Practical TrueType Hinting

For use on low-resolution displays at small point sizes, the font rasterizer needs hints to produce a pleasing result. Low-resolution devices are monitors and printers with a resolution below 600 dpi in the eyes of fontographers. The combination of low resolution and small point size results in a small number of pixels per character. Since the size of the character grid is called 'Em', the size of the character is expressed in PPM (Pixels Per eM) when we set up the hinting in a font.

TrueType and Type1

The two main types of fonts – Type1 and TrueType – both use a different kind of hinting mechanism. In this article we'll take a look at TrueType hinting from a practical point of view. The reason for this choice is that TrueType hinting is far more complex than Type1 hinting and TrueType fonts are most of the time used in web pages and as display fonts. With the arrival of embedded font tools (such as the free WEFT by Microsoft), the use of other fonts than Arial, Verdana, Comic Sans, etc. has become much easier for the web designer. Other fonts however, often lack hinting programs needed to produce legible results in web pages.

The tools

Fontographer 4+ and Fontlab 3+ supply the tools needed to produce hinting programs. Surprisingly the tools available in these programs are quite different. Both have the usual align and link tools in their repertoire. Fontographer includes tools for making serif hints and diagonal hints, and Fontlab has chosen to include several other links and various delta-hinting tools. The reason is simply that TrueType hinting consists of a rather low-level language. The font editors include a set of higher level, visual hinting tools and produce the actual hinting program from the tools that the user applied to the glyphs.
There is also a visual hinting program available from Microsoft for free to those who apply for this tool.
In this tutorial we'll start by taking a look at the tools of Fontlab 3.1, simply because the delta-hinting plays a vital part in producing legible text in small sizes. The name Fontlab is abbreviated in the text as FL.

Theory

On the Microsoft website the TrueType standard and several articles about hinting instructions are available. This article will take you through the steps to produce a hinted font without explaining too much about the theory behind it.
A few abbreviations and terms from the typographer's lingo are used in the text. While most users of font designer programs are familiar with these words a small glossary of terms is included at the end of the tutorial.

Hinting is a program

Hints in TrueType fonts are implemented as interpreted programs. The programs are executed on the three levels mentioned in the next paragraph and the results are cached on each level. The language contains quite a collection of instructions, including instructions to access constants and variables, conditional jumps, etc. There are a dozen or so professional hinters in

the world who actually write hinting-programs using the instructions of the language. Luckily the visual tools that are available supply us with a nice collection of high level instructions.
Hinting is also called 'grid-fitting', because what we do is actually telling the font rasterizer how it should fit the outlines to the pixel grid. People often wonder why the rasterizer doesn't know how to do this. The reason is simply that the rasterizer doesn't see stems, bowls and shapes in general, but only a collection of points and how these points must be connected. Furthermore, displaying a glyph at a small size automatically leads to trade-offs between size, shape and proportion. The resulting bitmap should be legible, but also have the look and feel of the original shape. The TrueType rasterizer simply turns on the pixels that are inside the outline after scaling it down to the desired size. Hinting programs allow us to change this shape under certain circumstances and thus influence the resulting bitmap.

Top to bottom

TrueType hinting is active on three levels: *global* – for the entire font in all PPM sizes, *glyph* – per glyph for all PPM sizes, and *delta* – for one glyph in one (or a range of) PPM size(s).
Let's take a look the tools that are available and start with the highest level – the global hinting tools.

Global hinting

In this area of hinting there are two tools available. These tools apply to all glyphs at all sizes.

Alignment zones

Alignment zones define the areas in which an overshoot or undershoot glyph can snap to the gridline of a non-overshoot (or non-undershoot) character. In many fonts the bottom of the 'o' is a bit below the bottom of the 'x' and the top of the 'o' is a bit above the top of the 'x'. This way the size of the 'o' *appears* to be equal to the size of the 'x'. At low PPM sizes the overshoot can become exaggerated. Alignment zones can correct this by e.g. snapping the overshoot to the top of the 'x' when it comes within the alignment zone.
Primary alignment zones can hold one bottom alignment zone (baseline zone) and up to six top alignment zones. Secondary alignment zones can hold up to five bottom alignment zones.
Fontlab can automatically calculate the *primary* alignment zones by looking at several glyph pairs; e.g. 'o' versus 'x', 'O' versus 'H' and 'p' versus 'g'. If the preview of the font still shows exaggerated undershoots in several characters, one can easily add secondary alignment zones on a trial and error basis.

Standard stem widths

Many fonts show at some point sizes the effect that stems of certain characters are one pixel wide and stem of other characters are two pixels wide. This is due to rounding problems.
If for example one stem is 152 units (on a TT grid of 2048 UPM (Units Per eM)) and another stem is 156 units wide, displaying the font at 20 PPM would result in the first stem being 1 pixel (1.48) wide and the second stem being 2 pixels (1.52) wide. The difference of 1% resulted in 100% on the display.
In the stem widths dialog (*File, Font Info, Alignment*) you can enter one standard stem width and up to ten snap values for both the vertical and horizontal stems. TrueType hinting only uses the standard stem widths. The *glyph hints* that are discussed in the next chapter can be linked to these stem widths. This way we can make all vertical stems appear to be the same width, even though the outline tells us different.
Note: the automatic calculation of stem width in Fontlab 3.1 with a TT font does not work. See Stem widths *for instructions on how to manually set these values.*

Glyph hints

After running the global hinting program, the rasterizer starts to work on individual glyphs. Although the hints in this category are applied to a single point, they tend to affect the entire outline the point belongs to; mostly because of links that are formed with other points by other hints on this level.
The align- and link commands in this category either work horizontally or vertically. You can however apply both a horizontal and a vertical command to the same point.

Align commands

The align-button in the toolbar has actually two functions. If the point is within the top or bottom alignment zone it will apply the AlignTop or AlignBottom command to that point (unless you use the SHIFT-key while clicking the point). These are useful for characters that jump above the x-height or below the baseline for example.
In all other cases the alignment is to a nearby gridline. The following commands are available from the option panel:

- Round to closest grid line (code 0). Appears as a double headed arrow with the arrows pointing outward and a green mark at the end of the arrow. This will align the point with the nearest grid line.
- Round to left or bottom line (code 1). Appears as a single headed arrow, arrow pointed outward and a green mark and the end.
- Round to right or top grid line (code 2). Appears as code 1.
- Round to centre (code 3). Appears as a double headed arrow with arrows pointing inward and a green mark at both ends. Aligns the point with the gridline closest to the centre of the pixel.

