

Glyphs 3 Handbook



You are reading the Glyphs Handbook from April 2024 for the application version 3.2. You can find the latest version of the Handbook at **handbook.glyphsapp.com**

© 2011–2024 Glyphs GmbH

Written by Rainer Erich Scheichelbauer, Georg Seifert, and Florian Pircher.

Thanks to Nathalie Dumont, Jeff Kellem, Rob Keller, Toshi Omagari, and Claus Eggers Sørensen for their invaluable input.

Contents

- 1 Glyphs, 10**
 - 1.1 Glyphs Mini, 10
 - 1.2 Community, 10
 - 1.3 Issues, 10
- 2 Create, 11**
- 3 Settings, 13**
 - 3.1 Updates, 13
 - 3.2 Appearance, 14
 - 3.3 User Settings, 15
 - 3.4 Sample Strings, 16
 - 3.5 Sharing, 17
 - 3.6 Addons, 18
 - 3.6.1 Python version, 18
 - 3.6.2 Console Output, 18
 - 3.6.3 Alternate Plugin Repositories, 18
 - 3.7 Shortcuts, 20
- 4 Edit View, 21**
 - 4.1 Drawing Paths, 21
 - 4.1.1 Draw Tool, 21
 - 4.1.2 Pencil Tool, 22
 - 4.1.3 Primitives, 22
 - 4.2 Editing Paths, 23
 - 4.2.1 Selecting Nodes and Paths, 23
 - 4.2.2 Free-form Selections, 24
 - 4.2.3 Moving Selected Nodes and Paths, 24
 - 4.2.4 Converting Nodes and Segments, 25
 - 4.2.5 Nodes in Alignment Zones, 25
 - 4.2.6 Scaling & Rotating, 26
 - 4.2.7 Aligning, 26
 - 4.2.8 Duplicating Paths, 27
 - 4.2.9 Deleting Nodes, 28
 - 4.2.10 Opening and Closing Paths, 28
 - 4.2.11 Cutting Paths, 29
 - 4.2.12 Re-segmenting Outlines, 29
 - 4.2.13 Controlling Path Direction, 31
 - 4.2.14 Extremes & Inflections, 32
 - 4.2.15 Duplicate Nodes, 32
 - 4.2.16 Focusing & Locking, 32
 - 4.3 Graphic Attributes, 33
 - 4.3.1 Creating Strokes, 34
 - 4.3.2 Masking, 34
 - 4.4 Anchors, 35
 - 4.4.1 Adding, Editing, & Removing Anchors, 35
 - 4.4.2 Mark to Base Positioning, 36
 - 4.4.3 Mark to Mark Positioning, 36
 - 4.4.4 Curvise Attachment, 36
 - 4.4.5 Ligature Carets, 37
 - 4.4.6 Contextual Mark Attachment, 37
 - 4.5 Guides, 38
 - 4.5.1 Magnetic Guides, 38
 - 4.5.2 Local & Global Guides, 38
 - 4.5.3 Glyph-Specific Undo History, 40
 - 4.6 Info Box, 40
 - 4.6.1 Horizontal Layout, 41
 - 4.6.2 Vertical Layout, 41
 - 4.7 Glyph Display, 41
 - 4.7.1 Zooming, 41
 - 4.7.2 Panning, 42
 - 4.7.3 View Options, 42
 - 4.7.4 Glyph & Layer Colors, 43
 - 4.8 Background, 44

- 4.9 Entering Text, 45
 - 4.9.1 Text Tool, 45
 - 4.9.2 Writing Direction, 46
 - 4.9.3 Text Preview, 46
 - 4.9.4 Sample Strings, 47
- 4.10 Measuring, 48
 - 4.10.1 Measuring with the Info box, 48
 - 4.10.2 Measurement Tool, 49
 - 4.10.3 Measurement Guides, 50
 - 4.10.4 Measurement Line, 51
- 4.11 Annotating, 52
 - 4.11.1 Annotation Cursor, 52
 - 4.11.2 Annotation Text, 52
 - 4.11.3 Annotation Arrow, 53
 - 4.11.4 Annotation Circle, 53
 - 4.11.5 Plus & Minus Annotations, 53
- 4.12 Images, 54
 - 4.12.1 Adding Images, 54
 - 4.12.2 Manipulating Images, 54
- 4.13 Previewing & Testing, 55
 - 4.13.1 Previewing Kerning, 55
 - 4.13.2 Previewing Masters, 55
 - 4.13.3 Previewing OpenType Features, 55
 - 4.13.4 Previewing Interpolated Instances, 56
 - 4.13.5 Previewing on macOS, 57
 - 4.13.6 Previewing in Adobe Applications, 58
 - 4.13.7 Previewing in Web Browsers, 58
- 5 Palette, 60**
 - 5.1 Dimensions, 60
 - 5.2 Fit Curve, 60
 - 5.3 Layers, 61
 - 5.3.1 Master Layers, 61
 - 5.3.2 Backup Layers, 61
 - 5.3.3 Special Layers, 62
- 5.4 Transformations, 62
 - 5.4.1 Transformation Origin, 63
 - 5.4.2 Mirroring, 63
 - 5.4.3 Scaling, 64
 - 5.4.4 Rotating & Slanting, 64
 - 5.4.5 Aligning, 64
 - 5.4.6 Boolean Operations, 64
- 6 Font View, 66**
 - 6.1 Viewing Glyphs, 66
 - 6.1.1 Grid View, 66
 - 6.1.2 List View, 67
 - 6.2 Managing the Glyph Set, 67
 - 6.2.1 Adding New Glyphs, 67
 - 6.2.2 Copying Glyphs Between Files, 69
 - 6.2.3 Deleting Glyphs, 70
 - 6.3 Glyph Properties, 71
 - 6.3.1 Glyph Name, 71
 - 6.3.2 Metrics, 72
 - 6.3.3 Kerning Groups, 72
 - 6.3.4 Exports, 72
 - 6.3.5 Color Label, 72
 - 6.3.6 Tags, 73
 - 6.3.7 Unicode, 73
 - 6.3.8 Production Name, 74
 - 6.3.9 Script, 75
 - 6.3.10 Category & Subcategory, 75
 - 6.3.11 Case, 75
 - 6.3.12 Writing Direction, 75
 - 6.3.13 Sort Name, 76
 - 6.3.14 ID, 76
 - 6.3.15 Char, 76
 - 6.3.16 Note, 76
 - 6.3.17 Components, 76
 - 6.3.18 Last Changed, 76
 - 6.4 Names & Unicode, 77
 - 6.4.1 Glyph Info Database, 77
 - 6.4.2 Naming Glyphs, 78
 - 6.4.3 Glyph Naming Rules, 78

- 6.4.4 Copy Glyph Names, 80
- 6.4.5 Renaming Glyphs, 81
- 6.4.6 CID Mapping, 81
- 6.5 Batch-Processing, 82
 - 6.5.1 Selecting Glyphs, 82
 - 6.5.2 Batch Commands, 82
 - 6.5.3 Batch-
 - Renaming Glyphs, 83
 - 6.5.4 Filters, 84
 - 6.5.5 Palette, 84
 - 6.5.6 Plug-ins & Scripts, 84
- 6.6 Filtering Font View, 84
 - 6.6.1 Search Field, 85
 - 6.6.2 Categories, 85
 - 6.6.3 Languages, 85
 - 6.6.4 Smart Filters, 86
 - 6.6.5 List Filters, 88
 - 6.6.6 Managing Filters, 89
 - 6.6.7 Custom Categories & Languages, 89
- 6.7 Glyph Order, 89
- 6.8 Images, 90
- 7 Font Info, 91**
 - 7.1 Font, 91
 - 7.1.1 Family Name, 91
 - 7.1.2 Units per Em, 92
 - 7.1.3 Version, 93
 - 7.1.4 Creation Date, 93
 - 7.1.5 Axes, 93
 - 7.1.6 Custom Parameters, 94
 - 7.2 Masters, 94
 - 7.2.1 Managing Masters, 94
 - 7.2.2 General, 95
 - 7.2.3 Axes Coordinates, 95
 - 7.2.4 Metrics & Alignment Zones, 95
 - 7.2.5 Stems, 97
 - 7.2.6 Custom Parameters, 97
 - 7.2.7 Number Values, 98
 - 7.3 Exports, 98
 - 7.3.1 Active, 99
 - 7.3.2 Style Name, 99
 - 7.3.3 Weight & Width, 99
 - 7.3.4 Axes Coordinates, 100
 - 7.3.5 Style Linking, 100
 - 7.3.6 Custom Parameters, 101
 - 7.4 Features, 101
 - 7.4.1 Editing Feature Code, 102
 - 7.4.2 Automatic Feature Code, 102
 - 7.4.3 Manual Feature Code, 103
 - 7.4.4 Naming Stylistic Sets, 105
 - 7.4.5 Implicit Features, 105
 - 7.4.6 Export-Specific Features, 106
 - 7.5 Other Settings, 107
 - 7.5.1 Grid Spacing & Subdivision, 107
 - 7.5.2 Keyboard Increments, 107
 - 7.5.3 Use Custom Naming, 107
 - 7.5.4 Disable Automatic Alignment, 108
 - 7.5.5 Keep Alternates Next to Base Glyph, 108
 - 7.5.6 File Format Version, 108
 - 7.5.7 Font Type, 109
 - 7.6 Notes, 109
- 8 Import & Export, 110**
 - 8.1 Exporting Font Files, 110
 - 8.1.1 OpenType Export, 110
 - 8.1.2 Exporting Variable Fonts, 112
 - 8.1.3 Exporting UFO, 113
 - 8.1.4 Exporting Metrics, 114
 - 8.2 Source Formats, 114
 - 8.2.1 Glyphs File, 114
 - 8.2.2 Glyphs File Package, 115
 - 8.2.3 Unified Font Object, 115
 - 8.3 Opening Font Files, 116
 - 8.3.1 Font File Importing Behaviors, 116

