Ygt: A graphical hinting program

Contents

Introduction

Ygt is a program for hinting TrueType fonts. Its interface is similar to those of other TrueType hinting programs, but it has somewhat different aims:

- Being written in Python, with PyQt6 and fontTools, it runs on all major platforms.
- It is independent of any font editing program.
- It can operate on either TrueType fonts or UFOs with quadratic outlines.
- It can output a hinted font, embed its output in a UFO, or produce a script that can be run from makefiles and other build scripts (it is perfect, for example, for use with fontmake).
- In addition to the kinds of hints that other programs generate, it provides an interface for user-written functions and macros, which can greatly simplify the handling of repeated features of fonts, like serifs and deltas.
- It works only with hints and does not touch any other features of a font (for example, it doesn't reposition components the way VTT does).

Getting started

Install (1) by downloading the appropriate executable file from the Ygt repository and moving it to a convenient place, (2) by typing `pip install ygt` at the command line, or (3) by downloading the source from GitHub and running `pip install .` from the base directory (Python 3.10 or higher must be installed for methods 2 and 3).

To get started on a project, launch Ygt either by double-clicking its icon (if you have the executable file) or by typing "ygt" on the command line. ☞ The executable may be slow to start: the program icon may disappear and appear again before the program's main window opens. Then open either a TrueType (`.ttf`) font or a Unified Font Object (UFO—a directory with the extension `.ufo`).

Ygt will save your work in one of two ways. If you started your project by opening a TrueType font, Ygt will save hints in a file with the same name as the font but the extension `.yaml` instead of `.ttf`. When you return to the project after the first session, open the `.yaml` file instead of the font, which must remain where it was when you first opened it.[1]

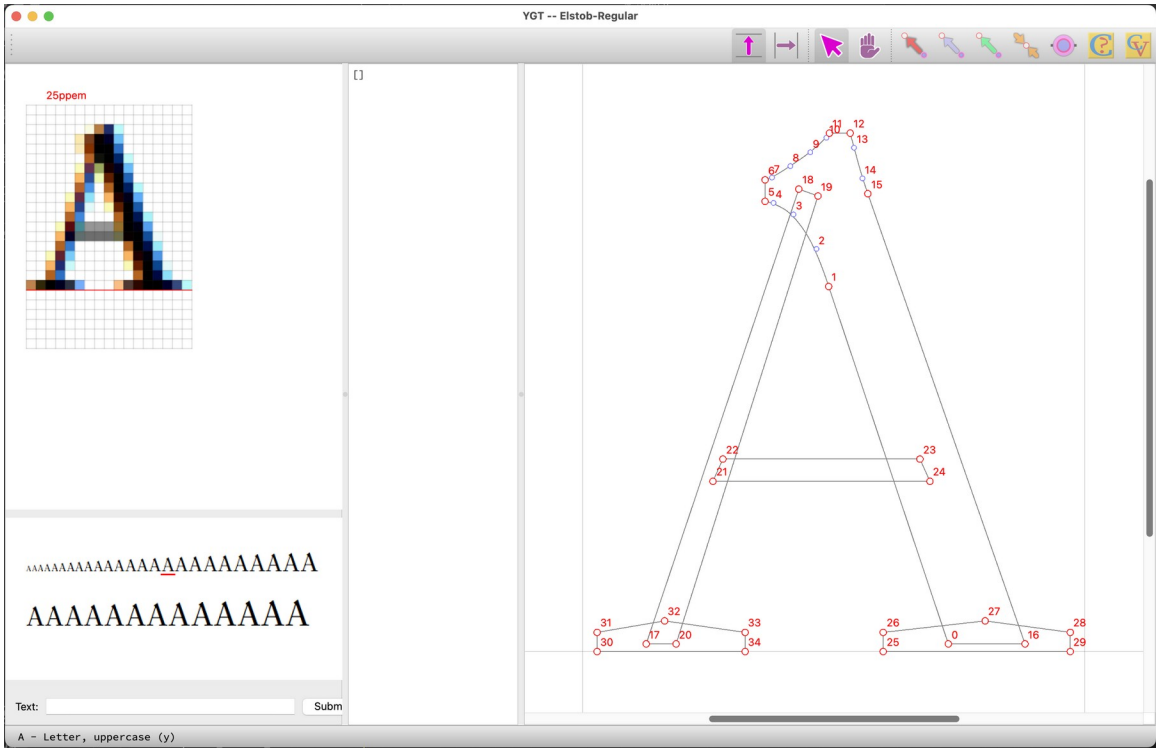
If you started by opening a UFO, Ygt will save your work in the UFO's data directory (in a file named `org.ygthinter/source`). When you return to your project, open the UFO exactly as you did the first time: Ygt will find its data inside it.

In either case, the saved file is in exactly the same format, so that a file created for a UFO will work with a font and vice versa (use the menu command `File→Save As` to switch back and forth).

If you are working with a font that is already hinted, you can decide whether to keep its hints and add your own hints to it or discard them and start your hinting from scratch. For details, see "Merge Mode" below.

After you've opened a font, the screen looks like this:

1 If you want to move the font file, you can edit the "in" field of the "font" section of the `.yaml` file. Ygt currently provides no way to do this: you must edit the file with a text editor.



On the left is the **preview panel**, where you can view a blown-up image as it will be rendered on screen or in print. You can view it hinted or unhinted and in one of several rendering modes; if you are hinting a variable font, you can apply any of the font's instances to it (e.g. Bold, Condensed).

Below the main preview panel, a smaller panel displays the current glyph in an array of sizes (or resolutions, if you prefer, in **ppem**, pixels per em). Alternatively, you can type a short text in the "Text" box to see your glyphs in context. This text will be the same size as in the large preview window (but not blown up), and it will have the same rendering mode and with the same instance. In addition, you can apply OpenType features to this preview so that you can view glyphs that can't be typed in the "Text" box: ligatures and any variants.

For more about the preview panel, see "The preview panel" below.

In the middle of the window is the **code panel**, which displays the source code for either the y or the x axis of the current glyph. This source code is in YAML, a very simple data serialization language. Ygt uses YAML instead of borrowing or inventing a programming language because its source is meant to be descriptive, not imperative: it describes the positions of the points that make up a glyph outline rather than dictating the commands that will position them. You can edit the code in this panel: when you have done so, type "Ctrl-R" to compile it and display the results in the editing panel to the right.

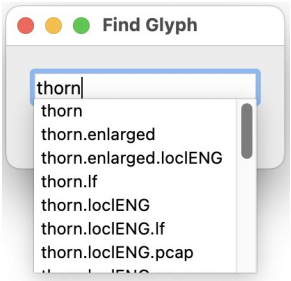
Ygt's hinting language will be described in The YAML source, below.

To the right of the code panel, the **editing panel** displays an outline of the current glyph with the points that define it (composites are not displayed here, but you can view previews of them). On-curve points are displayed as red circles and off-curve points as smaller blue circles. As hints are mostly applied to on-curve points, you may wish to hide the off-curve points. To do this, right-click (or, on a Mac, control-click) anywhere in the panel and select `Hide off-curve points` from the context menu. On the same context menu you can also decide whether or not to display point labels, which can be numbers, coordinates, or names (see Point identifiers below).

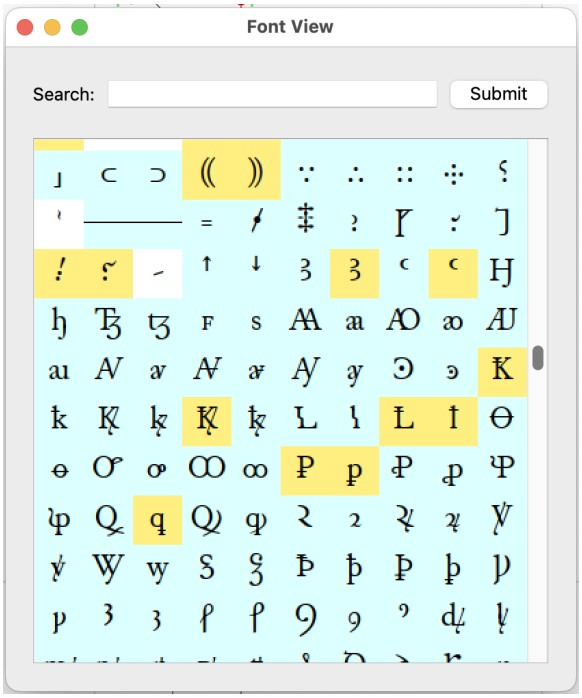
Above the editing panel is a toolbar with various tools for editing the glyph's hints. They are grayed out initially, but are enabled whenever you select an appropriate set of points.