The commands appear in the program panel as `AlignV 5 [1]`; Align<direction> <point number> <code>. You can change rounding code and convert an AlignV/H command to AlignTop/Bottom (if the point is inside or close to an alignment zone) in the context menu of the point.
These align commands are useful if a glyph is always or often higher or lower than the other glyphs in the font. You can now correct the glyph in a lot of PPM sizes at the same time. If one or two sizes do not work correctly, use the *Middle Delta* commands we'll discuss later.

Single link

The single link will keep the linked point at the same distance from the base point. You apply the single link by clicking the single link button and then dragging the base point to the linked point. The single link will either work horizontally or vertically.
But things get complicated and ugly here:

- Links may be attached to a stem width from the stem width table (which we defined globally). Then the width of the link will be made equal to the stem width value it's attached to. If the link is not attached to a stem width it may or may not be rounded to the grid, depending on whether the *Round distance* option was activated (the option is recommended to be on and is on by default).
- The base point can/should be aligned by an align command. The linked point will then move with the base point. This is actually the point of having single links: link a point on the outside of the top of the 'o' to the point on the inside and align the top of the 'o' with the top alignment zone. This way the inside of the character will move together with the outside.
  FL considers a link whose base point is not aligned or linked as an error and displays a red arrow near the link to indicate this.

- Normally the linked point cannot be aligned anymore, except when the single link is an *aligned single link*. By selecting the *Round destination* option and a link type (see section above on *Align* commands) you can make an aligned single link.

Links appear in the program panel as `SingleLinkH 1 -> 23 [0] 2`; SingleLink<direction> <base point> -> <linked point> [<stemnumber> | ns] <aligncode> | na.

Double link

This link will control only the distance between two points. It may be attached to a stem width, so the distance will be equal to the stem width. You cannot control in which direction the linked points are moved.
The use of this link is limited to stems of which the position does not matter. Using one of the points of a double link as the destination point of a single link will cause FL to display a red arrow as in indication of an error. You can however use one or both points of a double link as the base point for single links.
Double links appear in the program panel as `DoubleLinkV 2 <-> 12 [1]`; DoubleLink<direction> <point#> <-> <point#> [<stem#> | ns].

Interpolation

Some areas in glyph one point should be in the middle of two other points (in some fonts in the 'v', 'w', '3', 'B', etc.). The interpolation command can link that point to the other two points. Simply drag the first base point to the linked point and then drag the linked point to the second base point.
The linked point can also be aligned to the grid by selecting the *Round destination* option and selecting an align type (see *Single Link*).

Delta links

Delta links apply to one glyph and one PPM size only. They are useful for the little changes that need to be made to make the bitmap perfect. Again there are two types of delta hints.

Middle Delta commands

These are inserted in the hinting program and will often affect more than one point. But since they only work in one PPM size, the effect is limited. The commands are useful to correct the results of links and align commands for only a few sizes.
Simply activate the *Middle Delta* button and drag the point in the direction you want. Dragging the point again in a different direction easily changes the command. The command works in steps of ⅛ of a pixel. FL automatically combines similar delta instructions for different PPM sizes into one single command. FL also splits a non-orthogonal delta hint in two separate hints.

> TIP: To remove a combined delta instruction for one size, simply change the distance for this size and then the combined instructions are automatically split. Then the delta command can be removed from the program panel.

The command appears in the program panel as `MDeltaH 12 <9> 9-10`; MDelta<direction> <point#> <<distance>> <PPM0>-<PPM1>.
Note: Sometimes FL3.1 does not seem to split the combined delta hints (same hint for more than one PPM size) if you want to change them. Deleting the delta hint results in the deletion of the entire command (thus deleting the hint for more than one PPM size). If the combined hint was part of a non-orthogonal hint, only the hint for one direction (either horizontal or vertical) is deleted. The solution seems to be to write down the direction, length and PPM size(s) for that hint, delete it, apply the correct hint for the current size and then re-apply the hint for the other size(s).

Final Delta command

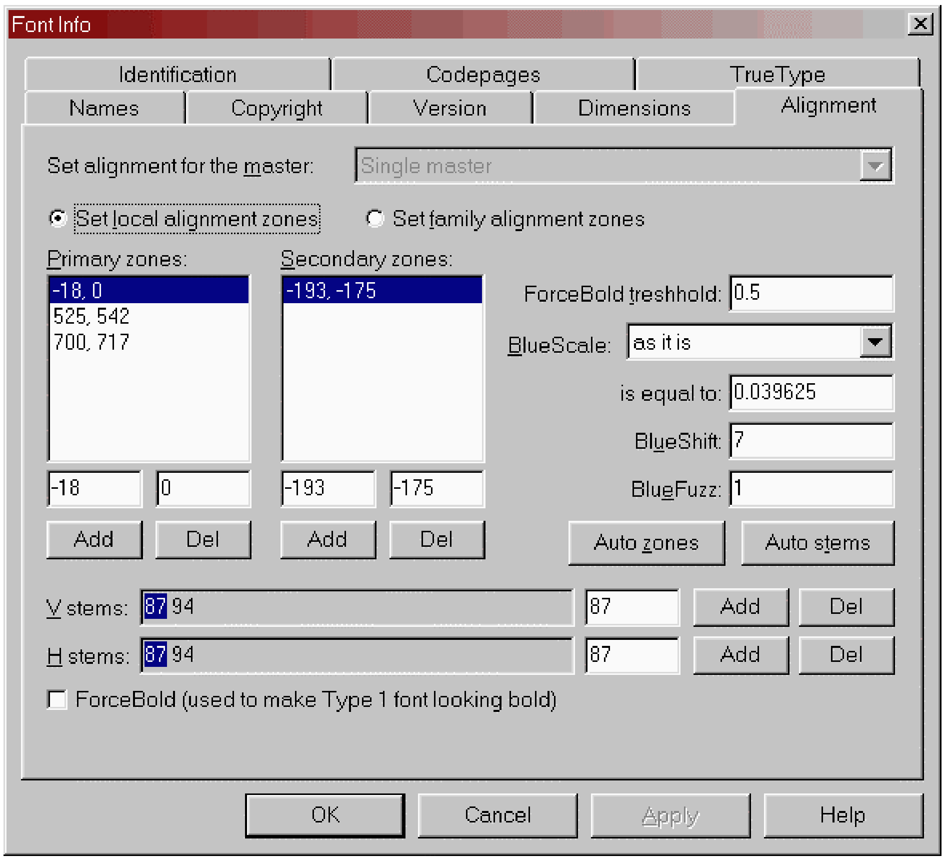
After the final interpolation of all untouched points the final delta instructions kick in. They only affect one single point of the interpolated outline. After achieving the best possible result with the middle delta instructions you can do the last pixel moving with the final delta instructions.
The final delta instructions have a similar appearance in the program panel.