8.3.2	Opening TrueType Font, 117	10	Reusing Shapes, 133
8.3.3	Importing Multiple Fonts Files into a Glyphs File, 117	10.1	Components, 133
8.3.4	Importing OpenType Features, 117	10.1.1	Building Composites, 133
8.3.5	Importing PostScript Hints, 117	10.1.2	Turning Paths into Components, 134
8.4	Importing Font Data, 117	10.1.3	Recipes, 134
8.4.1	Importing Outlines, 117	10.1.4	Editing Components, 135
8.4.2	Importing Metrics, 118	10.1.5	Moving between Base Glyphs and Composites, 135
8.4.3	Importing Feature Files, 119	10.1.6	Component Placeholders, 136
8.5	Vector Drawing Applications, 119	10.1.7	Anchors, 136
8.5.1	Adobe Illustrator, 120	10.1.8	Automatic Alignment, 137
8.5.2	Affinity Designer, 120	10.1.9	Locking Components, 139
8.5.3	Sketch, 120	10.1.10	Decomposing, 140
8.6	File Format Interoperability, 120	10.1.11	Combining Paths and Components, 140
8.7	Projects, 121	10.1.12	Nesting Components, 140
8.7.1	Setting up a Project, 121	10.1.13	Preferred Marks for Glyph Composition, 141
8.7.2	Exporting a Project, 122	10.1.14	Underscore Components, 141
9	Spacing & Kerning, 123	10.2	Smart Components, 141
9.1	Spacing, 123	10.2.1	Setting up Smart Glyphs, 142
9.1.1	Info Box, 123	10.2.2	Using Smart Components, 144
9.1.2	Spacing Shortcuts, 124	10.2.3	Width & Height Properties, 144
9.1.3	Metrics Keys, 124	10.2.4	Smart Handles, 145
9.1.4	Metrics Keys & Automatic Alignment, 126	10.3	Corner Components, 145
9.2	Kerning, 126	10.3.1	Creating Corner Glyphs, 145
9.2.1	Kerning Modes, 127	10.3.2	Using Corner Components, 146
9.2.2	Info Box, 127	10.3.3	Adjusting the Entry Point of a Corner, 148
9.2.3	Kerning Shortcuts, 127	10.3.4	Adjusting the Exit Point of a Corner, 148
9.2.4	Kerning Groups, 128	10.3.5	Adjusting the Width and Height, 149
9.2.5	Kerning Group Exceptions, 129		
9.2.6	Kerning Window, 129		
9.2.7	Manual Kerning Code, 131		

- 10.3.6 Closed Paths in Corners, 149
- 10.3.7 Extra Nodes, 150
- 10.4 Cap Components, 151
 - 10.4.1 Creating Cap Glyphs, 151
 - 10.4.2 Using Cap Components, 151
- 10.5 Segment Components, 152
 - 10.5.1 Creating Segment Glyphs, 152
 - 10.5.2 Using Segment Components, 152
- 10.6 Brushes, 153
 - 10.6.1 Creating Brush Glyphs, 153
 - 10.6.2 Using Brushes, 154
- 10.7 Pixel Tool, 154
 - 10.7.1 Setup, 155
 - 10.7.2 Drawing Pixels, 155
 - 10.7.3 Pixel Shape, 155
- 11 Interpolation, 156**
 - 11.1 Interpolation Applications, 157
 - 11.2 Setting up Axes, 158
 - 11.3 Setting up Masters, 158
 - 11.3.1 Axes Coordinates, 159
 - 11.3.2 Minimal Multiple Masters Setup, 159
 - 11.3.3 Elaborate Multiple Masters Setups, 160
 - 11.4 Setting up Instances, 160
 - 11.4.1 Static Instances, 161
 - 11.4.2 Variable Font Settings, 161
 - 11.4.3 Subsetting the Designspace, 161
 - 11.5 Outline Compatibility, 162
 - 11.5.1 Identifying Incompatible Outlines, 162
 - 11.5.2 Correcting Path Direction, 162
 - 11.5.3 Reordering Shapes, 163
 - 11.5.4 Master Compatibility, 163
 - 11.6 Intermediate Layers, 165
 - 11.6.1 Intermediate Layer Setup, 165
 - 11.6.2 Virtual Masters, 165
 - 11.7 Switching Shapes, 166
 - 11.7.1 Alternate Layers, 166
 - 11.7.2 Replacing Glyphs at Export, 168
 - 11.7.3 Conditional Glyph Substitutions, 168
 - 11.8 Editing Multiple Masters, 169
 - 11.8.1 Select All Layers Tool, 169
 - 11.8.2 Show All Masters, 169
 - 11.8.3 Keep Layer Selection in Sync, 169
 - 11.9 Working with Multiple Fonts, 169
 - 11.9.1 Grouping Fonts into Families, 169
 - 11.9.2 Glyphs Files, Masters, & Instances, 170
 - 11.9.3 Compare Fonts, 170
 - 11.9.4 Compare Family, 171
 - 11.10 Variable Font Options, 172
 - 11.10.1 Variable Font Origin, 172
 - 11.10.2 Axis Location, 173
 - 11.10.3 Axis Mappings, 173
 - 11.10.4 Style Attributes Table, 175
- 12 Filters, 177**
 - 12.1 Applying Filters, 177
 - 12.1.1 Filter Menu, 177
 - 12.1.2 Filters as Custom Parameters, 177
 - 12.2 Built-in Filters, 179
 - 12.2.1 Shape Order, 179
 - 12.2.2 Extrude, 179
 - 12.2.3 Hatch Outline, 180
 - 12.2.4 Offset Curve, 181

- 12.2.5 Roughen, 182
- 12.2.6 Round Corners, 183
- 12.2.7 Rounded Font, 184
- 12.2.8 Transformations, 184
- 12.2.9 Add Extremes, 187
- 12.2.10 Remove Overlap, 187
- 13 Feature Code, 188**
 - 13.1 Tokens, 188
 - 13.1.1 Number Value Tokens, 188
 - 13.1.2 Glyph Property Tokens, 188
 - 13.1.3 Glyph Predicate Tokens, 189
 - 13.2 Conditional Feature Code, 190
 - 13.3 Variable GPOS, 191
 - 13.4 Deleting Glyphs, 192
 - 13.5 Standalone Lookups, 192
 - 13.6 Optional Closing Lookup Name, 193
 - 13.7 Multiple Languages Syntax, 193
 - 13.8 Multiple Substitution With Classes, 194
 - 13.9 Lookup Flags, 194
- 14 PostScript Hinting, 195**
 - 14.1 Font-Wide Hints, 196
 - 14.1.1 Standard Stems, 196
 - 14.1.2 Alignment Zones, 197
 - 14.1.3 Custom Parameters, 199
 - 14.2 Autohinting, 199
 - 14.2.1 Flex Hints, 199
 - 14.3 Manual hinting, 200
 - 14.3.1 Stem Hints, 201
 - 14.3.2 Ghost Hints, 202
 - 14.3.3 Hinting Multiple Masters, 203
- 15 TrueType Hinting, 204**
 - 15.1 Autohinting, 204
 - 15.2 Font-level Hints, 205
 - 15.2.1 TrueType Zones, 205
 - 15.2.2 TrueType Stems, 208
 - 15.2.3 TrueType BlueFuzz, 209
 - 15.3 Glyph-level Hints, 209
 - 15.3.1 Hinting Outlines, 210
 - 15.3.2 Pixel Size, 211
 - 15.3.3 Hint Direction, 211
 - 15.3.4 Hint Order, 211
 - 15.3.5 Show Point Indexes, 212
 - 15.3.6 Hinting Preview, 212
 - 15.3.7 Web Preview, 212
 - 15.4 Instructions, 213
 - 15.4.1 Snap (A), 213
 - 15.4.2 Stem (S), 214
 - 15.4.3 Shift (F), 217
 - 15.4.4 Interpolate (G), 219
 - 15.4.5 Delta (E), 220
 - 15.4.6 Points in Overlapping Intersections, 220
 - 15.5 Advanced TrueType Hinting, 221
- 16 Color Fonts, 224**
 - 16.1 Working with Color Fonts, 224
 - 16.1.1 Keeping the Metrics in Sync, 224
 - 16.1.2 Previewing Color Fonts, 224
 - 16.1.3 Exporting Color Fonts, 224
 - 16.2 Layered Color Fonts, 225
 - 16.2.1 Initial Setup, 225
 - 16.2.2 Editing Color Layers, 226
 - 16.2.3 Exporting, 226
 - 16.3 COLR/CPAL Fonts, 226
 - 16.3.1 Defining the Color Palette, 227