Navigation

To move from the current glyph to the next or previous one, type the right or left arrow key. (Glyphs in the font are ordered by Unicode value; glyphs that lack Unicode values come after those that have them, and these are ordered alphabetically.) If you know the name of the glyph you want to go to, or at least how it begins, select `Edit→Find` (or type "Ctrl-F"—"Cmd-F" on the Mac) and type in the dialog that appears, which will offer you a choice of glyphs matching what you have typed so far:



You can get an overview of all the glyphs in the font by selecting `View→Show Font Viewer`:



In this window (it's not a dialog: you can and probably should leave it open while you work), glyphs that have been hinted are highlighted in blue, composites in gold. The glyph currently being edited is framed in red. Click on any glyph to go to it in the editing panel. As it may be difficult to find the glyph you're looking for in a large font, you can filter this display by typing in the "Search" box at the top of the window. Only glyphs whose names contain what you have typed will be displayed after you click "Submit."

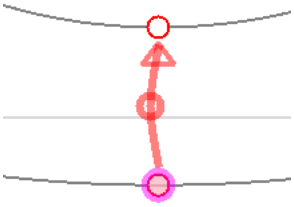
Finally, if text is displayed in the lower preview panel, click on any glyph in the text to go to that glyph.

Selecting points and hints

Selection of points works here the way it does in most graphical editors: select a point by clicking on it or dragging over it. The hollow center of the point will be filled with blue. To select multiple points, drag over them. To add to or subtract from a selection, hold down the Shift key while dragging or clicking.

You can't select a hint by dragging: you must click on it. Hints marked by arrows have a button in the middle of the stem to provide an easy target for the mouse. Selected hints are darker in color than unselected ones.

Anchor hints are signaled by a pink circle around a point:



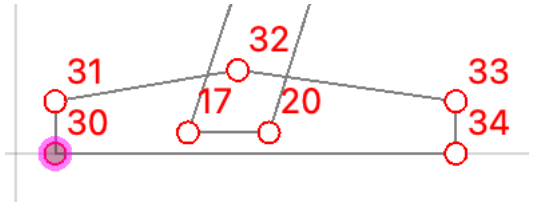
If you click anywhere in this circle, you will select the hint and not the point. To select the point rather than the hint, you must drag over it. If you want to select both the hint and the point, first click on it, then drag over it while holding down the Shift key.

The pink-tinted interior of the point signals that it is **touched**—its position regulated—by a **rounded hint**. A touched point is one that has been affected by a hint. A rounded hint may either cause the point(s) it touches to be rounded to the grid or round the distance between two points.

Basic hinting

To add a hint, select one or more points (click on them or drag the mouse over them) and either click on a button in the toolbar or type a shortcut key. (The shortcut keys are displayed in the tooltips that appear when you move the mouse over the tool buttons.)

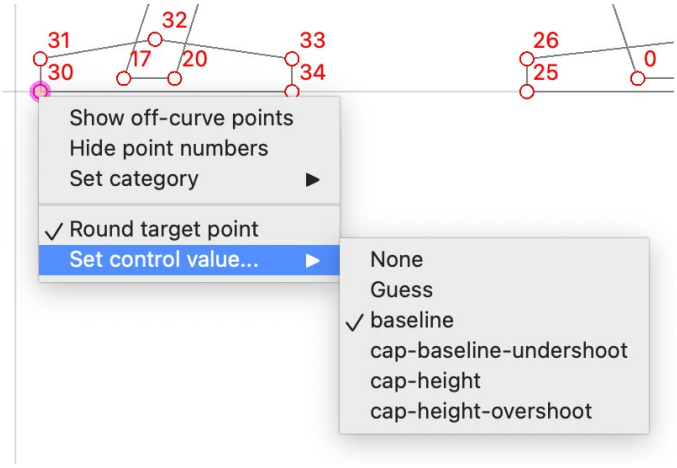
For example, to anchor a point to a position on the grid (which you must do first whenever beginning a sequence of hints), select a point and either click the "Anchor" button (see right) or type "A." The anchored point will be highlighted in pink:



Now you should assign a **control value** (CV) to this hint. A CV is a number that specifies the **position** of a point or a **distance** between two points. Use CVs to regulate features (like serifs and the tops of ascenders) that recur in a number of glyphs. CVs are named rather than numbered in Ygt, making them easy to use. When you first open a font, Ygt creates a little collection of position-type CVs: check over these in the "Font Info" window (Ctrl-I) and add more as you need them.

To use a CV, right-click on a hint, select `Set control value` from the context menu, and choose a CV from the sub-menu. The CVs on this menu are a subset of the whole collection—in this case, the ones most appropriate for a capital letter. They are also alphabetized (a fact you should

remember when you name your own CVs). If you're not sure which CV is best, or if it's easier, choose `Guess` from the menu. Alternatively, select the hint and click the "Guess Control Value" button or type a question mark.



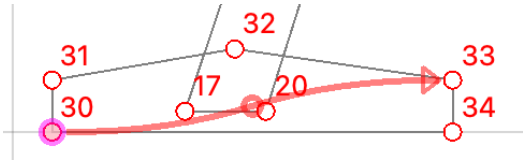
☞ You can combine the actions of adding a hint and guessing at a CV by holding down the Control (Command) key when you add an anchor or stem hint (these being the hint types that can make use of CVs). ☞ To round a target point or points (regardless of the default for the hint type), hold down the Shift key while adding the hint.

As you add hints in the editor, the code panel is automatically updated:

```
- ptid: 30
  pos: baseline
```

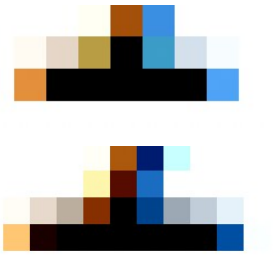
Here **ptid** ("point identifier"—it isn't always a number, as we'll see later on) identifies the point affected by this hint, and **pos** is a position-type CV.

To add another hint dependent on the first, select the point you just hinted (remember that you must drag over an anchored point to select it), then select another point and either click the tool with the red arrow or type "T":



You have just added a **stem hint**—that is, one that regulates the distance between two points—usually, but not always, a stem (more on this in a moment). It is displayed as a red arrow running from a **reference point** (in this case, point 30, which you just anchored) to a **target point** (33).

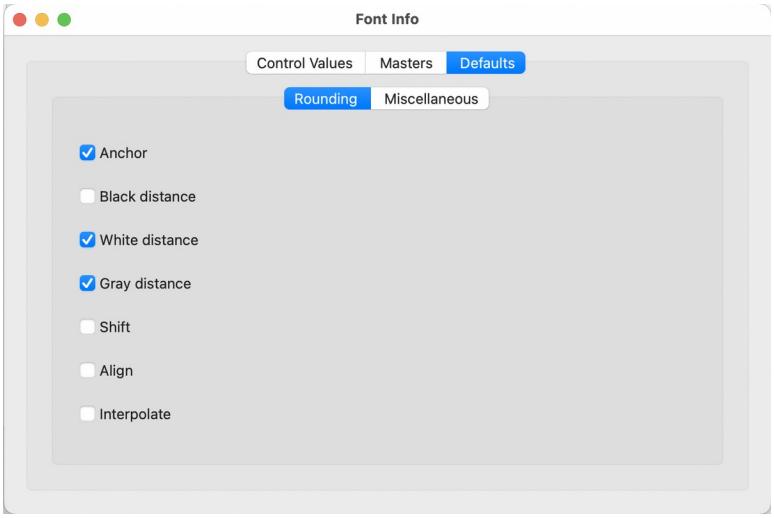
The effect of this hint is to move point 33 to a position on the y axis which is the same distance from point 30 as in the original outline. This is, by default, a **rounded** distance—that is, a distance in whole, not fractional pixels.



At this point, you can go in either of two directions. One is to assign a distance-type CV to the stem hint: this is most appropriate to the CRT screens of yesteryear and early flat screens. The more modern choice is to turn off rounding for the stem hint and leave it at that. The top of the serif will now be anti-aliased, with the result that it will appear to the reader to have a weight consistent with the glyph's original design. To turn off rounding for the hint, right click on it and uncheck `Round target point` on the context menu.