What shall we do with all these hinting tools?

Use a practical approach! Work from global settings to fine detail. So…
In the *Font Info* dialog FL can easily calculate the alignment zones by itself. Simply click on the *Auto zones* button.

Stem widths

Strangely enough the automatic stem width calculation does not work with TrueType fonts. We have to do a little manual labour. Enter the measure mode of FL and note the widths (and heights) of vertical stems, horizontal stems and the horizontal and vertical size of the diagonal stems (in glyphs such as "aksvwxyzAKSVWXYZ"). In *File, Font Info, Alignment* you can simply add these numbers. Start with the standard width of the vertical and horizontal stems.


Font Info

Tabs: Identification, Codepages, TrueType, Names, Copyright, Version, Dimensions, Alignment

Set alignment for the master: Single master

[✓ Set local alignment zones] Set family alignment zones

Primary zones:
-18, 0
525, 542
700, 717

-18 | 0

Secondary zones:
-193, -175

-193 | -175

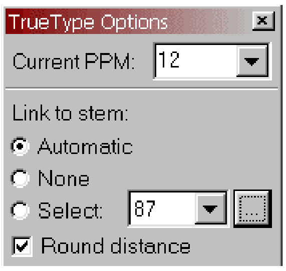
ForceBold treshhold: 0.5
BlueScale: as it is
is equal to: 0.039625
BlueShift: 7
BlueFuzz: 1

V stems: 87 94 | 87
H stems: 87 94 | 87

☐ ForceBold (used to make Type 1 font looking bold)

In the illustration FL calculated the alignment zones by hitting the *Auto zones* button. After measuring the stem widths the widths for vertical and horizontal stems (87) and the width for diagonal stems (94) were added to the list. Simply click on the *Add* button and type the number in the text box on the left of this button. Click on *Add* again to enter another stem size. FL (and we) will use these values to attach links to; this will take care of the problem of certain stems looking thicker than others at certain sizes.

Stem width to pixel size


TrueType Options

Current PPM: 12

Link to stem:

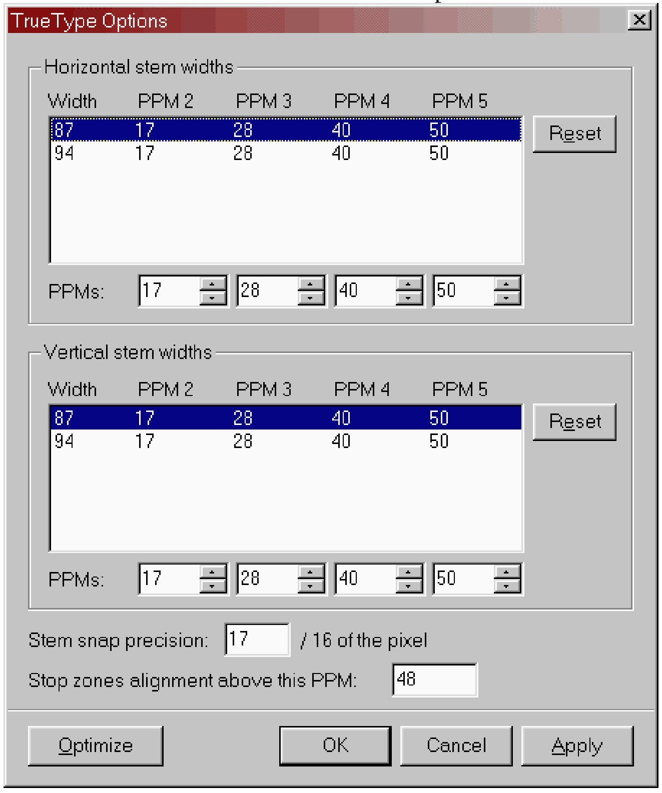
[✓ Automatic]

None

Select: 87

[✓ Round distance]

It's now time to link stem widths to pixel sizes. In other words: at which PPM size should a stem be displayed as 1, 2, 3, 4 or 5 pixels? Open an edit window for a glyph and activate the TrueType hinting mode. On the panel *TrueType Options* hit the button with the ellipsis [...] next to the dropdown button in the *Link to stem:*-section. Another window with the same title opens, in which we can set these sizes.


TrueType Options

Horizontal stem widths

Width	PPM 2	PPM 3	PPM 4	PPM 5
87	17	28	40	50
94	17	28	40	50

PPMs: 17 28 40 50

Vertical stem widths

Width	PPM 2	PPM 3	PPM 4	PPM 5
87	17	28	40	50
94	17	28	40	50

PPMs: 17 28 40 50

Stem snap precision: 17 / 16 of the pixel

Stop zones alignment above this PPM: 48

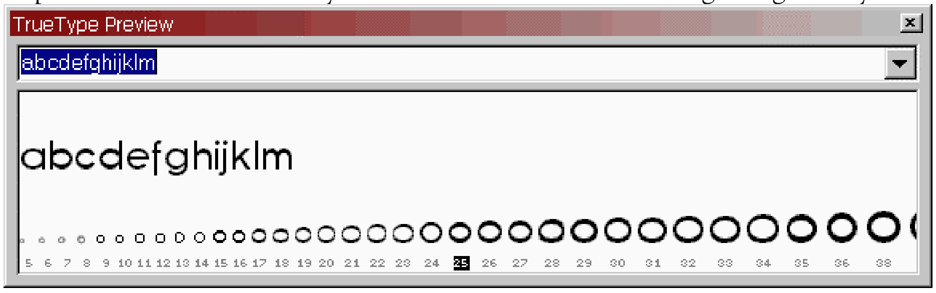
The *Optimize* button on the bottom really works well. You can easily change the suggestions that FL made by clicking on a row and changing the values in the boxes below.

If you have a whole family of fonts, it might be a good idea to make sure the values you set here work well together. A bold face should always have thicker stems than a normal/regular/medium face; otherwise nobody would see it as bold. This is especially important with smaller sizes.

Also remember to check whether the horizontal and vertical stem widths settings work well together. In a sans serif font with equal stem widths for vertical and horizontal stems it would look very wrong if the same width would be displayed as two pixels for vertical stems and one pixel for horizontal stems. We identified this effect as one of the problems we tried to solve by applying hints, and we certainly wouldn't want to introduce it here instead!