16.3.2	Master Layer as Fallback, 227	17	Extensions, 235
16.3.3	Color Palette Layers, 227	17.1	Plugin Manager, 235
16.3.4	Exporting, 228	17.2	Scripts, 236
16.4	sbix Fonts, 229	17.2.1	Run Scripts, 236
16.4.1	Standard Bitmap Graphics, 229	17.2.2	The Scripts Folder, 236
16.4.2	Preparing Images, 229	17.2.3	Creating Scripts, 237
16.4.3	Adding Images to Glyphs, 230	17.3	Plug-ins, 237
16.4.4	Exporting, 231	17.3.1	Installing Plug-ins, 238
16.5	SVG Color Fonts, 231	17.3.2	Creating Plug-ins, 238
16.5.1	Converting to SVG, 232	18	Appendix, 240
16.5.2	Importing Existing SVG Files, 232	18.1	Regular Expressions, 240
16.5.3	Creating SVG Glyphs, 232	18.2	Custom Feature Code Snippets, 241
		18.3	Automatic Feature Generation, 242
		18.4	Custom Parameters, 247

1 Glyphs



Glyphs 3 is a professional Mac application for creating OpenType fonts. It allows you to draw, edit and test letterforms in the context of words, and helps you manage all aspects of modern font production, including the efficient reuse of recurring shapes and the creation of large font families. You can extend the functionality of Glyphs with plug-ins and Python scripts, many of which are freely available and accessible from within the application.

1.1 GLYPHS MINI

As an alternative to the pro version, there is a slimmed-down and more affordable version of the application called Glyphs Mini. It lacks some features of the pro app, notably interpolation and extensibility, and has a narrower range of export formats. The Glyphs Mini handbook is available at glyphsapp.com/learn.

1.2 COMMUNITY

Glyphs is also a vibrant community. On our website you will find many ways to get involved. We encourage you to use the Forum to ask questions and discuss issues with other users and the developers. The Learn section has a large collection of tutorials, screencasts, online classes and other educational material. Keep an eye out for workshops, conferences, and the latest updates in the News section. If you are planning a workshop, organizing a Glyphs-related event, or releasing a plug-in you have developed, we will be happy to share your contribution on glyphsapp.com.

1.3 ISSUES

If the application crashes, submitting the crash report dialog that appears the next time you launch the application will help us troubleshoot the problem that caused the crash. If you can, please describe steps for reliably reproducing the issue in the forum. The most likely cause of a crash is an incorrect interaction with a plug-in. You can disable plug-ins by holding down the Option and Shift keys when the application starts.

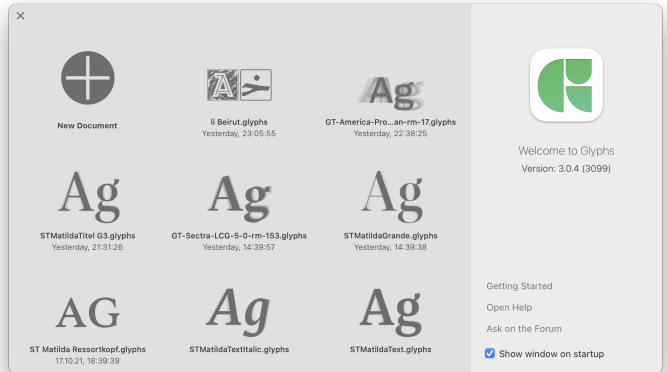
An issue you encounter may have been fixed in the latest beta version of the application. See section 3.1, 'Updates' (p. 13) for details.


2 Create

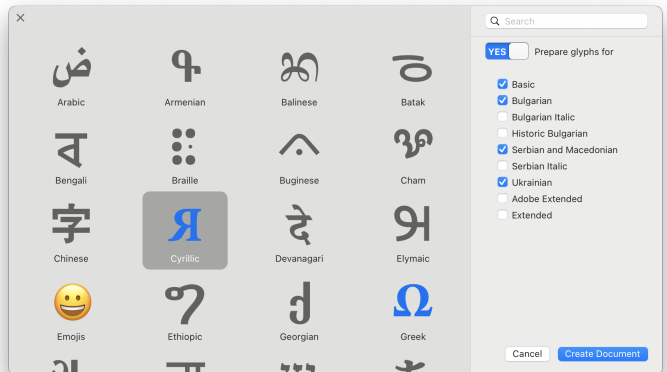
Glyphs shows the Start Window when opened without documents: The right side contains information about the Glyphs version and links to the Glyphs website. Uncheck *Show window on startup* to hide the window on future launches.

The left side shows a list of recently opened documents. Double-click a document to open it. Open the Start Window anytime from the menu bar: *Window > Start Window*.

Tip: Control-click or right-click a document and choose *Show in Finder* from the context menu to reveal the file in Finder.



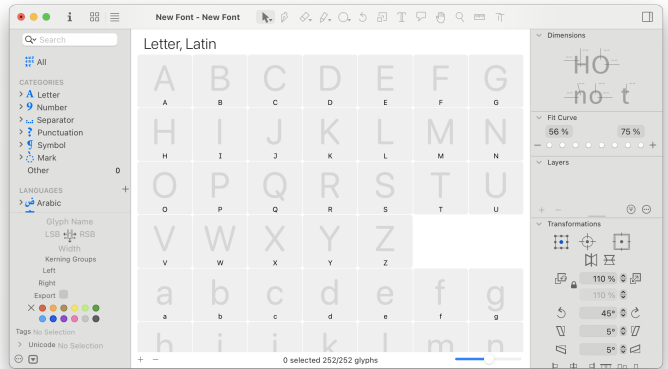
Click the plus  button to create a new Glyphs document. The Start Window then lists a set of scripts. These scripts are also shown when there are no recent files.



Click a script and include it in the new document by toggling *No* to *Yes* in the top-right of the window. Included scripts appear

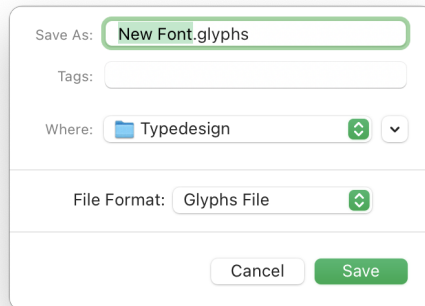
with a colored icon in the list. Customize the included glyphs by toggling the respective checkboxes on the right side. Search for scripts by name using the search field in the top-right corner of the window.

Create a document containing empty glyphs for all selected scripts using the *Create Document* button in the lower right:



Save the document with *File > Save* (Cmd-S). Give the file a name and pick a folder in which to save it. Glyphs offers three file format options. See section 8.2, ‘Source Formats’ (p. 114) for details, or pick the default *Glyphs File* for a start.

A Glyphs document can be changed to a different file format at any time with *File > Save As...* (Cmd-Shift-S).



Create a new document at any time with *File > New from Glyph Sets...* The list remembers the previous selection of scripts and glyph sets. Quickly create a new document with the basic ASCII glyphs by choosing *File > New* (Cmd-N).

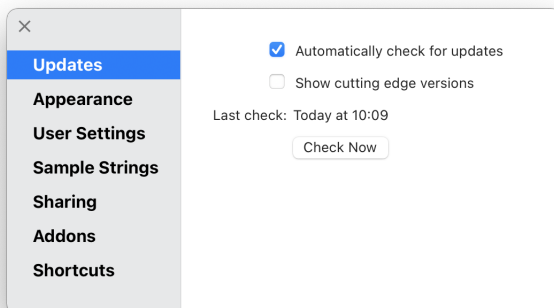
See the *Getting Started* tutorials on the Glyphs website¹ for an introduction to working with Glyphs.

1 glyphsapp.com/learn/recommendation:get-started

3 Settings

Choose *Glyphs > Settings...* from the menu bar (or press Cmd-Comma) to open the settings window. On macOS versions before macOS 13 ‘Ventura’, the menu item is called *Preferences*.

3.1 UPDATES

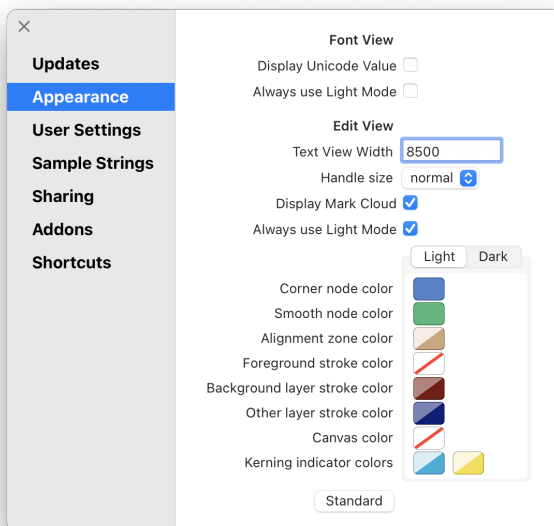


The *Updates* section controls how Glyphs is updated. Check for a new version of Glyphs by clicking on the *Check Now* button. If one is available, Glyphs will offer to download the latest version and show a list of the new additions and changes.

