mong non-base characters

Character range
0x1885 - 0x1886
0x18A9 - 0x18A9

Table 8.64: mymr base characters

Character range	Description
0x1000 - 0x109F	Myanmar
0xA9E0 - 0xA9FF	Myanmar Extended-B
0xAA60 - 0xAA7F	Myanmar Extended-A

Table 8.65: mymr non-base characters

Character range
0x102D - 0x1030
0x1032 - 0x1037
0x103A - 0x103A
0x103D - 0x103E
0x1058 - 0x1059
0x105E - 0x1060
0x1071 - 0x1074
0x1082 - 0x1082
0x1085 - 0x1086
0x108D - 0x108D
0xA9E5 - 0xA9E5
0xAA7C - 0xAA7C

Table 8.66: nkoo base characters

Character range	Description
0x07C0 - 0x07FF	N'Ko

Table 8.67: nkoo non-base characters

Character range
0x07EB - 0x07F5
0x07FD - 0x07FD

Table 8.68: `olck` base characters

Character range	Description
0x1C50 - 0x1C7F	Ol Chiki

Table 8.69: `orkh` base characters

Character range	Description
0x10C00 - 0x10C4F	Old Turkic

Table 8.70: `osge` base characters

Character range	Description
0x104B0 - 0x104FF	Osage

Table 8.71: `osma` base characters

Character range	Description
0x10480 - 0x104AF	Osmanya

Table 8.72: `rohg` base characters

Character range	Description
0x10D00 - 0x10D3F	Hanifi Rohingya

Table: `rohg` non-base characters

Character range

Table 8.73: `saur` base characters

Character range	Description
0xA880 - 0xA8DF	Saurashtra

Table 8.74: `saur` non-base characters

Character range
0xA880 - 0xA881
0xA8B4 - 0xA8C5

Table 8.75: `shaw` base characters

Character range	Description
0x10450 - 0x1047F	Shavian

Table 8.76: `sinh` base characters

Character range	Description
0x0D80 - 0x0DFF	Sinhala

Table 8.77: `sinh` non-base characters

Character range
0x0DCA - 0x0DCA
0x0DD2 - 0x0DD6

Table 8.78: `sund` base characters

Character range	Description
0x1B80 - 0x1BBF	Sundanese
0x1CC0 - 0x1CCF	Sundanese Supplement

Table 8.79: `sund` non-base characters

Character range
0x1B80 - 0x1B82
0x1BA1 - 0x1BAD

Table 8.80: `taml` base characters

Character range	Description
0x0B80 - 0x0BFF	Tamil

Table 8.81: `taml` non-base characters

Character range
0x0B82 - 0x0B82
0x0BC0 - 0x0BC2
0x0BCD - 0x0BCD

Table 8.82: `tavt` base characters

Character range	Description
0xAA80 - 0xAADF	Tai Viet

Table 8.83: `tavt` non-base characters

Character range
0xAAB0 - 0xAAB0
0xAAB2 - 0xAAB4
0xAAB7 - 0xAAB8
0xAABE - 0xAABF
0xAAC1 - 0xAAC1

Table 8.84: `telu` base characters

Character range	Description
0x0C00 - 0x0C7F	Telugu

Table 8.85: `telu` non-base characters

Character range
0x0C00 - 0x0C00
0x0C04 - 0x0C04
0x0C3E - 0x0C40
0x0C46 - 0x0C56
0x0C62 - 0x0C63

Table 8.86: `thai` base characters

Character range	Description
0x0E00 - 0x0E7F	Thai

Table 8.87: `thai` non-base characters

Character range
0x0E31 - 0x0E31
0x0E34 - 0x0E3A
0x0E47 - 0x0E4E

Table 8.88: `tfng` base characters

Character range	Description
0x2D30 - 0x2D7F	Tifinagh

Table 8.89: `vaii` base characters

Character range	Description
0xA500 - 0xA63F	Vai

Table 8.90: `yezi` base characters

Character range	Description
0x10E80 - 0x10EBF	Yezidi

Table 8.91: `yezi` non-base characters

Character range
0x10EAB - 0x10EAC

9 History

9.1 Version 1.8.4 (2021-Aug-14)

- Support for Hanifi Rohingya, Nyiakeng Puachue Hmong, Medefaidrin, and Yezidi.