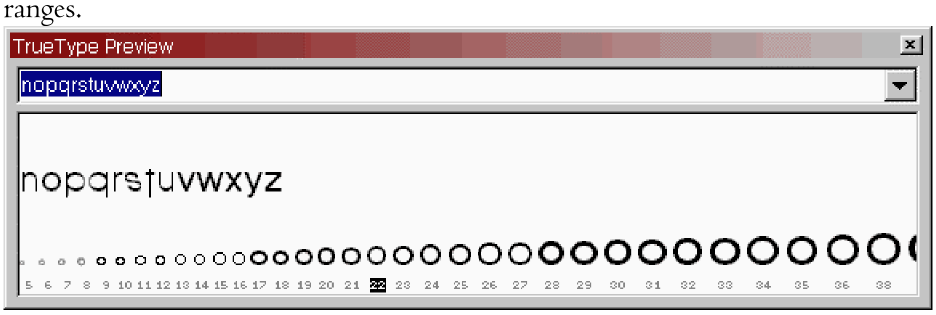
Autohinting

After that we'll let the font program do a little auto hinting. This will result in some improvement. After a while you will often use manual hinting straight away.



This is how the 'o' looked at 5-38 PPM before the auto-hinting was applied (font smoothing was left on). This glyph really suffers from the stem width problem described above. There is hardly any size where the vertical and horizontal parts of the letter are equal in size. Measuring the distances in the outlines themselves results in identical widths. Rounding the nodes to the grid has resulted in very odd shapes.

Note that the settings in *File, Font Info, TrueType* were that for size 0-8 only smoothing and no gridfitting was to be used. Therefore hints have no effect on the sizes 5-8 in this window. If you want to make a font that is useful at very small sizes you will first have to set the correct PPM-ranges.



After applying auto-hinting to this glyph it looks much better. The stem width problem is solved.

What was the right solution?

First of all, in the dark art of TrueType hinting *the* right solution doesn't exist. Once you get the hang of using the hinting tools you'll use your previous experiences to try and predict what certain combinations will do, but there is often more than one way to improve the looks of a glyph.

You can see what auto-hinting did for you by right clicking on the work area and selecting *convert hinting to instructions*. Now the results of the auto hinting become visible.

The shape of this glyph is rather simple, so there aren't too many possibilities. Let's see one possible solution:

```
AlignH 32 [1]
```

Align the base point of the sidebearing on the left (point number 32) to the gridline on the left (code 1).

```
AlignH 33 [0]
```

Align the sidebearing on the right to the nearest gridline (code 0).

These instructions are mostly used to provide a base for the single links (single links need an aligned base point). In this case no links were attached.

```
DoubleLinkH 15 <-> 0 [0]
DoubleLinkH 7 <-> 23 [0]
```

These links attach stem width number 0 (see the list of stem widths on the *Alignment* tab of the *Font Info* dialog). This way we make sure that the thickness of the shape in this area will do what we want by setting a stem width and linking it to certain pixel sizes at certain PPM sizes, as we did earlier.

```
AlignTop 4 [0]
AlignBottom 11 [0]
```

These align the top and bottom of the glyph to the alignment zone. This makes sure that the glyph will not stick out with small sizes.

```
SingleLinkV 11 -> 19 [0] na
SingleLinkV 4 -> 28 [0] na
```

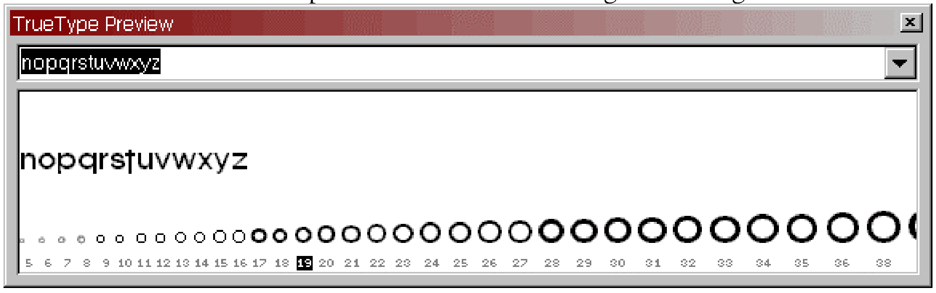
Since we aligned the top and bottom with the alignment zones, these single links can be attached to stem width 0. Now the thickness of the vertical and horizontal parts is linked with one stem width. This ensures us that the problem of changing thickness is solved once and for all.

... and the O goes round and round...

After the auto-hinting kicked in we're left with the problem of the glyph being not completely round at some sizes (most notably 9, 10, 11, 17, 18, 19, 21, 23, 29, 30, 33, 34, ...). Eleven glyphs seem to be a lot, but we can actually fix this by a couple of middle delta hints per size. Let's see what it takes for sizes 9-19.

```
MdeltaH 0 <-4> 9-10
MdeltaH 0 <2> 11-11
MdeltaH 0 <-4> 17-17
MdeltaH 0 <-5> 19-19
MdeltaH 0 <-7> 18-18
MdeltaH 7 <5> 11-11
MdeltaH 7 <5> 17-17
MdeltaH 15 <-5> 17-17
MdeltaH 15 <5> 11-11
MdeltaH 15 <3> 12-12
MdeltaH 15 <-6> 18-19
MdeltaH 15 <-6> 10-10
MdeltaH 23 <4> 11-11
MdeltaH 23 <6> 17-17
MdeltaH 23 <-2> 9-10
MdeltaH 23 <-4> 12-12
```

The sizes 9, 10, 11 and 12 required a little more work. Remarkably, the bitmaps for these sizes looked right before the auto-hinting was applied. The actual work involved in fixing these four sizes was not much. A couple of middle delta hints gave us a legible character.



What's left now is making the sizes 23 and up a bit more round.

Is auto-hinting useful for me?

A lot of people complain about the auto-hinting features not doing a very good job. In general this is right; auto-hinting will not result in a perfect font. But it *will* give you a starting point for your own hinting instructions. Sometimes it will suggest very odd constructions that actually work well with a number of sizes.
It's also useful to see what the removal of one link or changing a rounding option will do. Using the preview window and scrolling through the different PPM sizes you can see where problems arise.

The problems with auto-hinting

A few problems are not solved by auto-hinting. In many cases the algorithm doesn't detect characteristic features of the shape. One characteristic of the font used here as an example – Florencesans – is that the round parts of e.g. the 'a' and 'p' meet the vertical stem in a sharp angle. The white area between the round part and the stem adds to the look and feel of this font. Using one AlignTop and one SingleLinkV in that area makes sure that with small sizes the white area will be present.
Another problem that occurs is incorrectly attaching single links to stem widths. The algorithm likes to attach links to stem widths. In some cases this is incorrect. You can solve this by removing the attachment to the stem width (right click on the link and deselect the option) or by adding a stem width to the list we've defined and attaching the link to that width.