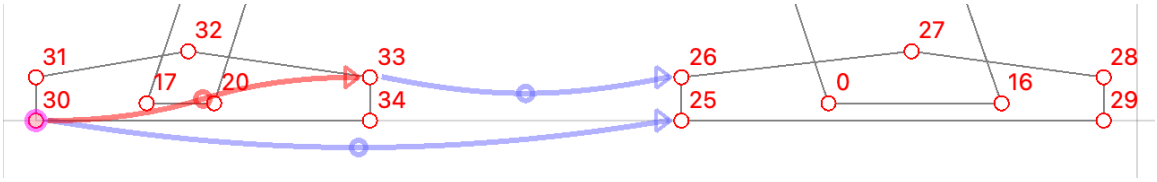
There are three kinds of stem hint, and Ygt tries to choose the best one automatically. The most common kind is for a "black distance" (**blackdist** in the code), that is, a distance filled with ink or pixels—usually black. The second kind is a "white distance" (**whitedist**), a usually white space between filled areas. The third kind is a "gray distance" (**graydist**), any space that is neither white nor black. (Note that the stem of the arrow is styled differently depending on the kind of distance: solid for black, short dashes for white, long dashes for gray.) If you think Ygt has inserted the wrong kind of hint, select the hint and choose "Black," "White," or "Gray" from `Set Distance Type` on the context menu.

To turn off rounding for a particular kind of stem hint, bring up the "Font Info" window (`File→Font info`), click over to the "Defaults" tab, then the "Rounding" tab, and uncheck the appropriate box. Here we turn off rounding for **blackdist** hints:



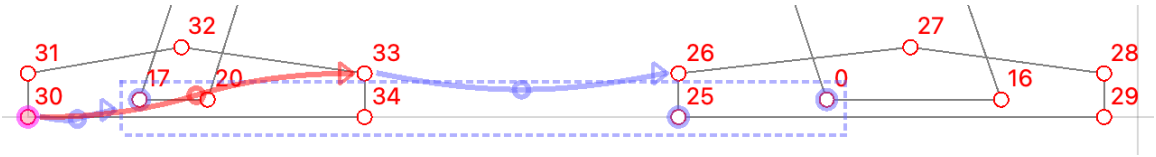
Now any stem hint we add will be unrounded when it is regulating a black distance. You can change the rounding of an individual hint later if you wish.

Since we've hinted only one serif, the pair of them will be mismatched at some resolutions. To make sure they always match, we'll add some **shift** hints (using the tool with the blue arrow). A shift hint causes a target point to be moved by exactly as much as the reference point has been moved. Select points 30 and 25 and click the blue arrow or type "H." Then do the same with 33 and 26:



We also need to **touch** the bottoms of the two diagonal stems to keep them from being pushed above or below the serif. The best way to do that is to add one point on the bottom of each stem to one of the nearby shift hints.

Here you need to know that shift (blue arrow), align (green arrow) and interpolate (gold arrow) hints can target more than one point—a **set** (stem hints, by contrast, can target only one point at a time). All you need to do, once you've added the hint, is select the hint and as many points as you want to add to it and type the **plus key**. The result looks like this:

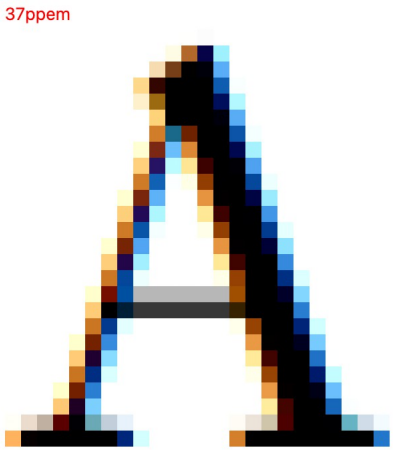


On screen, a **set** is a group of highlighted points enclosed in a box of dotted lines (points 20 and 34, which are inside the box but not highlighted, are not part of the set). It belongs to and is part of a hint, and the hint's arrow now points at the box. Clicking on one of the highlighted points or on the box will select the whole hint. As with an anchor hint, you can select a highlighted point (but not the hint) by dragging over it.

Another way to make shift, align, and interpolate hints target more than one point is to select more than one point to target when you add the hints. If you had added the lower shift hint in the example above by selecting points 30, 17, 25, and 0 and typing "H," the effect would have been the same as you got using the plus key.

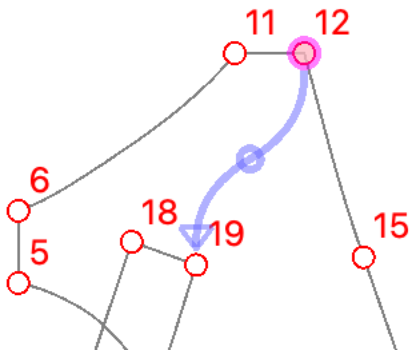
You can delete any point or points from a set by selecting them and typing the **minus or hyphen key**. You cannot remove the last point this way; instead, delete the hint by selecting it and typing the delete or backspace key (deleting a hint also deletes any dependent hints).

The result of our efforts so far looks like this:



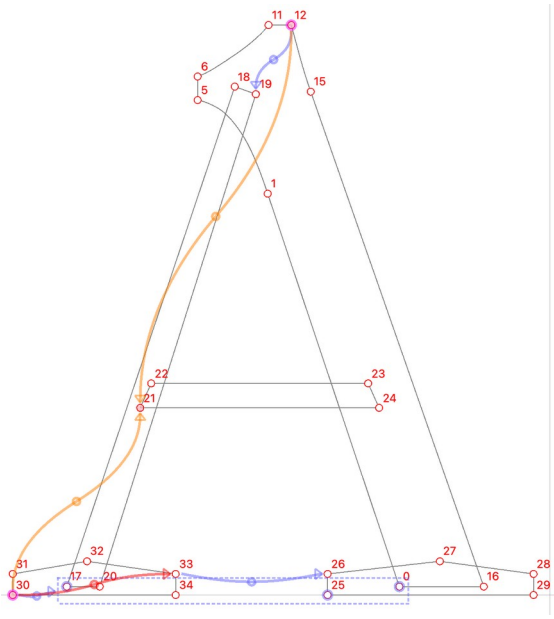
Now the serifs match, but we've got more to do. The top of the letter needs to be hinted to make sure it harmonizes with the tops of other capitals, and the cross bar has to be dealt with.

All we need to do at the top of the letter is anchor it with the CV "cap-height-overshoot" (set up by Ygt when we first opened the font) and link the anchored point to the top of the left-hand stem with another shift hint:



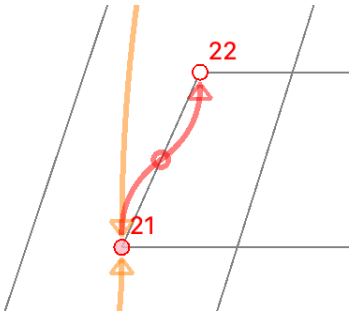
These hints ensure that point 12 always appears at the same y position on the grid, and point 19 is always positioned correctly relative to point 12.

To make sure the bar is correctly positioned, we need an **interpolate** hint, which positions one or more points relative to two reference points so that the relationship of all these points is the same as in the original glyph. To do that, select two touched points and one that has not yet been touched by a hint, and click the "interpolate" tool or type "I":

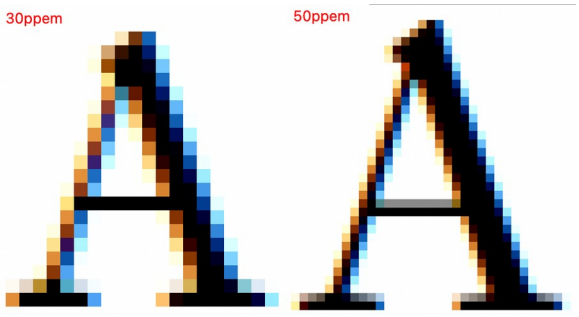


Rounding point 21 to the grid will make sure the bottom edge of the bar is sharp. To do that, right click on one of the "interpolate" hint's buttons and choose `Round target point` from the context menu. ☞ Alternatively, hold down the shift key when you create the hint to round the affected point or points to the grid.

For the top edge of the bar, we face a decision like the one we made for the serifs: round to the grid while regulating the bar's width with a CV or add an unrounded stem hint. Once again, the modern choice is the unrounded hint. Select points 21 and 22, type "T," and uncheck `Round target point` on the context menu:



At low resolutions, both edges of the bar will now look sharp, because the **blackdist** hint maintains a **minimum distance** of one pixel between the reference and target points. At higher resolutions, the top of the bar will often seem fuzzy if you look closely. But this fuzziness tricks the eye into perceiving the edge of the bar as falling between two pixel edges:



You can dramatize the effect by repeatedly typing the up or down arrow key to change the resolution of the preview while watching the bar and the serifs gradually thicken.