Automatically check for updates will look periodically for new updates. We recommend leaving this option selected.

Choose *Show cutting edge versions* to download beta versions of Glyphs. Beta versions are released frequently, providing bug fixes and early feature releases. Because of their frequency, beta versions are not as thoroughly tested as release versions. Therefore, we recommend working with copies of font files when trying a Glyphs beta. Revert to the latest stable version of Glyphs by re-downloading it from glyphsapp.com/buy. Downloading a new version or reverting to an old version of Glyphs will *not* reset the settings.

3.2 APPEARANCE



The *Appearance* section defines the visual appearance of Font View and Edit View.

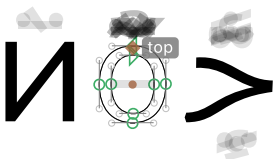
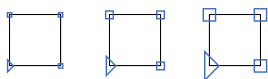
Display Unicode Value controls whether glyphs with Unicode values show a simplified Unicode icon or display the code point in the bottom-right of the glyph cell.










Always use Light Mode is available for both Font View and Edit View. These options allow Glyphs to run using the dark appearance of the system (configurable from System Settings) while still displaying the window content with a light appearance.

Text View Width controls the line length of text in Edit View. The value is measured in thousandths of an em. It is independent of the units per em (UPM) of the font.

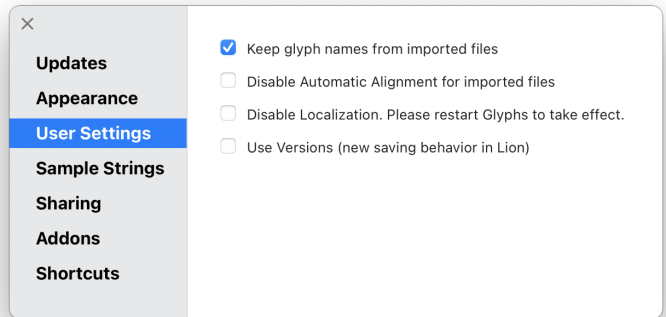
Handle size controls the size of the points, such as on-curve nodes, control handles, and anchors. Smaller sizes present a cleaner user interface; larger sizes are easier to see and click-select.

Display Mark Cloud shows a superimposition of the combining marks of a glyph. A ‘mark cloud’ appears for all glyphs when an anchor is selected. The glyph info database defines the marks shown for a glyph and its anchors. (See section 6.4.1, ‘Glyph Info Database’ (p. 77) for more on the glyph info database.) Disable this option to suppress the display of mark clouds.



Various colors can be configured for both the light and the dark system appearance. Available customization options are the color of corner nodes , smooth nodes , alignment zones , the color of strokes on the foreground layer , the background layer , and other layers  (made visible by clicking on the eye  icon in the layers list), the color of the Edit View canvas, and the kerning indicator colors (by default light blue  for negative kerning and yellow  for positive kerning). Clicking the *Standard* button resets all colors to their default values.

3.3 USER SETTINGS



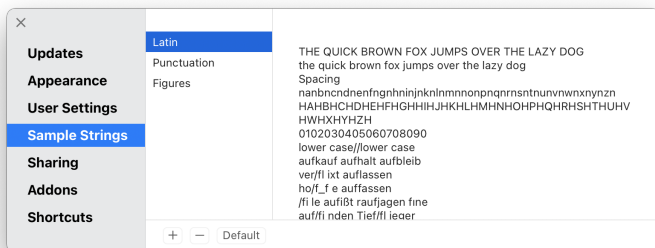
Keep Glyph Names from Imported Files maintains the names of the glyphs when opening a file in Glyphs instead of forcing the default naming scheme. This option is useful for workflows that rely on a particular naming scheme, such as exchanging files with other apps.

Disable Automatic Alignment for Imported Files disables the automatic positioning for all components when importing a font. See also section 10.1.8, ‘Automatic Alignment’ (p. 137).

Disable Localization keeps the user interface language of Glyphs in English, rather than the localization set in System Settings. Glyphs needs to be restarted after toggling this option for the change to take effect.

Use Versions employs the *Versions* file saving method. *Versions* automatically saves files when closing a document and allows browsing previous versions of the file. Enable this option to match the behavior of other Mac apps like Pages, or disable it to prevent autosave modifications when opening and closing a file.

3.4 SAMPLE STRINGS



The *Sample Strings* preference stores short pieces of text—sample strings—that can quickly be inserted into Edit View to review and edit glyphs. Sample strings are organized into groups. Click the plus (+) button to add a group, and click the minus (–) button to delete the selected group.

Each sample string is written on a separate line in the text field to the right. The text field is Unicode-savvy and accepts all diacritic marks and non-ASCII characters. Write \n (a backslash followed by a lowercase n) to include a line break in a sample string. Specify a glyph by typing its character or writing / (a forward slash) followed by the glyph name and a space character. This is useful for glyphs without a Unicode value or glyphs that are difficult to type. If multiple glyph names follow each other, the space character may be omitted. Use /Placeholder to insert a glyph that mirrors the currently selected glyph.

So, to write ‘¡Hola’ followed by a placeholder glyph, ‘!’, then a new line, and the text ‘second line’ write the following line of text into the field:

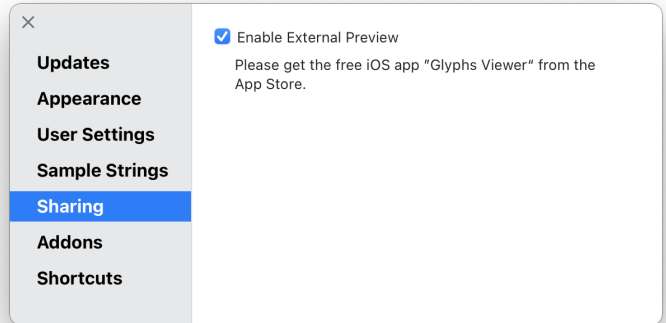
```
/exclamdown Hola /Placeholder/exclam\nsecond line
```

Click the *Default* button to reset all sample strings to the default values and delete all custom groups.

See section 4.9.4, ‘Sample Strings’ (p. 47) to find out more about the use of sample strings.

¡Hola o!
second line

3.5 SHARING



Glyphs can stream the current Edit View to a second display such as an iPhone, iPad, or iPod Touch. Choose *Enable External Preview* and download the *Glyphs Viewer* app from the App Store.¹

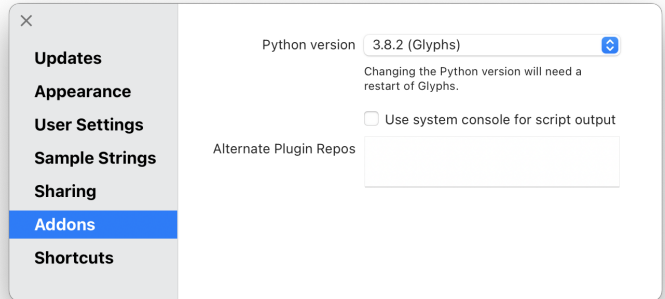
The Mac running Glyphs and the iOS device need to be connected to the same network. Open the iOS app and select the Mac with which to connect. On the selected Mac, Glyphs will ask to confirm the connection. Tap and hold anywhere on the app to return to its menu.

Glyphs Viewer displays the active Edit View. Pinch to zoom with two fingers, and pan around with a single finger gesture. Any change made to the glyphs is reflected immediately on the iOS device.

If the Glyphs Viewer app cannot find the Mac, make sure both the Mac and the iOS device are connected to the same wireless network. If the problem persists, restart both devices and try again. If they still cannot connect, try using a different network, or correct the router settings. If the Mac and the iOS device communicate to the router with different wireless standards, Glyphs Viewer cannot connect to the Mac. This issue can happen if the router is set up to simultaneously use multiple modulation standards (for example, g and n). Setting the router to either 802.11 g only or 802.11 n only may help in such cases. Once a connection has been established, a *Reset* button is added to the *Sharing* settings, which resets the list of trusted devices.

¹ appstore.com/schriftgestaltungde/glyphsviewer, or search for 'Glyphs Viewer'

3.6 ADDONS



Glyphs can be extended with plug-ins, scripts, and modules from the Plugin Manager (see p. 235). The *Addons* settings control how these extensions work and which extensions are available.

3.6.1 Python version

Python is a programming language popular among type designers and font engineers. The Python language is under active development, and new versions with new features are released regularly. Some plug-ins or scripts may require a specific Python version.

Most Glyphs plug-ins and scripts require Python. Select the version of Python that Glyphs should use for plug-ins and scripts from the *Python version* field. Note that Glyphs requires Python version 3 or later.

If no Python version is selected or the currently selected version does not work with the installed scripts and plug-ins, go to the menu bar and choose *Window > Plugin Manager > Modules* and install *Python*. Switch back to the settings window and choose the Python version labeled '(Glyphs)'.