Hinting strategies

As we've seen, the most practical way is to use auto-hinting and fix the remaining problems with delta hints. It is interesting to see how we can do it all by ourselves. It's hard and it requires a lot of experience or a lot of trial and error. The more complex a shape becomes the harder it gets for both the auto-hinting algorithm and us mortals. Glyphs such as 'a', 'g', 's' will form the biggest problem in sans serif fonts. Also the 'x', 'B', 'E' will provide you with the challenge of placing the middle part at various PPM sizes.

Trading off between size and proportion

Most of the time we have to decide between the absolute size of a part of the glyph and the relative size (compared to the entire glyph and to other parts of the glyph). The serifs of many characters are a good example. Serifs are an important part of the look and feel. If the outline is scaled down the serifs are often so small that you should remove them completely. This changes the character of the font too much, so we decide to make them a lot larger.
The same decision must be made for characters such as the 'E'. If the entire glyph is an even number of pixels high the horizontal stem in the middle cannot be placed in the centre. Using alignments and links you can decide whether it should be above or below the centre. Another example of this dilemma is the 'm'. If you link all the vertical stems and the spaces between them from left to right you will end up with the correct proportions. At small sizes the total width of the character will most likely be larger than intended. You decide what to do.

The rasterizer is stupid

This is an important thing to remember. If you place a link between the outside and the inside of a 'B' to fix the vertical stem, you have to place two links: one to the upper and one to the lower 'hole'. The rasterizer does not see shapes; everything is a collection of points.
Even worse: it can easily ruin the thickness of the cups of the 'B' in this case. It simply moved the two holes to make the vertical stem right. You also have to apply links to fix the thickness of the two cups.

Identify key points

The rasterizer will move unaffected points to approximate the original shape and proportions of an outline as good as possible. Your efforts should be focused on aligning and linking the extreme points and other important points.
So, start by aligning the right and left sidebearing and the top and bottom points of the outline to the alignment zones. This will give you enough points as base points for single links.
After that decide where the white space in the grid has to come. Rounding can completely remove white space. Applying enough single links can even make the bitmap wider than the width of the glyph. One single link (not attached to a stem width!) can fix the white space.
Next we provide links to correct the stems and distances that should be the size of a stem width.
The last things to do will be the serifs (size and shape!), interpolation (in sans serif 'W', the middle stem of the 'B', etc.) and fixing other things that don't look good with a large number of sizes.
Remember it's better to apply a few hints that will solve things in 80% of the sizes than to try and fix everything with delta hints.

Which sizes should be hinted?

In the *Font Info* dialog, the *TrueType* tab lets you set which font sizes should be displayed using what algorithms. The settings S(moothing) and G(ridfitting) are available here. The TrueType standard recommends that smoothing is set for sizes 0-8, gridfitting for sizes 9-17, and both smoothing and gridfitting for sizes 18 and larger. This means that sizes 9-17 will not be smoothed. This results in more legible fonts – if they are well hinted – although customers seems to expect that turning font smoothing on results in smoothed fonts at all point sizes.

If these standard settings are used you should concentrate your efforts with delta hinting on sizes 9-17.
Also keep in mind that hints only make a difference of one pixel. Changing the height of a character with one pixel has more impact on small font sizes and less on larger font sizes. You will use more delta hinting at smaller sizes and let the links, etc. do the work at larger sizes. Turn of font smoothing in Windows to see the results of hinting in larger font sizes. If the non-smoothed character looks good, the smoothed version will be good also. It's easier to see problems in non-smoothed characters however.
Note: The Visual TrueType tool from Microsoft includes the possibility to have separate hinting for smoothed text and for non-smoothed text.

The limitations

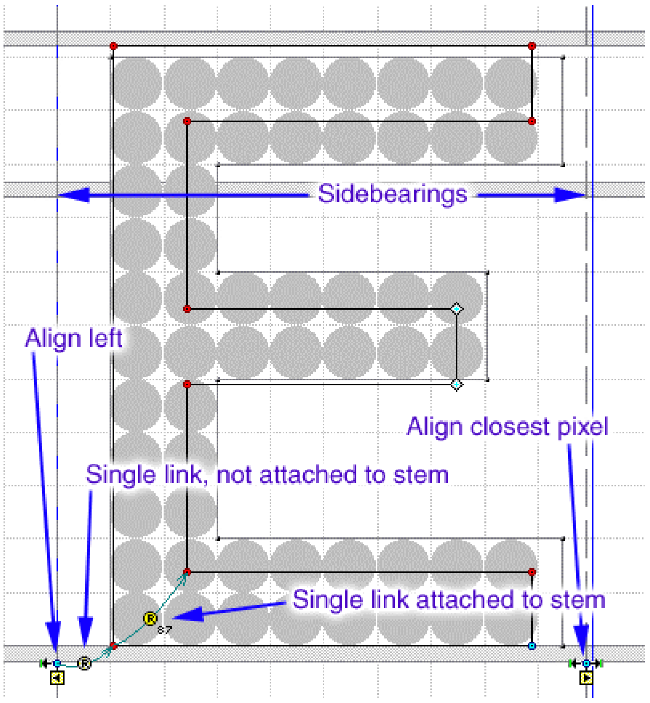
Fontlab provides us with a useful, but limited set of high level hinting tools. This takes away the problems of combining delta hints, setting the right order, combining the instructions into useful tools, etc. On the other hand, a lot of the power has disappeared. TrueType hinting allows the font to make decisions based on various conditions, including the PPM size, etc. An example is a font with ornamented caps where the ornaments are removed for small sizes.
In most cases you can dramatically improve the quality of the font for use on screen by using a rather limited number of hints.

Real world examples

Let's take a look at a few glyphs. We'll start with a couple of easy ones, and as we learn we'll take on the challenge of more complex situations. Always remember that these examples seemed to work for me in this particular font. Others may have found better solutions or more elegant ways.

The E

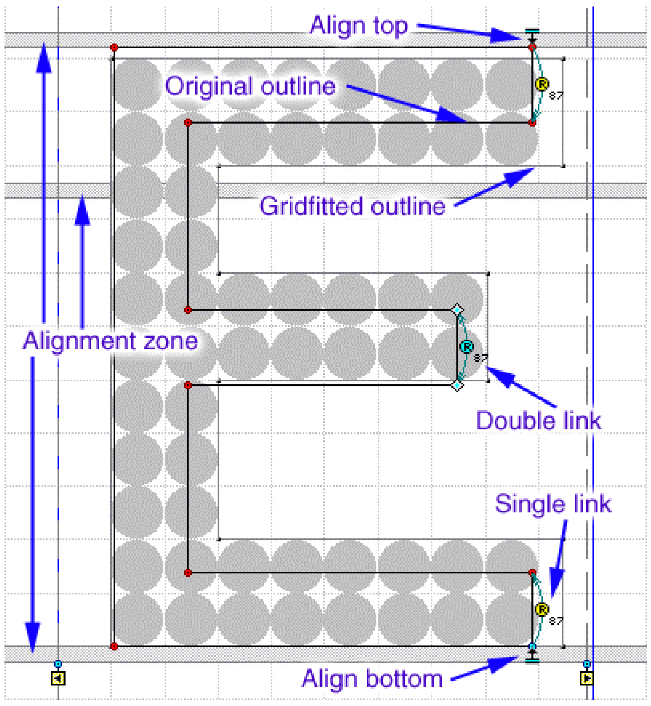
Since hints are either applied horizontally or vertically we have to decide where to start. Due to some inexplicable twist of my brain, I'd like to start with the horizontal hints.