You should save frequently when working with Ygt, as it is new software and may contain bugs that cause it to crash. (If the program crashes, please copy the stack trace from the console and open an issue at the GitHub Ygt site.) To save to either an external `.yaml` file or a UFO, simply type "Ctrl-S." When you return to the project, open either the `.yaml` file or the UFO—never the TrueType (`.ttf`) font. To export a hinted font, type "Ctrl-E" (for `File→Export font`). This will either produce a hinted copy of the TrueType font (with "-hinted" added to the file name) or write the compiled instructions into the individual .glif files of the UFO. Alternatively, run the program Xgridfit (one of Ygt's dependencies) on the .yaml file:

```
xgridfit Elstob-Regular.yaml
```

If you are running the program from a Makefile or other script, add the option -q or -qq to suppress most or all warnings and messages.

The preview panel

By default, the upper preview panel displays an enlarged image of the current glyph at 25 ppem:

To compare the hinted with the unhinted version of a glyph, turn hinting off and on with `Preview→Hint preview` (Ctrl–P).

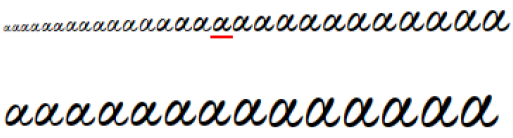
You can change the size of the preview with the `Grow` and `Shrink` options on the `Preview` menu, or with the up and down cursor keys, optionally modified with the Control (Command) key to increase or decrease the size by ten pixels. You can also set the size with `Preview→Set size` (Ctrl-L), which brings up a dialog that lets you enter a size in the range 10–400.

You can change the render mode of the preview, choosing between grayscale and two styles of subpixel rendering—one displaying the R, G, and B components of each pixel in a single blended block and the other displaying each pixel's color components separately. The default is "Subpixel (1)," which shows the blended color components.

Normally Ygt displays either black characters on a light background or white characters on a dark background, depending on whether you have chosen a dark or a light theme for your display. However, you can choose a black-on-white or white-on-black theme for the preview so that you can see how the hinted glyph will appear under various conditions. For example, the image to the right shows the letter g at 45ppem with the render mode "Subpixel (2)" on a black background. You can also turn the grid off and on with `Preview→Show grid`.

If you're hinting a variable font, you can choose from the font's named instances via `Preview→Instances`, or cycle through them with `Preview→Previous/Next instance` (Shift–greater and Shift-less—on an English keyboard above the comma and period).

The small lower panel normally shows the current glyph in an array of sizes, beginning at 10ppem and increasing until there is no more room in the panel. The glyph that matches the one in the upper preview panel is underlined in red:



All of the options you can apply to the upper panel are also available here, except that you cannot change the size and there is only one subpixel render mode available. Click on any size glyph in this panel and the image in the upper panel will be displayed in that size.

You can also use the lower panel to display arbitrary text in the current size and instance, and with the current render mode. Simply type something in the "Text" box below the panel and either click the "Submit" button or press the Enter or Return key. As many fonts contain characters that can be displayed only via OpenType features, you can also select features for this panel. In addition to certain default features (see below), Ygt allows you to apply any features that it finds in the font. Here, for example, "smcp – Small Caps" is turned on:

When zombies arrive,

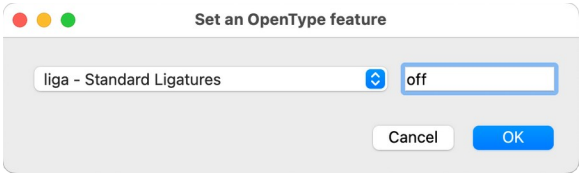
quickly fax Judge Pat

The selection of available features often depends on the script and language settings, which you can also select from the `Preview` menu. The default script and language are "DFLT:dflt," but you can change these to anything defined in the font.

To apply OpenType features, select the appropriate script and language, then select as many features as you like. When you're satisfied, click the "Submit" button to see the result.

Most OpenType features are either on or off, but some (especially salt and the cvNN features) are usually indexed. When you choose salt or cvNN via the `Preview→Features` menu, a dialog box appears in which you can type in an index number. If any other features in your font are indexed, turn them on via the `Preview→Feature...` command, which brings up a dialog box in which you can select the relevant feature and type either **on**, **off**, or a number between 1 and 99.

Note that several positioning (GPOS) features are on by default,[2] as are some substitution (GSUB) features.[3] These do not appear in the `Preview→Features` menu, but the GSUB features can be turned off and on again via the `Preview→Feature...` command:



2 These are **kern** "Kerning," **mark** "Mark Positioning," **mkmk** "Mark-to-mark Positioning," **abvm** "Above-base Mark Positioning," **blwm** "Below-base Mark Positioning," **curs** "Cursive Positioning," and **dist** "Distance."

3 These are **ccmp** "Glyph Composition/Decomposition," **liga** "Standard Ligatures," **calt** "Contextual Alternates," **rlig** "Required Ligatures," **locl** "Localized Forms," **clig** "Contextual Ligatures," and **rclt** "Required Contextual Alternates."

The YAML source

The source displayed between the preview and editing panel is editable. Once you have learned the simple hinting language used by Ygt, you may sometimes find that editing the source is the quickest and surest way to manage a glyph's hints.

The source editor employs a couple of methods to help you write valid code. First, it validates continuously as you type, displaying the status of the code at the bottom of the Ygt window:

When the source is invalid, Ygt will also display an Error Console window with more detailed information about the error. When this window appears, leave it on screen so that you can consult it if necessary. Nearly all the program's errors are displayed here.

Second, the code panel employs a simple scheme of color coding. When a **point block** (we'll explain the concept shortly) is nested inside another point block (nesting signaled by a two-space indentation), it changes color. The hyphen that signals a list in YAML will be red when it is correctly positioned:

```
- ptid: 20
  pos: xheight-overshoot
  points:
  - ptid: 10
    rel: blackdist
    round: false
- ptid: 37
  pos: lc-baseline-undershoot
  points:
  - ptid: 0
    rel: shift
    points:
    - ptid: 30
      rel: blackdist
      round: false
  - ptid: 52
    rel: blackdist
    round: false
```

When a keyword in the block is wrong (in the example below, "ptic" for "ptid"), it turns black. When a hyphen is wrongly positioned, it turns a different color (and of course, the status bar will signal that the code is "Invalid"):

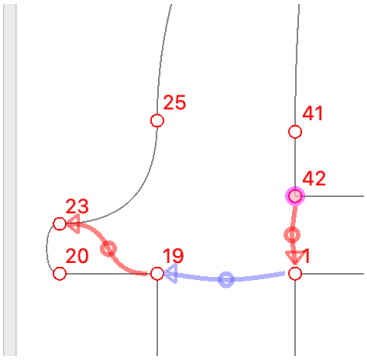
```
- ptid: 0
  rel: shift
  points:
- ptic: 30
    rel: blackdist
    round: false
```

When you have made a change to the source code, type "Ctrl-R" ("Cmd-R" on the Mac) or select `Code→Compile`. If the code is valid, your changes will be reflected immediately in the editing panel, and a second or so later in the preview panel.

Most of the time, though, the program will compose the YAML code as you work in the editing panel. Ygt will do its best to keep the code in order (for example, making sure that reference points are always points that have already been touched). If you suspect that the code has gotten out of order (for example, you have added all the hints you need but the preview still doesn't look right), try selecting `Code→Clean up`. Chances are that will fix the problem; if not, you may have to edit the source yourself.