Relaunch Glyphs for the Python version change to take effect.

3.6.2 Console Output

Use system console for script output directs the log output of plug-ins and scripts to the system console instead of the Macro Panel console (*Window > Macro Panel*, *Cmd-Opt-M*). Select this option for debugging a plug-in or script when the Macro Panel is inaccessible.

3.6.3 Alternate Plugin Repositories

A plug-in repository defines a set of plug-ins, scripts, and modules that can be installed from the Plugin Manager. In Glyphs, there is a main plug-in repository that is accessible to all Glyphs users. It provides some of the most popular Glyphs plug-ins, scripts, and modules.

Glyphs supports alternate plug-in repository URLs with HTTP Basic authentication over HTTPS, which requires a username and password to access the repository.

Local repositories are supported by adding the full file path instead of a URL to the text field.

Define alternate plug-in repositories in addition to the main repository by adding URLs pointing to plug-in repositories in the *Alternate Plugin Repos* text field, one URL per line. Alternate plug-in repositories are helpful to distribute preview versions of extensions or for private scripts and plug-ins shared within a company.

For the file structure, reference the main plug-in repository.² A repository definition file must follow this structure:

```
{
  packages = {
    plugins = (
      {
        titles = {
          en = "Some_Plugin";
        };
        url = "https://github.com/example/plugin";
        path = "Some_Plugin.glyphsPlugin";
        descriptions = {
          en = "A_description_of_the_plugin.";
        };
        screenshot = "https://example.org/image.png";
      },
      ...
    );
    scripts = ( ... );
    modules = ( ... );
  };
}
```

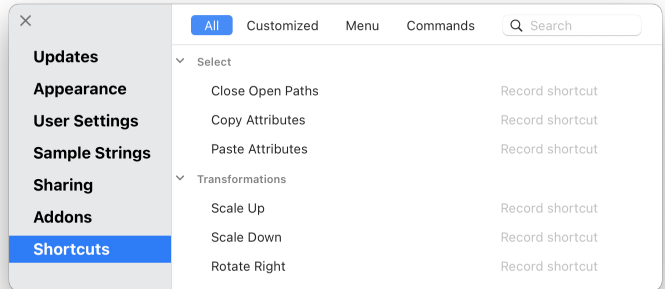
See the main plug-in repository for all available fields. Only the lists currently in use must be specified. For example, when only distributing scripts, the `plugins`, and `modules` values may be left off.

The `titles` and `descriptions` fields can offer multiple languages. See the package index website³ for a list of language codes. Glyphs requests the `screenshot` URL with an `Accept-Language` HTTP header, allowing the server to respond with a localized screenshot.

² github.com/schriftgestalt/glyphs-packages/blob/glyphs3/packages.plist

³ github.com/schriftgestalt/glyphs-packages

3.7 SHORTCUTS



The *Shortcuts* settings allow recording keyboard shortcuts for frequently used actions. Invoke a shortcut by pressing one or more modifier keys (⌘ Command, ⌥ Option, ⌘ Control, ⇧ Shift) and a regular key like A, 5, Return, or Escape.

Click *Record shortcut* and press a shortcut on the keyboard to add the shortcut to the selected action. Keep the current shortcut by clicking the curved arrow ↶ icon, or click the cross ✕ icon to remove the current shortcut from an action:



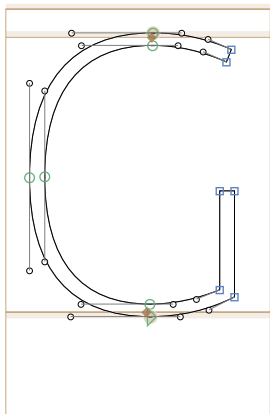
Shortcuts are shown using key symbols:



Use the buttons at the top of the *Shortcuts* settings window to filter the actions list. *All* shows all actions, *Customized* lists only the actions with a changed shortcut, *Menu* shows all actions accessible from the menu bar at the top of the screen, and *Commands* shows actions within the current selection.


Actions are grouped into sections. Click the disclosure chevron ▼ next to the title of a section to collapse and expand the section. Filter the list of actions further by searching for actions with the search field in the top-right.

4 Edit View




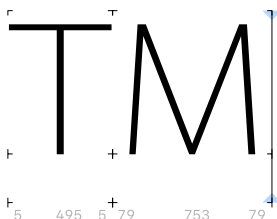
Edit glyph outlines, spacing, kerning, mark-positioning, hinting, and more in Edit View. Edit a glyph by double-clicking it. Open multiple glyphs in Edit View by selecting them and choosing *View > New Tab* (Cmd-T). In Font View, a selection of glyphs can also be edited by double-clicking the selection or pressing Cmd-Down Arrow.

A Glyphs window can contain multiple Edit View tabs. Switch to a tab by clicking its tab title. Press Cmd-Opt-2 through Cmd-Opt-9 to switch to the respective tab. Cmd-Opt-1 switches to Font View (see p. 66), which is always the first tab in the window.

Switch to the next or previous tab with *View > Navigation > Show Next Tab* (Ctrl-Tab) or *Show Previous Tab* (Ctrl-Shift-Tab). Close a tab by moving the mouse cursor over its title and clicking the close  button or choosing *View > Close Tab* (Cmd-Shift-W). Close all tabs except for the current one by holding down the Option key when clicking the close button. Restore an accidentally closed tab by holding down Option and choosing *View > Reopen Last Closed Tab* (Cmd-Opt-Shift-W).

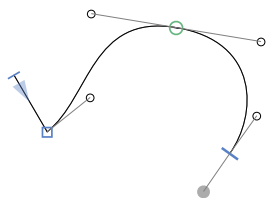
Edit View has two modes: *text mode* and *edit mode*.


- ▶ Use **text mode** to insert and remove glyphs from Edit View, or adjust the spacing and kerning between glyphs. Choose the Text tool  (shortcut T) to enter text mode. See section 4.9, ‘Entering Text’ (p. 45) and chapter 9, ‘Spacing & Kerning’ (p. 123) for more details.
- ▶ Enter **edit mode** by choosing a different tool or by double-clicking a glyph. Or, press the Escape key to edit the glyph located after the text cursor.



4.1 DRAWING PATHS

4.1.1 Draw Tool



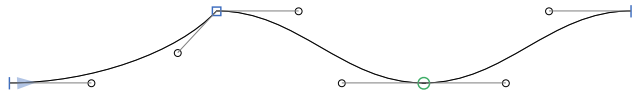
Draw outlines with the Draw tool  (shortcut P as in ‘Pen’ or ‘Path’). Click anywhere on the canvas to place a node. Placing multiple nodes connects them to a path.

Click and drag to create a curved path segment. Dragging extends two handles from the placed node. The length of the handles controls the curvature of the segment. Hold down the Option key to change the handle of the next segment only, and

This handbook uses the term *node* to refer to on-curve points, *handle* for off-curve points, and *point* as an umbrella word for both types.

keep the handle of the previous segment as it is. Hold down the Command key while dragging to change the length of only the next handle while keeping both handles at the same angle. When dragging, hold down the Space bar to reposition the node. Click on the first node of a path to close the path.

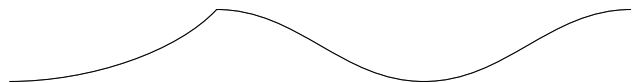
A node can either be a *smooth node* (●) or a *corner node* (■). Smooth nodes always keep both of their handles in a straight line. They appear round and green. Corner nodes appear square and blue. Their handles may form a straight line, but they can also form any non-smooth corner. The size and colors of nodes are configurable in the settings (see p. 14).



Handles (also called *Bézier control points* or *off-curve points*) control the curvature of their path segment. They are displayed as small circles and connected to their node with a thin gray line.

In open paths, start and end points are displayed as short perpendicular blue lines. The start point also features a blue triangle, indicating the path direction. Points are stored in path direction order, which is significant for some path operations. See also section 4.2.13, ‘Controlling Path Direction’ (p. 31).

Toggle the display of nodes with *View > Show Nodes > In Foreground* (Cmd-Shift-N).

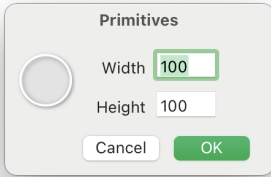




4.1.2 Pencil Tool

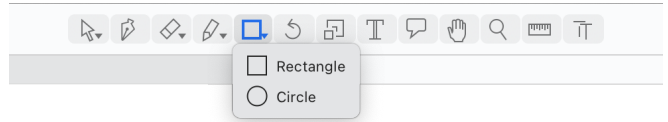
The Pencil tool (✎) (shortcut B) offers a quick way to draw freehand curves, especially when using a drawing tablet. This includes drawing with Apple Pencil on an iPad using Sidecar. The resulting paths will need some cleaning up (see section 4.2, ‘Editing Paths’, p. 23) because Pencil paths usually contain too many nodes in an attempt to reproduce the pencil drawing faithfully.

4.1.3 Primitives

Glyphs offers rectangles and ellipses as built-in primitive shapes.