Sidebearings
Align left
Align closest pixel
Single link, not attached to stem
Single link attached to stem

First I align the base points of the left and right sidebearing. In most cases an Align left for the left one and an Align to closest pixel for the right one.
For this font I decided (usually the designer should make the decisions) to have a little white space on the left side of the glyphs. This way I can work from left to right, but the other way around should be just as easy.
We use the left base point as the base point for a single link to the left side of the vertical stem. Do not align this link with a stem width, simply because it isn't a stem and shouldn't always be as wide as a stem. I used the bottom point of the vertical stem, simply because I'm lazy and the top

point was too far away. From the left side of the vertical stem I made a link to the right side and attached this link to the appropriate stem width. This way we make sure that the stem is as wide as all other corresponding stem in all sizes.

Vertical



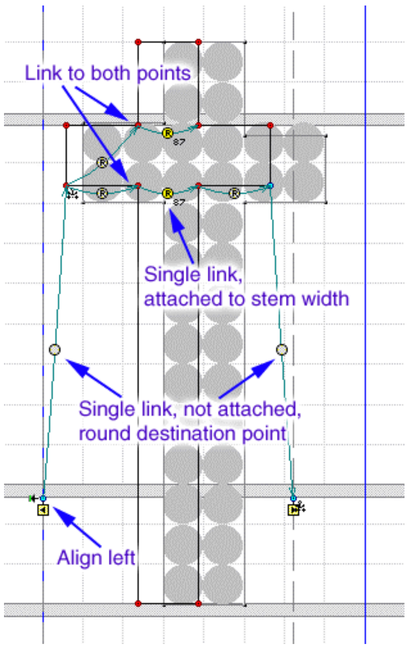
The vertical hints were pretty simply also, mainly due to the rather simple geometrical shape of the glyph.

First I used two align commands to align the top and bottom to the alignment zones. If the point is inside or near an alignment zone, FL will convert any align command to AlignTop or AlignBottom.
From these aligned points I made single links to the other side of the horizontal stems (attached to the appropriate stem widths).
Now we're left with the arm in the middle. One double link (attached to a stem width) seems to do the job well. We need to apply some middle delta hints at a couple of sizes, to move the arm to the desired position, but that's all.

The t

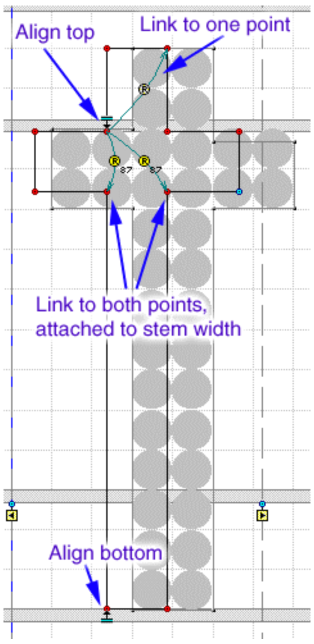
In the Florencesans font the shape is very simple. Hinting proved to be a little more complex than expected.

Horizontal



As before I started with an AlignLeft for the left base point. After I added some links to the stem I noticed that the width of the entire character was such that in many cases the strokes on either side of the stem were not equal in size. A little experimenting lead to the following:
An aligned single link from the base point to the left side of the stroke, a single link to the left side of the stem, a single link (attached to stem width) to the other side of the stem and another single link to the right side of the stroke. Because this changed the width of the glyph beyond the right sidebearing in certain sizes, a final link (aligned single link) was necessary to the base point on the right.
The part of the stem above the stroke was sometimes shifted to the left (the rasterizer doesn't see this as a stem!), so I had to link both parts of the left side of the stem to the stroke.

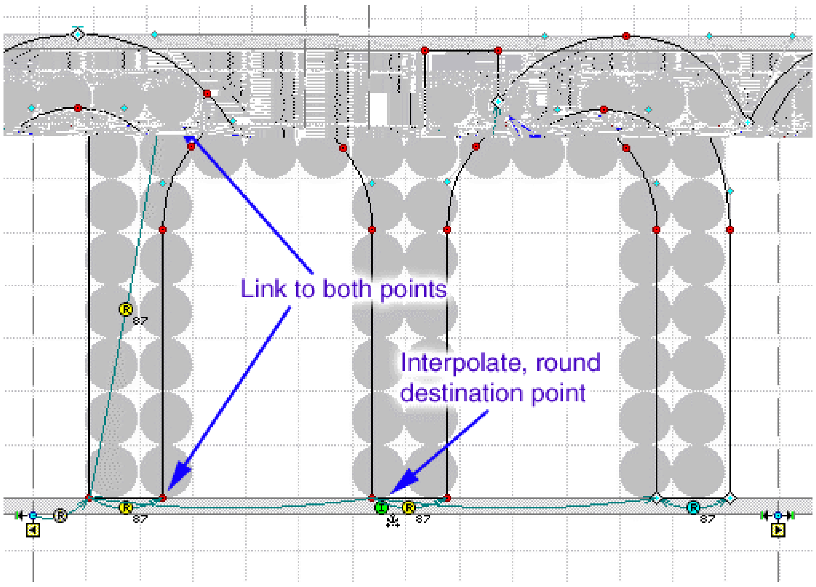
Vertical



Luckily the vertical hinting was very simple as you can see.
Align the top of the stroke with the alignment zone and use it as the base point for the single links to the other side of the stroke (to attach this to the stem width) and to the top of the vertical stem (not attached).
The last bit was to align the bottom of the letter to the alignment zone for descenders.