- Bug fix: Option combination `-f none -S` to suppress hinting of unassigned glyphs was broken.

- Parallel builds of the package are now fully supported.

- The `configure` option `--with-freetype-config` has been removed; similarly to HarfBuzz, the `pkg-config` interface is now used exclusively for finding FreeType.

9.2 Version 1.8.3 (2019-Apr-09)

- Support for Mongolian. The de-facto standard is to render the glyphs horizontally (left-to-right) line by line, then rotating each line clockwise by 90 degrees. This allows the usage of ttfautohint.

- Bug fix: Glyphs with complicated shapes or a large number of outlines could be distorted if displayed with FreeType (or stay unhinted if viewed on Windows).

9.3 Version 1.8.2 (2018-Jul-24)

- Support for Georgian Mtavruli.

- Bug fix: Compilation of ttfautohintGUI with homebrew on the Mac works again.

- Bug fix: Point deltas didn't work for composite glyphs.

9.4 Version 1.8.1 (2018-Jan-01)

- Bug fix: The `configure` script couldn't recognize Qt 5.10.x.

9.5 Version 1.8 (2017-Dec-31)

- A third, 'natural' stem width mode has been added: No adjustments to stem widths, discrete blue zone positioning. This is what FreeType uses for its 'light' (auto-)hinting mode.

- A new option `-a` has been implemented to select the stem width mode for the three rendering targets (grayscale, GDI ClearType, DW ClearType). This supersedes option `-g`, which is now deprecated.

- Stem widths for the hinting process can now be manually set using the new `width` keyword in a control instructions file. This is especially useful for extra-bold fonts, where the algorithmically derived values are often too large, causing the filling of 'eyes' in 'e' or 'a' glyphs at small sizes.

- libttfautohint gets now installed as a DLL or static library (or both, depending on the parameters of the `configure` script), together with its header files.

- Two functions `TTF_autohint_version` and `TTF_autohint_version_string` have been added to the library, which do the obvious.

- Two options to `TTF_autohint` have been added: `alloc-func` and `free-func`. These are necessary on some platforms if ttfautohint is compiled as a shared library, and the application uses a different runtime library. This can happen, for example, on the MS Windows platform if your program is written in Python and communicates via the 'ctypes' interface with the ttfautohint DLL.

- `TTF_autohint` options `{gray,gdi-cleartype,dw-cleartype}-strong-stem-width` are now superseded by `{gray,gdi-cleartype,dw-cleartype}`-stem-width-mode' to handle the new natural stem width mode also.

- Bug fix: Later control instructions now correctly overwrite earlier entries as documented.

9.6 Version 1.7 (2017-Aug-26)

- Support for Adlam, Avestan, Bamum, Buhid, Unified Canadian Syllabics, Carian, Chakma, Coptic, Cypriot, Deseret, Glagolitic, Gothic, Kayah Li, Lisu, N'Ko, Ol Chiki, Old Turkic, Osage, Osmanya, Saurashtra, Shavian, Sundanese, Tai Viet, Tifinagh, and Vai scripts.

  Note that the used blue zone characters might not be optimal. Please report any problems so that I can fix issues!

- Option `--fallback-stem-width` now works for non-symbol fonts also, as documented.

- TrueType bytecode generated by ttfautohint now correctly supports most recent MS interpreter versions.

- A bug with bytecode for composite glyphs with composite subglyphs is now fixed.

9.7 Version 1.6 (2016-Nov-27)

- A new option `--reference` (and `--reference-index` to select a font within a TTC) makes ttfautohint use the blue zones from another font. This helps synchronize the ascenders and descenders of font families.