Click the Primitives tool /, or press F to activate it. Click and hold the Primitives tool icon or press Shift-F to switch between the two shape options. Alternatively, use *Draw Circle* or *Draw Rectangle* from the context menu to switch between the two modes.

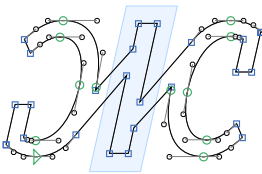
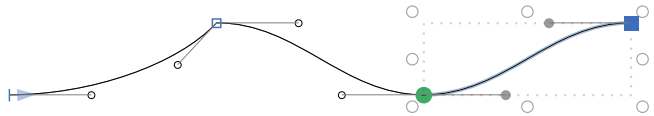


Click once on the canvas to create a primitive by entering its measurements with the keyboard. Click and drag to draw it directly into the edit area. Hold down Shift for a perfect square or circle. Hold down Option to draw the shape from its center.

4.2 EDITING PATHS

4.2.1 Selecting Nodes and Paths

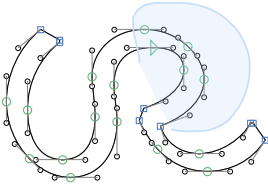
Click and drag with the Select tool  (shortcut V) to select nodes and handles inside a rectangular selection area. Hold down the Option key to ignore the handles and only select on-curve nodes.






While dragging a selection, hold down the Control key to change the selection angle. Release Control to resize the selection at its current angle. Slanted selections are particularly helpful for italic designs.

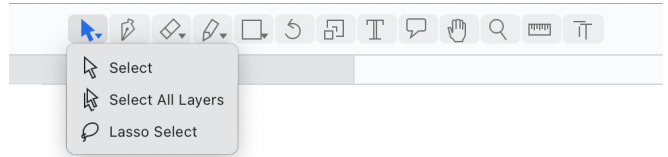
Click a point or a segment to select it. Hold down the Shift key to extend or reduce the selection. Double-click near an outline segment to select the complete path. Choose *Edit > Invert Selection* (Cmd-Opt-Shift-I) to select all unselected points, or *Edit > Deselect All* (Cmd-Opt-A) to cancel the selection. Handles can be selected independently of the nodes. Shift-select handles to create a non-contiguous selection. When a single node or handle is selected, press the Tab key to select the following point on the path, or Shift-Tab to select the previous point.

4.2.2 Free-form Selections



Use the Lasso tool  (shortcut V or Shift-V) to draw a non-rectangular selection. This is particularly useful for glyphs with many points, where a rectangular selection would not be precise enough.

Activate the Lasso tool when it is not displayed in the toolbar by clicking and holding the Select tool  and choosing *Lasso Select* from the menu. Alternatively, press Shift-V until the Lasso tool  icon is shown.



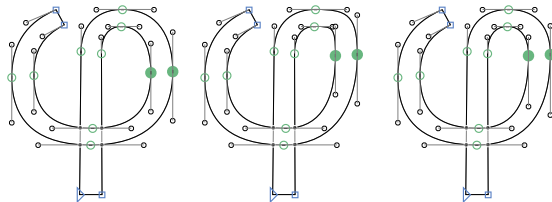
4.2.3 Moving Selected Nodes and Paths

Move the selection using the mouse or the arrow keys. Moving nodes will move the attached handles, even if they are not selected. Hold down Shift for increments of 10, and Command for increments of 100. Hold down Option to move only the selected points and not the attached handles. While moving one or more nodes, hold down both Control and Option (or add Option after starting to drag) to ‘nudge’ them, that is, to proportionally adjust the surrounding unselected handles at the same time.


Left: Original glyph outline with two selected nodes.

Center: Move selected nodes, handles stay the same.

Right: Nudge selected nodes, handles are adjusted.



Move a handle by dragging it with the mouse or pressing the arrow keys. If multiple handles are selected, they all move simultaneously. Moving one or more handles while holding down the Option key preserves their angles. When using the keyboard, add the Shift key for increments of 10 or the Command key for increments of 100.

While moving a handle of a smooth  node, hold down Control and Option simultaneously to reproduce the length and

angle of that handle to its matching one on the other side of the node.

Drag a segment to move both connected nodes and their handles. Option-drag a segment to change the length of the handles but keep their respective angles.

4.2.4 Converting Nodes and Segments

Convert between smooth ● connections and corners ■ by double-clicking a node or selecting one or more nodes and pressing Return.

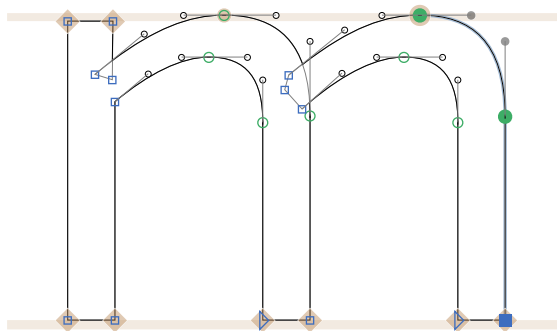
When converting from corner to smooth, ○ handles are added on either side of the node, possibly distorting the outline to keep the handles level. Hold down the Option key while double-clicking or pressing Return to maintain the current outline.

The *Path > Tidy up Paths* command (Cmd-Shift-T) applies heuristics to set the appropriate mode for all nodes at once or a selection of nodes. It also removes superfluous points, for example, handles on a straight segment or an on-curve point exactly on the line between two others. Hold down Option to choose *Path > Tidy up Paths for all Masters* (Cmd-Opt-Shift-T), applying the command on all masters of the selected glyphs.

Option-clicking a line segment converts it into a curve segment, and handles are added to the two bounding nodes. Convert a curve back into a line segment by selecting and deleting one or both of its handles.

Be careful when tidying up paths: In Multiple Master setups, superfluous points may be necessary for outline compatibility.

4.2.5 Nodes in Alignment Zones



Nodes located exactly on a vertical metric line (see p. 95) are highlighted with a beige diamond ◆. Inside an alignment zone,




the highlighting assumes the shape of a circle ●. This helps to control the position of nodes even at small zoom scales.



4.2.6 Scaling & Rotating



The attributes of the current node selection are shown in the Info box (*View > Show Info*, Cmd-Shift-I):

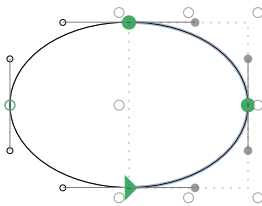


Tip: In all number input fields in Glyphs, use the up and down arrow keys to increase or decrease the value.

When multiple nodes are selected, scale and move the selection by changing the coordinates (X and Y) and the dimensions (↔ width and ⇅ height) in the Info box. Set the transformation origin with the reference points  on the left. Close the lock  symbol to scale width and height proportionally. Open the lock  to scale width and height independently of each other. Use the Up and Down arrow keys to step through the numbers. Hold down Shift for increments of 10.

When multiple points are selected, the solid square  indicates the number of selected points. The number next to the outlined square  represents the total number of points on the current glyph layer.

Rotate and scale the selection manually with the Rotate tool  (shortcut R) and the Scale tool  (shortcut S). With one of these tools active, click anywhere on the canvas to set the transformation origin, and then click and drag to transform the current selection. Hold down the Shift key to rotate in steps of 90° or to scale proportionally.



When multiple nodes are selected, a bounding box is displayed with transformation knobs on all four sides and corners. Drag a knob to scale the selected points with respect to the opposite knob. Hold down Shift to scale proportionally, and hold down Option to use the center of the box as the transformation origin.

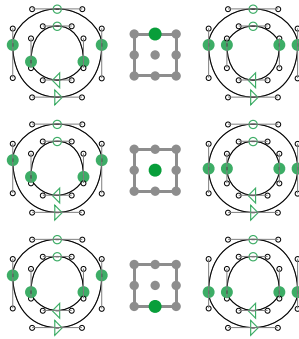
Toggle the display of the bounding box with *View > Show Bounding Box* (Cmd-Opt-Shift-B). More path transformations are possible via the Palette. See section 5.4, 'Transformations' (p. 62) for further details.

4.2.7 Aligning

Choose *Path > Align Selection* (Cmd-Shift-A) to quickly align all selected points. The command aligns nodes, handles, and anchors. Glyphs will automatically choose between horizontal

and vertical alignment, whichever is smaller for the current selection. The *Align Selection* command respects the transformation origin of the transformation palette (see also section 5.4.1, ‘Transformation Origin’, p. 63):

Align Selection for different transformation origins.



Alternatively, set the width or the height value of two or more selected points to zero in the Info box. Align an anchor horizontally above a point with the *Align Selection* command while exactly one anchor and one path point are selected, or center an anchor between two points while two points and one anchor are selected. Glyphs respects the italic angle when aligning anchors to nodes.

Using the *Align Selection* command while exactly one point and one component are selected will align the origin point of the component to the selected node. The node keeps its position. The origin point is where the baseline crosses the left sidebearing if the italic angle is zero. If the component glyph contains an **origin** anchor, its position is used as the origin point instead.