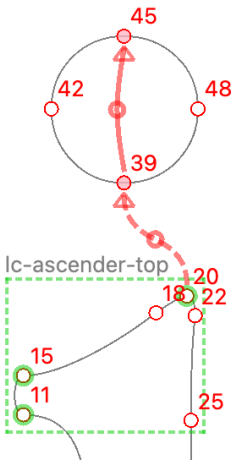
The YAML source consists of a list of **blocks**—that is, descriptions of how a point or set of points should be positioned on the grid. Each block can contain a list of one or more dependent blocks under the "points" keyword. For each dependent block, one reference point (in the graphical editor, the point at the starting end of a hint arrow) must be the same as one of the parent block's target ("ptid" for "point identifier") points. For example:

```yaml
- ptid: 42
  pos: xheight
  points:
  - ptid: 1
    rel: blackdist
    round: false
    points:
    - ptid: 19
      rel: shift
      points:
      - ptid: 23
        rel: blackdist
        round: false
```



Here is a sequence of four points in the cross bar of the letter f. It begins with an anchored point (42), as sequences always must. This hint uses the CV "xheight." Second, point 1 is an unrounded black distance from 42; third, point 19 is shifted along with 1; fourth, point 23 is an unrounded black distance from point 19. The reference point ("ref") for each of the arrow hints is implicit: it is understood to be the same as the parent target ("ptid"), at least when the parent target is unambiguous. When a hint targets more than one point, its child hint must have a "ref" property indicating which of the parent points is its reference.

```
- ptid:
    p1: 20
    p2: 11
    p3: 15
  macro:
    nm: lc-ascender-top
    cv: lc-minim
  points: []
- ptid: 39
  ref: 20
  rel: whitedist
  points:
  - ptid: 45
    rel: blackdist
    dist: dot-dist
```



For example, here the macro “lc-ascender-top” (we’ll say more about macros and functions later) regulates three points (11, 15, 20), so the “whitedist” hint that positions the dot must have a “ref” property indicating that point 39 is to be positioned with reference to point 20 (not 11 or 15). But the hint for point 45 does not need a “ref,” since its reference point is implicitly the “ptid” (39) of its parent.

Here is a list of types of point blocks (excluding functions and macros, which we will discuss later):

anchor

An anchor positions a point at a particular place on the grid. It can take the following properties:

ptid — Required. A **point identifier** for the target, which must be a single point.

pos — Optional. A position-type CV prescribing a location for the target point on the current axis. If this is absent, the point is either moved to the nearest grid line (if “round” is “true”), or simply touched without being moved (if “round” is “false”). If the original position of the point is too far from the position indicated by the CV, the point won’t be moved to that position, but rather touched or rounded to the grid (see next paragraph).

round — Optional. The default is “true.” Whether the point should be moved to the nearest grid line after any other positioning. This may also be a type of rounding: “to-grid” (the default), “to-half-grid,” “to-double-grid,” “down-to-grid,” or “up-to-grid.” At present these special rounding types must be entered in the YAML code: the GUI offers no way to enter values other than “true” and “false.”

points — Optional. A list of dependent points or hints—that is, points that should be positioned relative to the point anchored by this hint.

stem hints: blackdist, whitedist, graydist

These hints (always represented by red arrows) regulate the distance between a reference point and the target point. With a "blackdist" hint, the distance crosses a space that will be filled with ink. With a "whitedist" hint, the distance crosses a space that will not be filled with ink. A "graydist" hint regulates any distance that is neither black nor white.

rel — Required. Expresses the relationship between the reference point and the target point. This may also be thought of as the type of the hint. Permitted values: blackdist, whitedist, graydist.

ptid — Required. The target point. It must be a single point.

dist — Optional. The name of a distance-type CV. If it is present, the distance between the reference point and the target point will be regulated by the named CV. If absent, the original distance between the two points will be used. In the modern style of hinting, it is seldom necessary to use distance-type CVs.

ref — Required when the parent hint has more than one target point. The point that the target point is positioned relative to.

round — Optional. The default for stem hints is "true." Whether to round the distance between "ref" and "ptid." If "ref" is on a grid line, then "ptid" will be positioned on a grid line too. You can enter rounding types in the source code, as with the anchor hint.

min — Optional. The default for stem hints is "true." Whether to maintain a minimum distance between "ref" and "ptid."

points — Optional. A list of dependent points or hints.

shift, align

"Shift" moves the target point along the current axis by as much as the reference point has been moved. "Align" aligns the target with the reference point, so that aligned points will end up in a horizontal line when the axis is "y" and in a vertical line when the axis is "x." The "align" hint can be tempting, but "shift" is generally much more useful.

rel — Required. The relationship between the reference point and the target point. This may also be thought of as the type of the hint. Permitted values: shift, align.

ptid — Required. The target can be either a point or a list of points (a "set").

ref — Required when the parent hint has more than one target point. The point that the target point or set is positioned relative to.

round Optional. The default is “false.” Otherwise, rounding will move targeted points to the nearest grid line after any other positioning.

points Optional. A list of dependent points or hints.

interpolate

Positions one or more points relative to two other points in such a way that all points have the same relationship to each other as in the original outline. This hint is never the child of another hint.

rel Required. The relationship between the reference points and the target point. For this type of hint the value can only be “interpolate.”

ptid Required. The target can be either a point or a list of points (a “set”).

ref Required. A list of two points.

round Optional. The default is “false.” Whether the position of the target point(s) should be rounded to the grid after interpolation.

points Optional. A list of dependent points or hints.

Functions and macros

Functions and macros are written in the Xgridfit hinting language (one or more other language options will be offered later) and embedded in the YAML source. In the TrueType instruction language, functions are somewhat like functions in other programming languages, but they are numbered rather than named, and they have no formal mechanism for passing parameters. Xgridfit supplies these wants. It also makes available a macro feature. Macros insert code instead of function calls in the program stream. Macros produce a program similar in size to one with equivalent functions, but macros can take advantage of conditional compilation, and can take sets and even bits of code as parameters.

Because functions and macros have to be written in Xgridfit (though you can use Xgridfit’s “command” element to insert raw TrueType instructions if you prefer), writing them involves a learning curve; but a function or macro, once written, may be used (possibly with minor changes) in a number of fonts.

Functions and macros are very much alike from a programmer’s point of view. They consist of a YAML list of parameters and the code itself. Here’s an example of a macro:

```
lc-ascender-top:
 p1:
  type: point
```

```
 subtype: target
p2:
 type: point
 subtype: target
p3:
 type: point
 subtype: target
cv:
 type: pos
 val: lc-ascender
code: |
 <code xmlns="http://xgridfit.sourceforge.net/Xgridfit2">
  <move distance="cv">
   <point num="p1"/>
   <move>
    <point num="p2"/>
    <move distance="lc-serif">
     <point num="p3"/>
    </move>
   </move>
  </move>
 </code>
```

The macro definition lives in a “macros” block in the source file: you can edit it by selecting `Code→Edit macros`. The macro begins with its name, followed by a list of parameters and the code itself.

Each parameter declaration needs, at minimum, a name and a type. The name should be something that helps you remember what the parameter is for; the type can be **point**, **pos**, **dist**, **int**, or **float**. A point parameter can also have a subtype, **target** or **ref**. Those who are familiar with the TrueType instruction set will recognize that these types have no counterpart in the compiled code; but they enable functions and macros to behave like other hints in the YAML source.

A parameter declaration can also have the property **val**. This is the default value of the parameter —the value that is used when the parameter is omitted from the function or macro call.

The code itself must be enclosed in <code></code> tags, and the opening <code> tag must have the Xgridfit namespace declaration, as in the example.

A macro call looks like this:

```
- ptid:
  p1: 10
  p2: 12
  p3: 14
 macro:
```

```
    nm: lc-ascender-top
    cv: xheight-overshoot
```

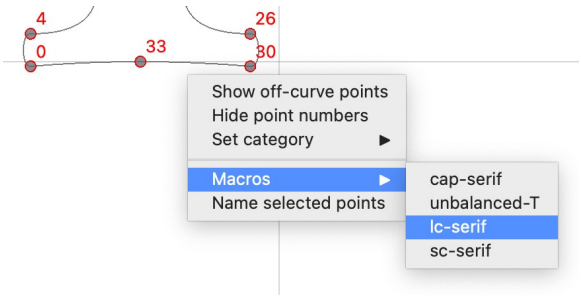
If you find it more convenient, you can also use a JSON-like syntax:

```
{ptid: {p1: 10, p2: 12, p3: 14},
macro: {nm: lc-ascender-top, cv: xheight-overshoot}}
```

Ygt will convert this to its default YAML syntax when you compile the code (Ctrl- or Cmd-R).

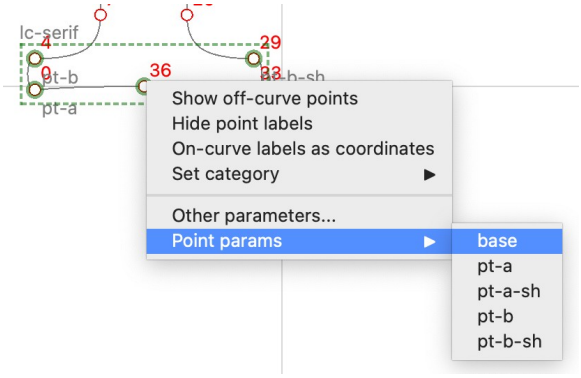
Those of the macro's parameters that refer to points are passed in the property "ptid," others in the "macro" (or "function") property. The latter property must also declare the macro's name ("nm").