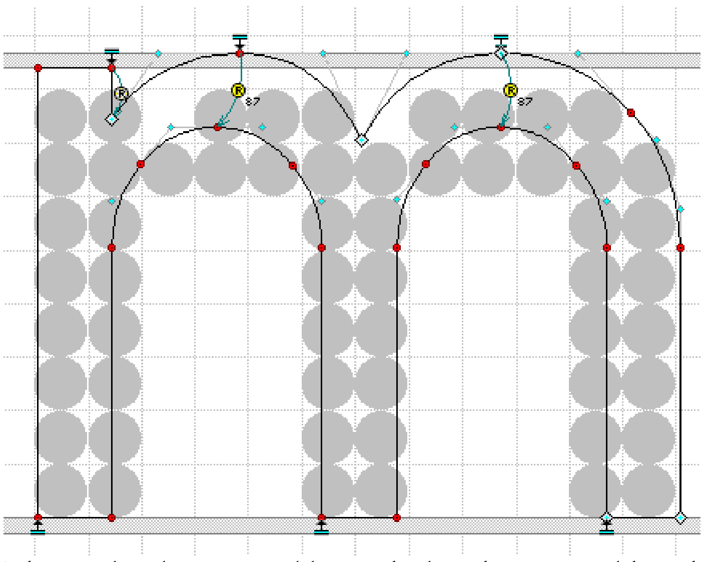
The m

Horizontal



The links on the left side are fairly standard by now. Do not forget to make a link to upper part of the stem, as the rasterizer does not recognize stems!
The challenge here was to place the central stem in the middle. An aligned interpolation command appeared to be the solution in this case.

Vertical

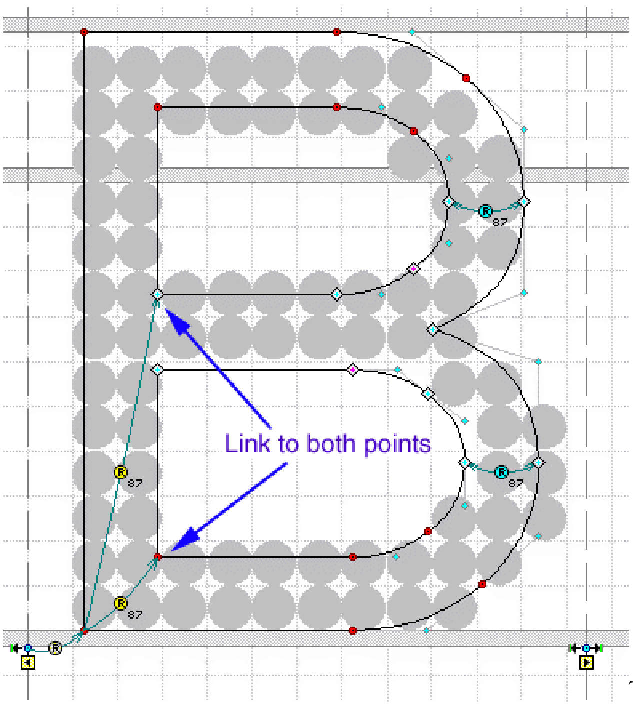


The two bowls were quickly attached to the stem width with two instructions each.
A very characteristic feature of this font is the white space between the stem and the bowl. I used one single link to tell the rasterizer to emphasise it. This point needed a couple of delta hints at various sizes however to become visible at all time.

The B

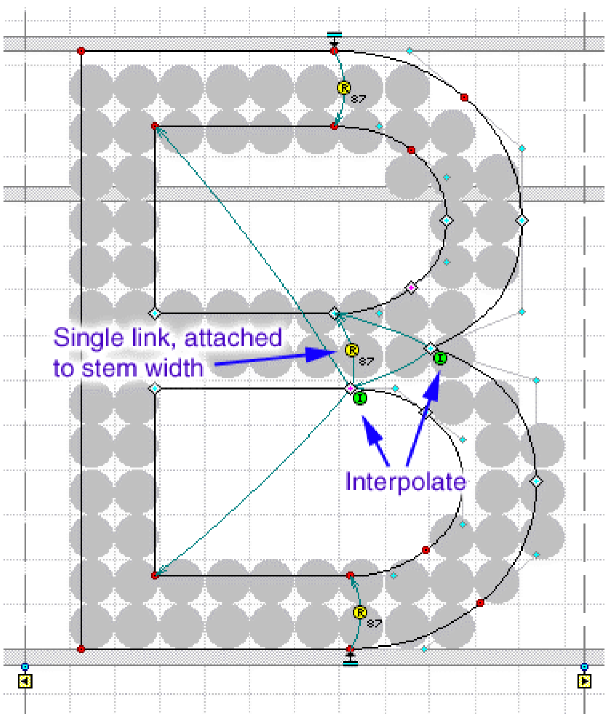
Often used as an example, it remains a surprising complex shape to render.

Horizontal



The horizontal hints are fairly straightforward. The only thing that needs attention is that both counters need to be linked to the left side of the stem. The thickness of the two bowls is easily fixed by two double links.

Vertical

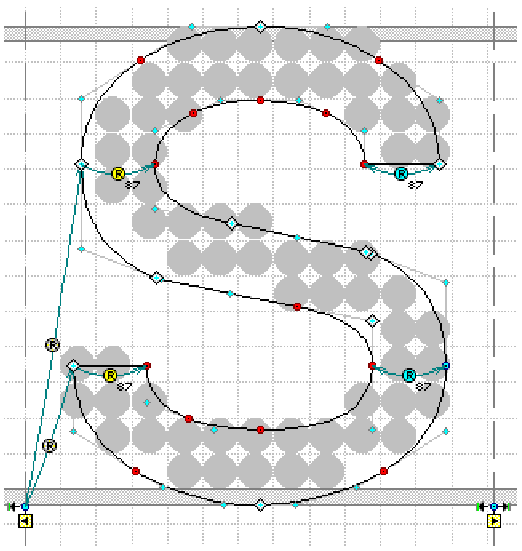


The horizontal parts on the top and the bottom are quite standard. The hardest part was to place the central bar and the point where the two bowls meet.
From the font preview it was clear that the central bar should be a bit above the middle in certain sizes. The bottom counter was a bit wider that the top one. So as a trade off, it should be a bit higher than the top counter also (if necessary). This could be achieve by applying an interpolation link to the bottom side of the central bar. This point can be use to attach the bar to the stem width.
After that the point where the two bowls meet had to be placed correctly. Another interpolation link proved to be the right way.
This glyph needed a fair amount of delta hinting before it was rendered well at all sizes.