- Support for Armenian, Cherokee, Ethiopic, Georgian, Gujarati, Gurmukhi, Kannada, Malayalam, Sinhala, and Tamil scripts.

- New option `--ttfa-info` to display a font's `TTFA` table (if present).

- Glyphs covered by the fallback script are now hinted by default. Previously, they were scaled only. The default fallback script `none` now implies hinting without script-specific blue zones (but still aligning stems to the grid if possible).

- The new option `--fallback-scaling` changes the behaviour of the fallback script from hinting to scaling (as implemented in previous versions).

- ttfautohint (but not ttfautohintGUI) now honours the SOURCE_DATE_EPOCH environment variable for reproducible builds. This corresponds to the new library option `epoch`.

- Bug fix: Allow dehinting of fonts that contain a `.ttfautohint` glyph.

9.8 Version 1.5 (2016-Jan-24)

- Support for Khmer, Myanmar, and Bengali scripts.

- Improved Devanagari hinting.

- ttfautohintGUI can now be compiled with Qt5.

- Bug fix: Too many delta control instructions for a single glyph caused a bytecode stack overflow, making the MS rasterizer ignore all hinting instructions for this glyph.

- Bug fix: Don't create multiple `TTFA` tables in font.

- Bug fix: Under certain circumstances, glyph indices used in Indic features were incorrectly assigned to the default script.

9.9 Version 1.4.1 (2015-Oct-17)

- A bug in handling control instruction files could cause severe glyph shape distortions of accent-like glyphs. All users should update.

9.10 Version 1.4 (2015-Oct-04)

- Support for Thai and Lao scripts.

- Support for the Arabic script.

- Better support for scripts that contain superscript-like and subscript-like glyphs, e.g., the International Phonetic Alphabet (IPA).

- Accents and other 'non-base' glyphs are now hinted without snapping to blue zones.

- A new control instruction syntax form was added to adjust the mapping between glyphs and styles. Right now, its usage is quite limited; a forthcoming version will give much more flexibility.

- The `touch` keyword in a control instructions file was buggy: If used for a point P at a ppem value s, it sometimes led to unwanted movements of P for ppem values unequal to s, thus causing outline distortions.

9.11 Version 1.3 (2015-Jan-06)

- Keywords in control instruction files can be more verbose to increase readability. You can now use `left`, `right`, `nodir`, `point`, `touch`, `xshift`, and `yshift` for `l`, `r`, `n`, `p`, `t`,`x`, andy`, respectively.

- A new control instruction keyword `touch` was added to apply delta instructions before the final IUP bytecode commands, also 'touching' the affected points (to use the TrueType instructions terminology). Such deltas *do* work even with ClearType if applied to the non-ClearType direction.

- Support for the Telugu script.

- The amount of information about ttfautohint and its parameters that gets added to the `name` table by default has been reduced. A new option `--detailed-info` restores the previous behaviour.

- ttfautohintGUI crashed if not used with a control instruction file.

- ttfautohintGUI now correctly switches to a horizontal two-column layout if the standard one-column layout would exceed the screen height.

- A new option `--family-suffix` makes it possible to append a suffix to a font's family name in the `name` table. This can be useful during the development process: It helps the operating system to simultaneously display several instances of a font that are processed with different ttfautohint parameters.

- The new library option `info-post-callback` helps in processing data from the `name` table.

9.12 Version 1.2 (2014-Oct-06)

- It is now possible to control the hinting process at a very low level using a 'control instructions' file. Right now, two modes are supported:

  – Adding segments and changing segment directions.

  – Applying delta exceptions. Note that this doesn't really work in ClearType.

  Please see the documentation for a description and a small tutorial.

  In the command-line front-end, use option `--control-file` to load such a file.

- Support for input file watching in ttfautohintGUI: If the 'Watch Input File' box is checked, the program automatically regenerates the output file as soon as the input font or control instructions file gets modified.

  The idea is to dock ttfautohintGUI to your favourite font and text editor instead of adding a cheap text editor to ttfautohintGUI itself.