It will usually be most convenient to insert the macro or function call via the graphical hint editor. To do this, select the points that will be affected by the macro or function, then right click to bring up a context menu and select the macro or function by name:

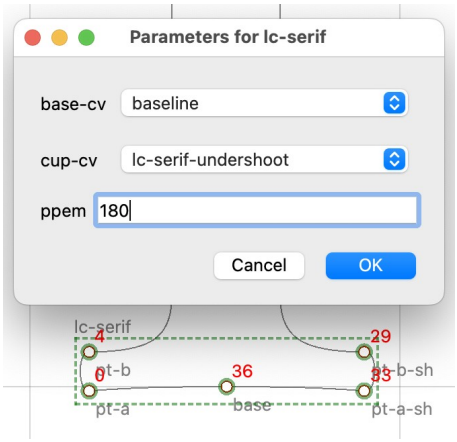


The menu presents only a selection of the macros and functions you have defined, depending on the number of points you selected.

Ygt has no way of knowing how to assign your selected points to parameters: they will be applied more or less at random. To sort them out, first select the macro by clicking on its border or one of its points. When the macro is selected, you will see which parameter is associated with which point. Right click on any point that is incorrectly assigned and assign it to the correct parameter by selecting from the `Point params` submenu:



Repeat this operation until all points are correctly assigned. To set other (non-point) parameters, select “Other parameters” from the same menu and adjust them using a dialog box like this one:

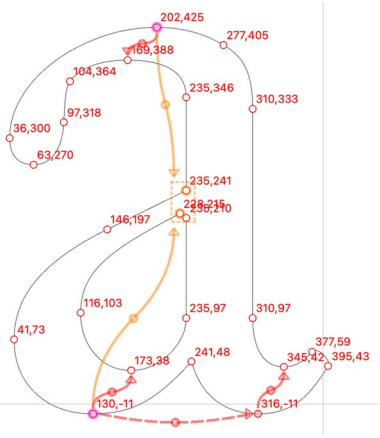


When you first open a font, Ygt inserts one user-callable function in the code: “delta,” which can be called like this:

```
- ptid:
  pt: 26
 function:
  nm: delta
  size: 26
  distance: -5
```

The function takes a single point as a parameter, a size (in pixels per em) at which the delta is effective, and a distance (from -8 to 8, in eighths of a point) to move the point on the current axis. A distance of zero disables the delta hint. Note also that point-moving deltas should not generally be used in variable fonts.

Point identifiers



In a TrueType font, the points that define a glyph’s outlines are numbered from zero, and TrueType hints refer to points by these numbers. You’ll have little need to think about point numbers unless you edit the YAML code yourself, but there is one thing worth knowing—namely, that Ygt can refer to points **either by number or by coordinates**. In the YAML code, a sequence like **{548;425}** means “the point at coordinates 548,425.”

The reason to use coordinates instead of indices in your code is that index numbers are unstable, especially if you use more

than one program to generate fonts, for different font-editing programs will number points differently. Further, if you make even a minor change in a glyph's outlines after you have hinted it, the point numbers will probably change next time you generate the font, even if you always use the same font editor. The result will be invalid hints and messed up glyphs.

But software that generates fonts rarely if ever alters the coordinates of on-curve points—so if you hint only on-curve points and designate those points by their coordinates it won't matter if you generate the font with, say, Glyphs or fontmake.

Display coordinates rather than indices for the labels of on-curve points by selecting `On-curve labels as coordinates` from the context menu. When this option is selected (even if point numbers are hidden), hints added to the YAML code will refer to points by their coordinates. You can also change all point identifiers in the current block of code to coordinates or indices by selecting `Indices to coords` or `Coords to indices` from the `Code` menu. It is a good idea, if you have been working with point indices (which are admittedly easier to work with than coordinates) to select "Indices to coords" after you are finished hinting a glyph.

Points and collections of points can also be named in Ygt. To do this, select one or more points, select `Name selected point` (or `Name selected points`) from the context menu, and supply a name when the "Name points" dialog appears. The name of a point will become its default label. Ygt will use point names when generating code (making the code more legible), and when editing your code you can use a name wherever you would use any other kind of point identifier.

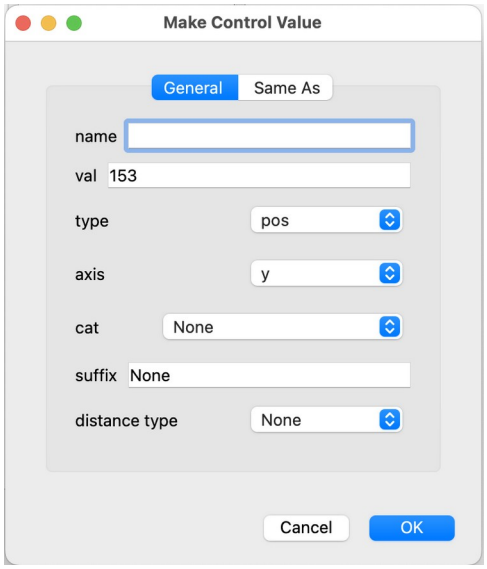
When you name more than one point, you create a **named set**, which you can use in your code wherever you can supply a list of points. You can also pass a named set to a macro as a parameter —a place where sets are otherwise not allowed.

☞ At present there is no way to enter a named set via the GUI: you must type or paste it into the YAML code.

Control Values

CVs are of two kinds: **position** (pos) and **distance** (dist). Position CVs specify the location of a point on the current axis (x or y); distance CVs specify the distance of one point from another—again on the current axis. In hinting for modern displays, position-type CVs are used more often than distance-type CVs.

The easiest way to add a CV is via the "Control Value" button on the toolbar. To use it, select one point for a "pos" CV or two points for a "dist" CV, then click the button or type "C." This dialog box will appear:



The box already has the most essential information filled in: the position or distance in the "val" box; the type of CV ("pos" or "dist"), and the axis ("x" or "y"). The only information you absolutely must supply is the CV's name (it should contain no spaces, but it can contain numbers, hyphens and underscores: make the name descriptive but not verbose).

Much of the information you supply in this box is used to filter the list of control values when you need to choose one from a menu. For example, when the current axis is "y," you only see CVs for the "y" axis. The more information you supply, the fewer choices you'll confront while hinting a glyph.

type Required. Whether the CV is for positions or distances.

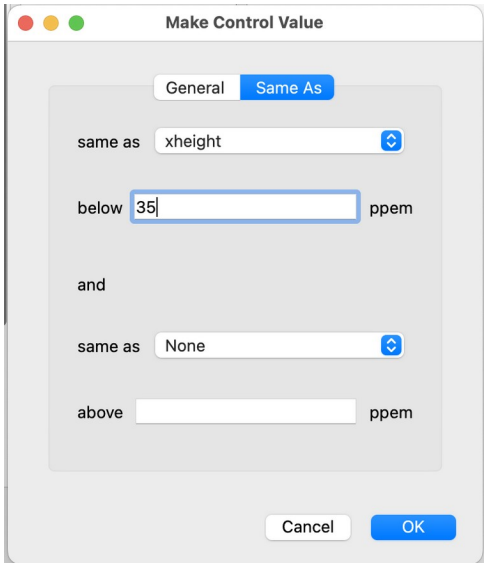
axis Required. Which axis the CV is intended for.

cat Optional. The unicode category of the characters that the CV will be applied to (e.g. "Letter, uppercase"). If supplied, the CV will only be offered for a character in that category (see below).

suffix Optional. Glyph names often have suffixes: for example, small caps commonly have names ending in ".sc" (e.g. "a.sc" for "small cap a"). If a suffix is supplied here, the CV will only apply to chararacters whose names have that suffix.

distance type Optional. This property is not for filtering CVs, but rather for affecting the behavior of hints. If a hint's distance type is not otherwise specified, this value is used. This option will rarely have an effect in the code, but it may be useful as a reminder of how the CV should be used.

Click over to the "Same As" tab if you want to make this CV equal to another (or others) at certain resolutions.



You can make the CV equal to a CV below a certain resolution and equal to another CV above another resolution. Just pick the CV you want to set this one equal to from the "same as" list and enter a resolution in the "below ppem" and/or the "above ppem" box. (You can change this number later in the "Font Info" window.)