If the italic angle is not zero, instead of the left sidebearing, an imaginary vertical line crossing the slanted LSB at half x-height is used. In that case, the origin point is where this line crosses the baseline.



Applying *Path > Align Selection* on a single node will try to move the node over the nearest node in the background. Align partial paths, complete paths, or components to each other using the Transformations section in the Palette (Cmd-Opt-P). See section 5.4, ‘Transformations’ (p. 62) for more details.

4.2.8 Duplicating Paths


Quickly duplicate the current selection by holding down Option while dragging a copy of the paths into their new position. Alternatively, copy (Cmd-C) and paste (Cmd-V) the selection.

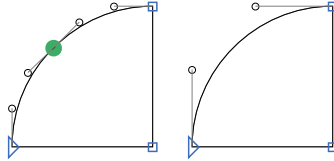
Option-dragging partial paths will duplicate the selected segments. This can be helpful when replicating glyph parts like serifs or spurs.

4.2.9 Deleting Nodes

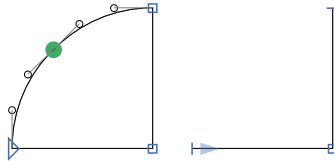
Note: If the Erase tool  icon is not visible, press Shift-E or click and hold the Knife tool (see p. 29)  icon and choose *Erase*.

Select a node and press the Delete key to delete it.

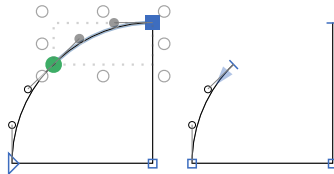
Alternatively, choose the Erase tool  (shortcut E) and click a node to delete it. Glyphs will keep the path closed and will try to reconstruct the same path segment without the node:





Press Opt-Delete to break the path by removing the node and both path segments surrounding the node:

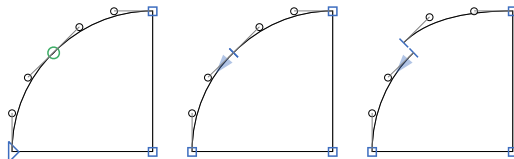








Delete a single segment between on-curve nodes with the Erase tool by Option-clicking it. Delete all selected segments by pressing Opt-Delete. Alternatively, select a handle and erase its segment by pressing Opt-Delete.




4.2.10 Opening and Closing Paths

With the Draw tool  (shortcut P), click a node to open the path at the node position. Short blue perpendicular lines mark open path endings. Drag the path ends apart using the Select tool  (shortcut V).

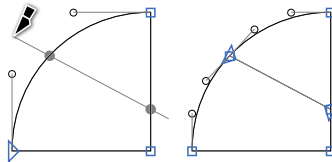



Close a path by dragging an open line ending on top of another with the Select tool. Select two path endings and choose *Connect Nodes* from the context menu to add a connecting line segment between the nodes:   → . Choose *Close Open Paths* to close the selected paths fully:   → .

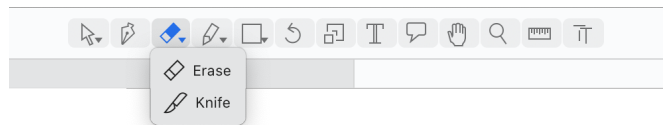
4.2.11 Cutting Paths

With the Knife tool  (shortcut E or Shift-E), click and drag a line across a path to cut the outline into two separate outlines. Glyphs will close the two resulting paths along the cutting line. Cutting across several overlapping paths will rewire the segments with each other.

Tip: When multiple tools share one icon in the toolbar, such as the Knife and Erase tools, add Shift to the tool shortcut to toggle between the tools.



Activate the Knife tool by clicking and holding the Erase tool  and choosing *Knife* from the menu. Alternatively, press Shift-E.

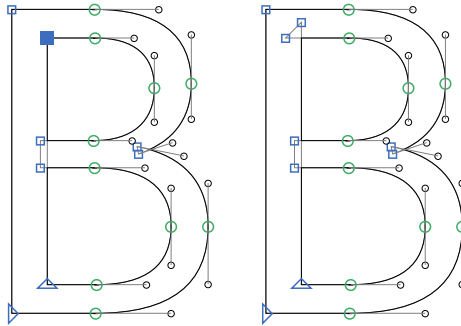


4.2.12 Re-segmenting Outlines

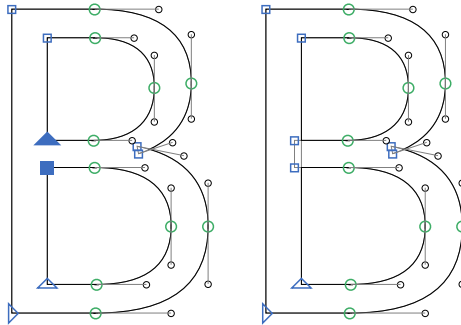
Open corners and reconnected nodes offer finer control and allow manipulating path segments independently of each other. This control also makes interpolating between masters easier.

Open a corner node into two nodes by selecting it and choosing *Open Corner* from the context menu:

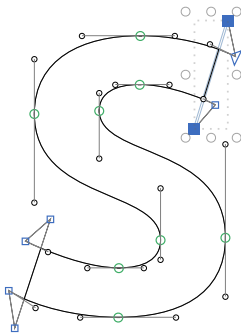
The top-left corner of the counter is opened with *Open Corner*. Opening corners only works on corner nodes.



Choose *Reconnect Nodes* from the context menu to the two nodes of an open corner to get back to the original corner node. Select an even number of nodes and apply *Reconnect Nodes* to reconnect each node with its closest neighbor:

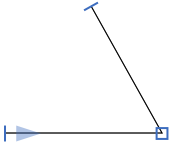


The size of the created overlap will be approximately half the first values entered for vertical and horizontal stems in *File > Font Info > Masters* (Cmd-I). Thus, the reconnected nodes should extend comfortably into the stem (when used on counterforms such as the B in the image above) or outside the outline, such as the s on the left.



Opened corners are considered invisible if the triangular overlaps are small enough in relation to the neighboring visible outline segments. That way, opened corners can also be placed on the outside of paths. If the overlap size goes beyond the threshold size, they will be visible. These outwards facing open corners are useful for editing bent terminals, as in a sans serif lowercase s. Select a path segment and open the two nodes on its extremities with the *Open Corner* command.

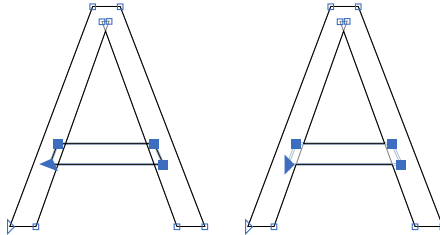
4.2.13 Controlling Path Direction



Tip: Quickly create a punched through counterform by overlapping two paths and pressing `Cmd-Shift-R` (*Path > Correct Path Direction*).

The starting point of a closed path is displayed as an arrowhead ► (or ► for a smooth node) following the path direction. The end nodes of an open path are displayed as short perpendicular blue lines, where a light blue arrowhead indicates the first node. On a closed path, make any on-curve node the first node by choosing *Make Node First* from the node context menu.

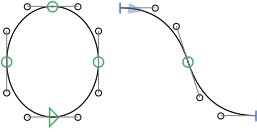
All outer paths need to run counterclockwise, while enclosed paths (such as the inner path of an O) must go clockwise. Change the path direction of a path by selecting it and choosing *Path > Reverse Contours* or *Reverse Selected Contours* from the context menu. Hold down Option to change the direction in all masters. When no path is selected, use *Path > Reverse Contours* from the menu bar or *Reverse All Contours* from the context menu to toggle all path directions in a glyph.



Path > Correct Path Direction (`Cmd-Shift-R`) will perform an informed guess and find the right path directions for all paths on the current glyph layer or all selected glyph layers. This will also rearrange the shape order and reset the starting points of all paths to their bottom-left node. Holding down the Option key changes the command to *Correct Path Direction for all Masters* (`Cmd-Opt-Shift-R`). As the name indicates, it will include all master layers, all Alternate (see p. 166), and Intermediate layers (see p. 165). The command ignores all other non-master layers. This is useful in a Multiple Master setup.


For successful interpolations, the path order, starting points, and path directions must be compatible and consistent throughout all font masters. See section 11.5.2, ‘Correcting Path Direction’ (p. 162) for details on fixing the path direction for interpolation.

4.2.14 Extremes & Inflections



Extrema are the positions on a path with a completely horizontal or vertical tangent. Inflections are positions in path segments where the segment changes its bend from clockwise to counterclockwise or vice versa.

It is considered good practice to have nodes on extremum points. Some font technologies, like hinting, require nodes at extremum positions. Some operations, like offsetting a curve (see p. 181), work better with inflection points placed on the undulating curves. Also, some font renderers may behave unexpectedly if such nodes are not in place. Furthermore, inflection points pose a problem for outline interpolation since they can easily cause kinks in outlines.

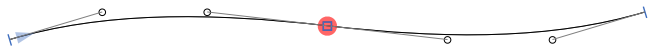
Insert nodes on extremum and inflection points by Shift-clicking a segment with the Draw tool  (P). A node will be inserted at the nearest extremum or inflection.