The s

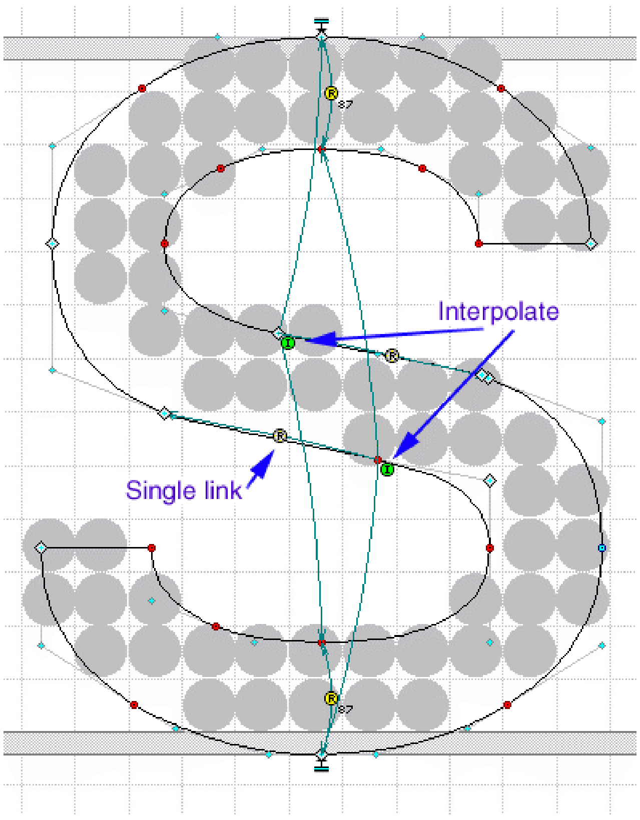
This letter is always a bit of a problem. It contains no straight (or almost straight) elements and the spine is often not horizontal at all.

Horizontal



The horizontal bit is fairly straightforward. We can only hint the parts that we are sure of. On the left side I created a bit of white space for the character spacing at small sizes and applied links attached to the stem width to make the thickness at different sizes fit the other glyphs.

Vertical



The vertical hinting started very commonly with the alignment of the top and bottom and a couple of single links to attach the upper and lower parts to the stem width.
The hardest part was the spine. In this font it had to be in the middle, so I tried to link it to the top and the bottom. The interpolation links were very useful in this situation. The only problem left was the fact that it made the spine a little too horizontal in the middle. This resulted in a spine that was the right width in the middle and that became thicker towards the left and right. A couple of single links did the trick, amazingly.
The glyph needed a lot of delta hinting, especially at small sizes and at the sizes where the stem width jumped from one to two, from two to three, etc. In these cases I decided that the thickness and character height/width ratio were more important than the exact shape. Especially at smaller sizes a legible letter was more important than anything else, so the delta hints distorted the shape quite a bit.

Shaping up with delta hints

Delta hints form a vital part of the hinting process. Exact rules and even examples are pretty useless as I found out. I'll try to address a problem where you can plan a hint fairly well beforehand.

Round bowls

Especially with smaller PPM sizes, a round shape for a bowl is impossible. You have to decide between two bad solutions: make the curve flatter (making it look a bit more like a square with rounded corners) or make the curve more elliptical (making it look a bit more like triangle). A middle delta on the extreme point of the curve will help in this case.
You can also adjust the curve of the counter a bit in this area. This way you can remove or add extra pixels to make it look a bit thicker or thinner.

If the problem is that only one single pixel must be removed, it's often easier to move the nearest point with a single final delta hint. Although the outline becomes very oddly shaped, the resulting bitmap should be what you focus on.

Positioning arms and strokes

Although it would be nice if you applied few hints that help position the arms (of e.g. the 'E', 'F') and cross strokes (of the 't', 'f', etc.), there will be situations where the rasterizer decides to put an arm or cross stroke in slightly the wrong place. Middle deltas are the obvious choice, because the other points of the shape are also moved accordingly. It is often necessary to apply hints to both sides of the arm (the upper and lower side)

Loose ends

In some letters ('9', 'a', 'e', etc.) the end of a curved part may get very close to another part of the glyph. At small sizes it may even come in contact with it, making it look like a closed shape. In this situation delta hints (either a final delta if only one pixel is involved or a middle delta) can move the end a bit away from the rest of the shape.

Strokes

Because FL has no tools to hint diagonal parts of the glyph, strokes are very hard to hint. You can try to measure the horizontal and vertical sizes, set extra stem widths with those values and attach a link to that stem width, but in most cases this is very hard too. The problem is that you must set PPM sizes at which the stem width will be displayed as 1, 2, 3, etc. pixels wide. The problem is that these sizes would not be the same as those for the standard stem width. You could try to set these values by trial and error, but delta hints will probably be necessary in this case.
Rendering a stroke may result in very ugly patterns. In this case a few delta hints may produce a more regular shape for the bitmap.

Glossary of terms

Apex – The top part of the 'A'
Arm – Horizontal parts of a glyph that have one free side (the three horizontal arms of an 'E').
Ascender – Part of some letters that is above the x-height.
Baseline – The horizontal line where bottom of most characters in a font seems to rest (except for the descenders).
Beak – Similar to a spur, but larger, sits on the end of the arms of the 'L', 'E' and 'T' in serif fonts.
Bowl – Curved part of a letter (the 'B' has two bowls).
Bracket – The connection between the serif and the stem or stroke.
Capline – The horizontal line through the tops of most capital letters.
Counter – The empty space inside a (partially) enclosed area.
Crossbar – Horizontal parts that connect two stems (the horizontal bar in the 'H').
Cross stroke – A horizontal element that intersects with a vertical element (the horizontal bar in the 'f')
Descender – The part of a lowercase letter that hangs below the baseline.
Ear – The decoration on the upper right side of the 'g'
Em – The size of the grid that contains the glyphs.
Eye – The counter in the 'e'
Glyph – the shape of a character (a letter, number, signs, letters with accents, etc.)

Hairline – The thinnest line of a typeface that consist of lines with varying widths.
Leg – The stroke on the bottom right part of the 'k'.
Link – The part between the loop and the bowl in the 'g'; a hint between two points that keeps those points at the same distance (see Single link and Double link)
Loop – The curved part on the bottom of the 'g'
Meanline – Runs through the top of the body of most lowercase letters (excepting those with ascenders).
PPM – Pixels Per eM.
Serif – Decorations on the ends of strokes and stems to improve readability.
Spine – The curve in the middle of the 'S' and 's' that runs from left to right.
Spur – Decoration at the end of the curve (the end of the 'C', 's')
Stem – The most important (almost) vertical bar of many letters.
Stroke – The most important diagonal part of a letter.
Tail – The stroke at the end of letters such as 'Q' and 'R'.
Terminal – The end of a stem or stroke without a serif.
UPM – Units Per eM.
x-height – distance between the baseline and the meanline; mostly the height of characters such as the 'x'.

Colofon

Text and illustrations by Jigal van Hemert.
Fontlab is a trademark of Fontlab Ltd.
Example font is Florencesans by Derek Vogelpohl available at Apostrophic Lab.
This file may be distributed and copied, provided that no money is charged for it.