The "below ppem" feature is useful for making stems of slightly different width and edges of slightly different height appear consistent at low resolutions. For example, xheight and xheight-overshoot should normally be set equal to each other below 30 or 40 ppem (experiment to find the right ppem number for your font).

Other, more advanced Control Value properties can be managed in the "Font Info" window, discussed below.

Categories

Every character with a Unicode code point belongs to a category assigned by Unicode. To see them, choose `Set category` from the context menu. Ygt uses these categories as filters for CVs: when you assign a category to a CV, that CV only appears on the menu for glyphs with matching categories (but any CV can be entered in the editing panel). This feature keeps the `Set control value` context menu manageable when you have many CVs.

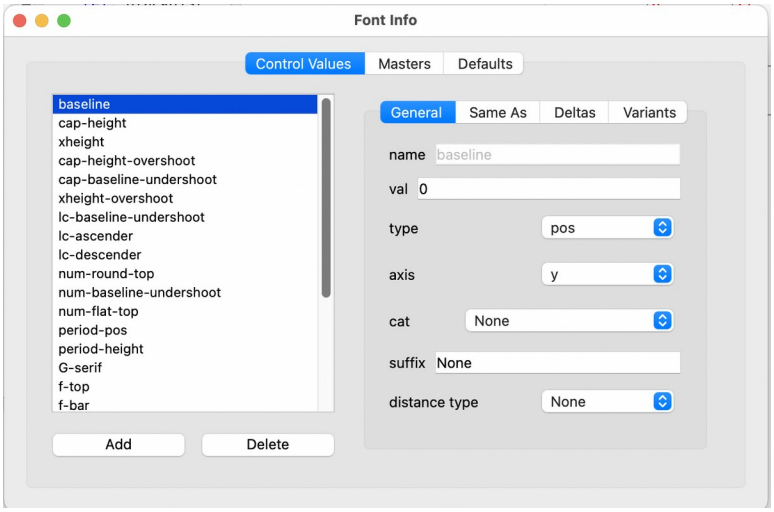
You can override these categories whenever it is useful to do so. ☞ The categories you choose are not recorded in the font, and they have no effect at all except on the selection of CVs in the menus. For example, the "yen" sign (U+00A5) is categorized as "Symbol, currency," but its shape is that of a capital Y with bars, and so it is useful to apply to it CVs for capital letters. Simply select `Letter, uppercase` from the `Set category` menu, and all capital CVs will be available for

that glyph. Likewise, if you use the Unicode Private Use Area, you'll find that Unicode classifies all the glyphs there as "Other, private use"—not very useful! But you can assign any category you like to Private Use glyphs for the purpose of hinting them.

Some groupings of characters do not have Unicode categories—small caps, for example. For these, use the "suffix" property of your CVs.

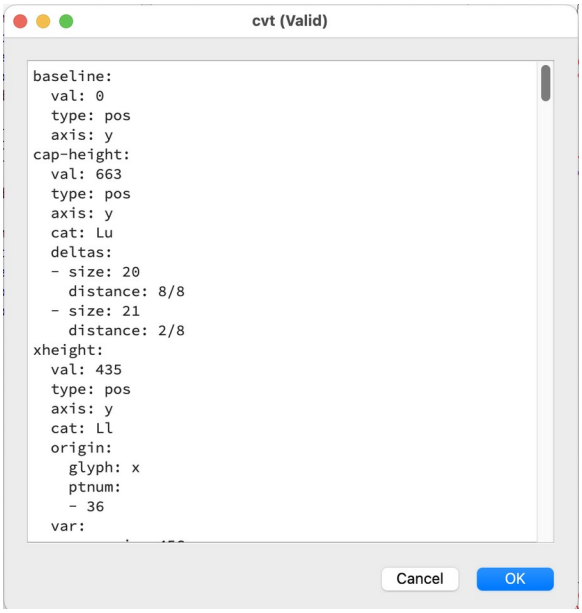
Editing Control Values

The Control Value Table (cvt) and related information can be edited in the "Font Info" window. To bring it up, select `File→Font Info` (Ctrl-I).



In the "Control Values" tab of this window, CVs are listed in the panel on the left; on the right is the same panel that appears in the "Make Control Value" dialog, but with one or two more tabs.

To edit any CV, double-click its name. Any value associated with the CV can be changed *except for its name*. The reason the name is not editable here is to guard against disconnecting the CV from the hints that use it. You can rename the CV, if you really want to do so, by bringing up the "cvt" dialog (`Code→Edit cvt`):



You can change anything you like in this dialog. When the YAML code for the CVT is valid, the editor's background is white. When invalid, it is pink. The background will frequently be pink while you're typing (because a keyword is incomplete or you have not yet typed a required value). If it is pink when you think you have finished typing, you'll have to investigate the cause: bad indentation, for example, or an unrecognized value. Here are the parts of a cvt entry:

name — Required. Every CV must have a name; Ygt's collection of CVs is organized with these names as keys. Names should contain no spaces, but they may contain numbers, hyphens, and underscores.

val — Required. As in the discussion of the "Make Control Value" dialog (above).

type — Required. As in the discussion of the "Make Control Value" dialog.

axis — Required. As in the discussion of the "Make Control Value" dialog.

cat — Optional. As in the discussion of the "Make Control Value" dialog.

col — Optional. The distance type, as in the discussion of the "Make Control Value" dialog.

same-as — Optional. As in the discussion of the "Make Control Value" dialog. This may contain one or two keys, "above" and "below." The "above" and "below" keys, when present, must contain two keys: "ppem," the resolution above or below which the present CV is set equal to another, and "cv," the CV that this CV will be set equal to.

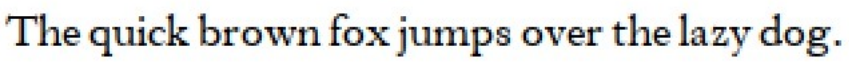
var — Optional. Variant CVs in a variable font. Discussed below.

deltas — Optional. Deltas for adjusting CVs at particular sizes. Discussed below.

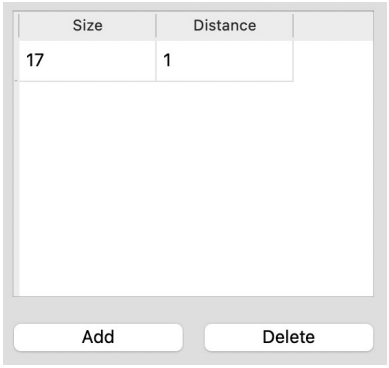
origin When a CV is supplied by Ygt, or when you create it using the "Make Control Value" dialog, the program records the glyph name and point index for the point on which the CV was based. This is used when the program automatically generates data for the cvar table; it should not normally be edited.

Control Value Deltas

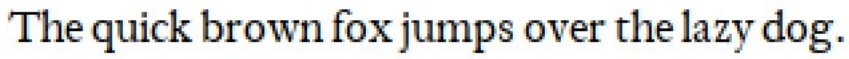
The "Deltas" panel lets you apply size-specific adjustments to your CVs. For example, at 17ppem, Elstob's lowercase letters look a little squashed:



You can fix this by adjusting the "xheight" CV. Simply add an entry in the "Deltas" panel for "xheight" and edit the "Size" and "Distance" cells:



This adds one pixel to the "xheight" CV (note that the "xheight-overshoot" CV, which is set to be the same as "xheight" at this resolution, is automatically adjusted as well):



You can also adjust a CV by a fractional amount, but Ygt may slightly adjust the "distance" value to conform to the requirements of the TrueType engine. An alternative is to use the notation "distance/increment," where "distance" is a number from -8 to 8 (note that a value of zero here disables the delta), and "increment" is one of 2, 4, 8, 16, 32, 64 (standing for one half of a pixel, one quarter, and so on).

For example, a distance of "8/8" is the same as "1," and a distance of "-1/16" causes 1/16 of a pixel to be subtracted from the CV.