- With the new option `--fallback-stem-width` it is now possible to set a default stem width for all scripts that lack proper standard characters in the font.

- Add alias `--adjust-subglyphs` for option `--pre-hinting` to better describe its functionality. The short form `-p` stays unchanged.

- New option `--ttfa-table` to add an SFNT table `TTFA` to the output font, containing an ASCII dump of all used ttfautohint parameters (including control instructions). Right now, this is mainly for archiving purposes. Forthcoming versions of ttfautohint will be able to re-use this data if a font gets re-processed.

- A harmless buglet was fixed that made the `glyf` table too large by one byte under some circumstances.

- A bug fix taken from FreeType, correcting a crash for unusual outlines.

- Better handling of TTC files.

- New library options `error-callback` and `error-callback-data` for improved diagnostics. [No change in the front-ends except better error messages.]

- Many other, minor fixes and improvements.

9.13 Version 1.1 (2014-May-05)

- Support for the Devanagari script.

- Bug fixes in the computation of strong stem widths. GDI hinting now gives much more consistent results.

- Better recognition of script coverage: ttfautohint now properly scans composite glyphs to cover components also.

- Improved glyph shape analysis: Non-flat local extrema are now recognized more reliably, and stem detection works better.

9.14 Version 1.00 (2014-Mar-20)

- Much less memory consumption while handling fonts with complicated glyphs.

- Option `-s` was partially broken.

9.15 Version 1.00rc1 (2014-Feb-07)

- OpenType feature support. ttfautohint now uses the HarfBuzz library to analyze data from the `GSUB` table. This allows the hinting of glyphs that don't have an entry in a font's `cmap` table, for example superscripts or small caps.

  Related to this, the new option `--default-script` controls the default (fallback) script used for OpenType features.

- More than a single standard character is used. For example, the 'latin' script uses characters 'o', 'O', and digit '0'. This improves the hinting of fonts (and features) that have only a partial coverage of a script's character repertoire.

- Much better GDI ClearType hinting in the range 30-80ppem (approx.), avoiding overly flat tops and bottoms of round glyphs.

- Better handling of non-square pixels (this is, horizontal and vertical resolutions differ) in the created TrueType bytecode.

9.16 Version 0.97 (2013-Nov-09)

- Improved script support. Besides Cyrillic and Greek, which are now handled separately from Latin, ttfautohint can handle Hebrew.

- Option `-f` now takes a parameter to specify the fallback script. The corresponding long option name has been renamed from `--latin-fallback` to `--fallback-script`.

- Work around a bug in display environments that use FreeType 2.5.0 and earlier for rendering: Sometimes, the 'strong' stem width routine was used for DW ClearType (this is, subpixel hinting in FreeType is enabled) even if 'smooth' was selected while generating the font with ttfautohint.

9.17 Version 0.96 (2013-Aug-06)

- Option `--components` has been replaced with `--composites`: By default, the components of a composite glyph are now hinted separately, since tests has shown that this gives good results in most cases. If this option is set, however, the composite glyph itself gets hinted (and the hints of the components are ignored).

  An unfortunate side effect is that ttfautohint's option `-c` (which stays as a shorthand for `--composites`) now does exactly the opposite as in previous releases.

- Older versions of Monotype's 'iType' bytecode interpreter have a serious bug: The DIV instruction rounds the result, while the correct operation is truncation. This caused 'exploding characters' with fonts hinted by ttfautohint. Since many printers contain this rasterizer without any possibility to update to a non-buggy version, ttfautohint now contains work-arounds to circumvent the problem.

- Better support for glyphs where some points have almost the same position (for example glyph 'Oslash' in font 'Roboto-Thin').

- Better support for glyphs that use explicit 'on' points around round extrema.

9.18 Version 0.95 (2013-Mar-07)

- New option `--dehint` to strip off all hints without generating new hints. This option is intended for testing purposes.

- Minor fixes to the created bytecode for compatibility.

- Minor GUI improvements.