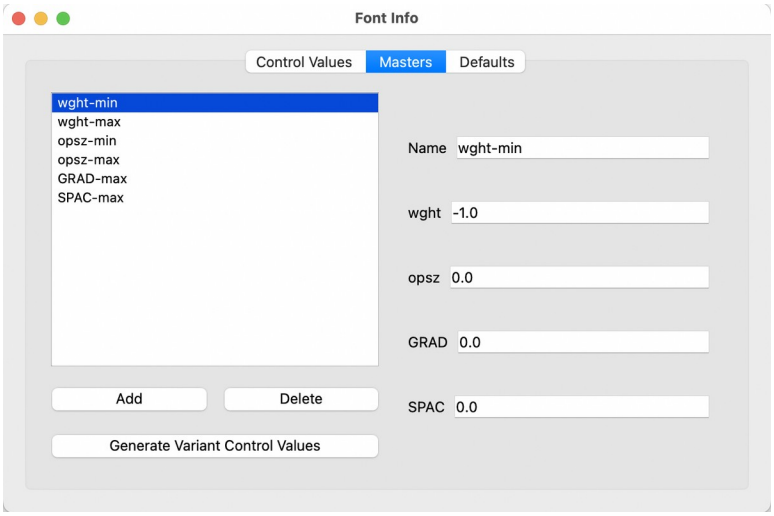
"Size" must be a number from 9 to 56 inclusive. It is possible to add deltas for other sizes by editing the "prep" code directly, but in general it is unnecessary to add deltas above 56 ppem and useless to add them below 9.

You will see the result of any added delta immediately in the preview panel. When using CV deltas in a variable font, carefully review all instances: CV deltas are not interpolated as CVs themselves are, but are applied *after* interpolation.

Variant Control Values

In a variable font, CVs need to be interpolated whenever outlines are interpolated. For example, in the Elstob font, the value of the xheight CV is 435, but in the "opsz-min" master the xheight is 456 while in the "opsz-max" master it is 414. The cvt entry "xheight" is 435, and the alternative values 456 and 414 are stored in the cvar (CVT Variations) table.

When you first open a variable font, Ygt takes a few seconds to generate a list of masters and search out values for the cvar table. The masters correspond in function to the masters you created when you designed the font. Their coordinates may also match those of the design masters, and you can supply names, if you like, that match those of the design masters. This is the "Masters" tab of the "Font Info" window for the Elstob font:

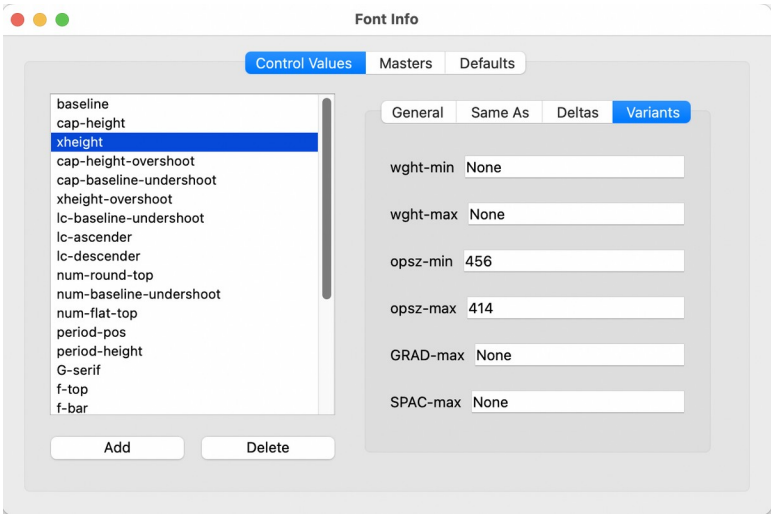


The values for each axis are normalized such that the minimum value (when it is less than the default) is -1.0 , the maximum value is 1.0, and the default value is 0.0. For example, Elstob's "weight" axis has a minimum of 200, a maximum of 800, and a default of 400. In normalized notation, 200 = -1.0, 400 = 0.0, and 800 = 1.0. Notice that it doesn't matter if the difference between the default and the minimum differs from that between the default and the maximum: the normalized values are still -1.0, 0.0, and 1.0.

For most masters, one of the axis values will be the maximum or minimum while the others will be the default. However, this is not always the case. You may wish to create a master with more than one non-default value, and you may type any number between -1.0 and 1.0 in any of the axis blanks.

You can add masters if you like, or delete unused ones. (If you don't delete unused masters, Ygt will ignore them when it compiles the font's TrueType instructions.)

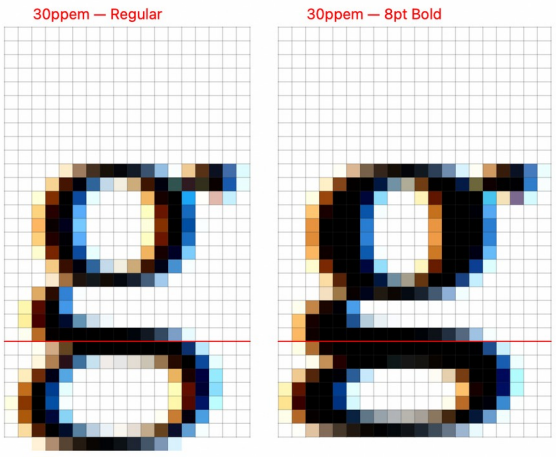
Variant CVs can be edited in the "Variants" panel of the "Control Values" tab:



For each master, you can enter either an integer value or "None" (or leave it blank) if the master has the same CV value as the default value in the cvt.

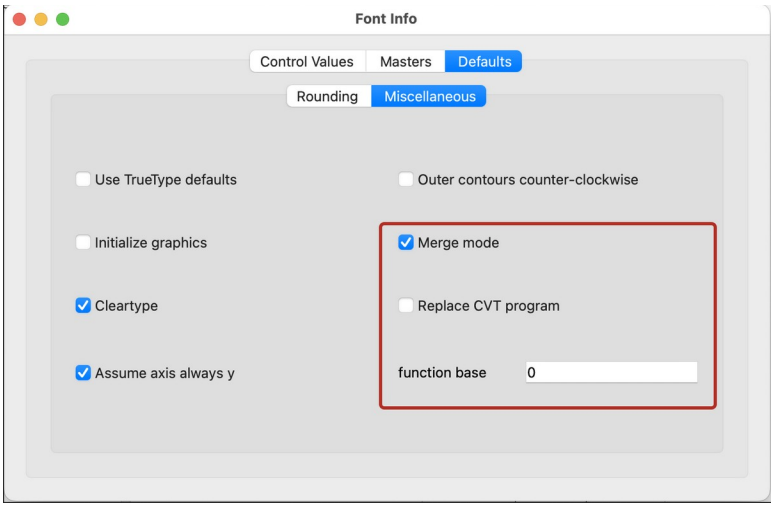
An alternative to entering values here is to press the "Generate Variant Control Values" button in the "Masters" tab. The program will take a few seconds to survey key points (for Control Values with "origin" properties) and fill in variant Control Values.

Test how your hinting behaves at various points on your axes by selecting different instances from the `Preview` menu. For example, below are previews of the Elstob **g** for the Regular and 8pt Bold instances at 30 ppem. The same set of hints produced both glyphs.



Merge mode

On the “Miscellaneous” pane of the “Defaults” panel of the Font Info window, you can shift Ygt into **merge mode**, in which it adds its hints to those already in a font. Normally, Ygt clears any existing hints before installing its own, but in merge mode it clears only those glyph programs for which it has replacements. This feature is useful if, for example, you are mostly satisfied with the output of an auto-hinter like ttfautohint but would like to manually hint several glyphs.



In merge mode, you must decide whether to replace the font’s existing CVT (or “prep”) program or retain it, appending Ygt’s generated CVT programming. The former is usually the safer course, but if you’re confident that a font’s existing hints do not rely on its existing CVT program, or that you can reproduce its functionality, it may make sense to replace.

You may also need to supply a **function base**—the number at which to begin numbering Ygt’s functions. When the function base is zero or not given, Ygt guesses at the appropriate number by consulting the “maxInstructions” setting in the font’s “maxp” table. This will be correct for a font autohinted by ttfautohint, but not (for example) for VTT, which skips over several numbers in numbering its functions. You can determine the function base by decompiling the font with ttx and examining the last function definition (FDEF) in the <fpgm> table.

Alternatively, you can simply set the function base to a higher number than autohinters typically use—150, for example.

Merge mode only works with TrueType font files: when you export hints to a UFO, these settings are ignored.

Color schemes

When Ygt detects that the display it is running on is employing a dark theme, it shifts to its own dark theme, using system colors where appropriate and its own colors otherwise. For the most part the choice is fully automated. But for the preview panels you can choose a theme by selecting `Auto` (the default), `Black on white`, or `White on black` from the `Preview→Theme` menu. Thus you can test your hinted glyphs under various conditions.