9.19 Version 0.94 (2012-Nov-29)

- New option `--windows-compatibility` that adds two artificial blue zones at vertical positions given by 'usWinAscent' and 'usWinDescent'. This helps ttfautohint's hinting algorithm reduce the possibility of clipping if those two values are very tight.

- Implement option `--x-height-snapping-exceptions`, making ttfautohint avoid x-height snapping for selected PPEM values. Useful in combination with `--windows-compatibility`.

- Minor fixes to the created bytecode for compatibility and robustness.

9.20 Version 0.93 (2012-Oct-09)

- New option `--components` to treat components of composite glyphs separately. This greatly reduces the bytecode size.

  I'm waiting for reports whether this option works for most fonts; in case this is true I'm inverting the option, making it the default (and the old behaviour optional).

- Full support of TTCs, this is, all subfonts get auto-hinted now.

- The upper limit of the `--increase-x-height` option has been removed.

- Drag-and-drop support in the GUI.

- The command-line version of ttfautohint now acts like a (Unix) filter, this is, it accepts stdin and stdout as input and output, respectively.

- Less memory consumption.

9.21 Version 0.92 (2012-Aug-07)

- A serious bug in the created bytecode has been fixed, causing incorrect rounding.

9.22 Version 0.91 (2012-Jul-12)

- A new, 'strong' routine to handle stem widths and positions has been added, to be selected with the `--strong-stem-width` command line option. If it is active, stem widths and positions are snapped to the grid as much as possible. This algorithm is useful for GDI ClearType support.

- A new command line option `--debug` (not available for ttfautohintGUI) to print very detailed debugging information.

9.23 Version 0.9 (2012-Jun-06)

- The created bytecode has been reduced in size, making it approx. 20% smaller.

- New option `--symbol` to use standard stem height and width values instead of using character 'o' (which may be missing). Use this option for symbol fonts or math glyphs.

- More documentation (in text, HTML, and PDF format). It's still incomplete, though.

- Option `--ignore-permissions` has been renamed to `--ignore-restrictions`. The short form is still `-i`.

- Defaults for various parameters have been set to more sensible values:

– hinting-range-max: 50 (was 1000)
– hinting-limit: 200 (was 1000)

- Option `--increase-x-height` now has a mandatory argument (in the range 6-20 or value 0 to disable it, default value is 14).

9.24 Version 0.8 (2012-Mar-21)

- Implement option `-x` to increase the x height of the font for small PPEM values by rounding up far more often then rounding down.

- Add option '`-G n`' to switch off hinting completely above value `n`.

- ttfautohint now appends version information and the used parameters to the 'Version' field(s) in the 'name' table. This can be suppressed with option `-n`.

9.25 Version 0.7 (2012-Feb-05)

- A GUI has been added, using the Qt framework. The binary is called 'ttfautohintGUI'.

9.26 Version 0.6.1 (2012-Jan-02)

- The improved handling of composite glyphs in 0.6 was buggy under certain circumstances, making ttfautohint crash and FontValidator complain.

- Dropout handling has been activated.

9.27 Version 0.6 (2011-Dec-25)

- Improved handling of composite glyphs.

- Implement option `-p` to pre-hint glyphs with original hints before conversion takes place.

- Don't add a `DSIG` table if there is none in the input font.

- Human-readable error messages instead of hexadecimal error codes.

- Better tests (both at runtime and compile time) to reject too old FreeType versions.

9.28 Version 0.5 (2011-Nov-06)

- Rendering on iOS is now expected to give good results.

- No bad rendering at very large PPEM values.

9.29 Version 0.4 (2011-Oct-27)

- The bytecode has been changed to 'create' twilight points. This should avoid rendering artifacts on some platforms.

9.30 Version 0.3 (2011-Sep-09)

- Fix font generation; sometimes the `glyf` table was one byte too short, making the font invalid.

9.31 Version 0.2 (2011-Jul-19)

- Fix bytecode bugs that prevented correct rendering on some platforms.

9.32 Version 0.1 (2011-Jun-30)

- First release.