Alternatively, choose *Path > Add Extremes*, and nodes will be added at extremes on all paths of the active layer. Glyphs will not add an extreme if the resulting segment were very short. In this case, it assumes that the node placement was intentional. Force Glyphs to add all extremes by opening the *Path* menu, holding down Option, and choosing *Force Extremes*. When a node is only slightly off the extremum position, Glyphs will attempt to preserve the outline shape while moving the node into the extremum position and turning the surrounding handles entirely vertical or horizontal.

Extremes are added automatically at export time with a *Filter* custom parameter called *AddExtremes*. See section 12.2.9, ‘Add Extremes’ (p. 187) for details. This can be useful for shallow curves or certain Multiple Master situations, where adding extremes would make editing or interpolating unnecessarily complex.

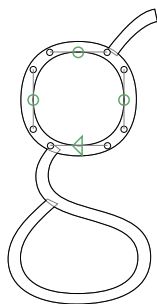
4.2.15 Duplicate Nodes

When two adjacent on-curve nodes share the same coordinates, they are highlighted with a red circle. Merge these nodes with *Path > Tidy up Paths* (Cmd-Shift-T).




4.2.16 Focusing & Locking

Prevent accidental edits with focusing and locking.



Focus on one or multiple paths by selecting at least one point for each path and choosing *Focus on Selected Path* from the context menu. Focusing on a path hides the controls for all other paths. Use path focusing to exclusively work on a single path, even in complex glyphs. Filters, plug-ins, and other scripts may still operate on all paths; only the Edit View controls are hidden. Release the focus by choosing *Clear Focus on Paths* from the context menu.

The reverse is possible by locking paths: Lock a path by Control-clicking or right-clicking one of its nodes and choosing *Lock Path* from the context menu. Points of a locked path cannot be selected or modified in Edit View. Like with focusing, locked paths can still be changed by filters, plug-ins, and scripts. Points of a locked path turn red when the mouse cursor is placed on them, indicating that they cannot be dragged. Unlock a path by choosing *Unlock Path* from the context menu on any of its nodes.

Lock and unlock a glyph by choosing *Locked* from its context menu. A lock  icon appears in the top-right corner of the glyph in Edit View and Font View. A locked glyph cannot be modified: not with the Select tool, not by a filter, plug-in, or script. This edit-protection is helpful if the glyph is considered final, and no further edits should be applied.

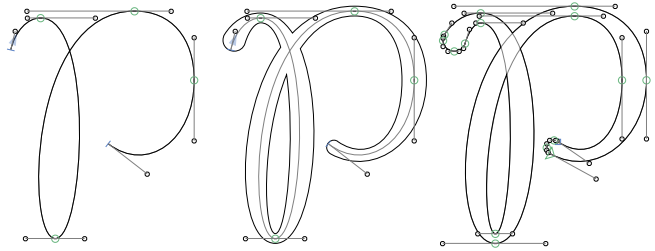
4.3 GRAPHIC ATTRIBUTES

Paths can be styled with graphic attributes. These attributes include stroke styles, fill styles, and masking. The attributes are applied to a path in the Palette (*Window > Palette*, Cmd-Opt-P) and are accessible when *View > Show Info* (Cmd-Shift-I) is active and a path is selected.

Copy the attributes of one path to another with the context menu: Choose *Copy Attributes* on a path and apply them to another by choosing *Paste Attributes*.

4.3.1 Creating Strokes

Left: an open path
Center: the path with a stroke
Right: the expanded outline



Expand a path to a shape by selecting one or more of its points and entering a stroke width and height in the *Stroke* section in the Palette. The stroke width (W) and stroke height (H) are measured in font units. If no height is set, then the width is used for both dimensions. Below the width and height fields are controls for the stroke placement (☐ along the path, ☐ to the left of the path, ☐ to the right of the path) and controls for the stroke ends (☐, ☐, ☐, ☐, ☐).

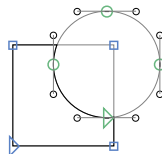
A stroke can be applied to both closed and open paths. For closed paths, select the *Fill* checkbox from the Palette to fill the path delineated by the new outlines.

The original path is displayed in a light gray color, while the stroke outline is black. A stroke path can be converted to a conventional outline by choosing *Expand Outline* from the context menu. Hold down Option to expand outlines on all masters.

4.3.2 Masking

Select a path and check *Mask* from the Palette to subtract the path from all shapes below it. Applying a mask is a non-destructive action; the path can still be moved and modified.

Subtracting a circle from a square by applying a mask to the circle path.



The subtracted parts of paths are drawn in a light gray color, while the exporting outline is drawn in black.

Masking is based on the order of shapes, since a masking path only affects shapes below it. Change the shape order by choosing

Filter > Shape Order. Drag the shapes into the desired order and confirm with *OK*.

4.4 ANCHORS

Anchors are special points that fulfill multiple tasks in Glyphs. They primarily serve as a connecting pivot for automatically aligning components, corners, caps, mark-to-base and mark-to-mark positioning, and cursive attachment. These anchors adhere to certain naming conventions. For more details on these uses, see section 10.1.7, ‘Anchors’ (p. 136). When an **origin** anchor is placed inside a glyph, it can align a component to a regular path node (see also section 4.2.7, ‘Aligning’, p. 26).

Some third-party scripts and plug-ins make use of special anchors. Refer to their documentation for further details.

4.4.1 Adding, Editing, & Removing Anchors

Insert an anchor by Control-clicking or right-clicking the Edit View canvas and choose *Add Anchor* from the context menu. An anchor named ‘new anchor’ will be placed at the click position. Its name is already selected for renaming. After typing the new name, confirm it by pressing the Return key or clicking anywhere in the canvas.




	new anchor
X	377
Y	610

Double-click an anchor to rename it, or select it and press Return, or select it and edit the anchor name in the Info box.

The glyph info database has default anchors associated with many glyphs. Add these pre-defined anchors to the selected glyphs by choosing *Glyph > Set Anchors* (Cmd-U) or, with the Option key held down, *Glyph > Set Anchors for all Masters* (Cmd-Opt-U). Delete all existing anchors and start over with *Glyph > Reset Anchors* (Cmd-Shift-U); hold down Option to reset the anchors across all masters.

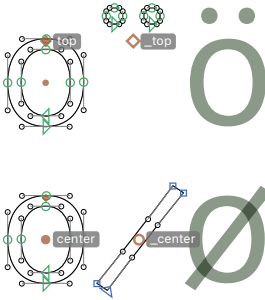
Select an anchor by clicking the orange dot ● that represents it. Select the next or previous anchor by pressing the Tab key, or Shift-Tab, respectively. Select multiple anchors by Shift-clicking them. Names are only shown for selected anchors. Select all anchors by running *Edit > Select All* (Cmd-A). This command may need to be issued twice, since *Select All* will select all paths and only select all anchors and components if all available paths have already been selected.




Move an anchor like a node: either with the Select tool and mouse or arrow keys or through the coordinates in the Info box.

An anchor is diamond-shaped  if placed exactly on a metric line such as the x-height or the baseline, square  in an alignment zone, and circle-shaped  in all other cases. Quickly duplicate an anchor by Option-dragging it. Since anchor names must be unique inside a layer, an underscore will be added to the end of the name. Remove all selected anchors by pressing the Delete key.

Align an anchor to one or two nodes by selecting the anchor and the nodes and choosing *Path > Align Selection* (Cmd-Shift-A). See section 4.2.7, ‘Aligning’ (p. 26) for details.

4.4.2 Mark to Base Positioning



Glyphs can automatically build the ‘mark’ (Mark to Base) feature using anchors. The combining diacritical marks must contain underscore-anchors (for example, `_top` or `_bottom`), and the base glyph must contain matching anchors without the underscore prefix (for example, `top` or `bottom`). Anchors with an initial underscore are displayed with a hole: , , . Combining glyph names often carry the word ‘comb’ at the end of their names, such as `acutecomb` (◌̇) or `macroncomb` (◌̄).

Combining diacritical marks have Unicode values and thus can be typed or inserted in a text. This way, a font user can place any mark on any base letter by first typing the regular letter and then inserting the combining mark.

Alongside the ‘mark’ feature, Glyphs will also build the ‘ccmp’ (Glyph Composition and Decomposition) feature if glyphs like `idotless` and `jdotless` are present. See section 7.4.5, ‘Implicit Features’ (p. 105) for further details.

4.4.3 Mark to Mark Positioning

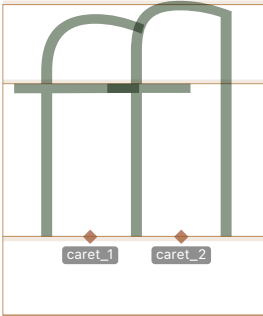
Glyphs will automatically build the ‘mkmk’ (Mark to Mark) feature if both underscore- and regular anchors are present in a combining diacritical mark. A font user will then be able to stack any combining mark on any other combining mark carrying both anchors.

4.4.4 Cursive Attachment

Enable proper cursive attachment in Arabic typesetting by adding `exit` and `entry` anchors to the respective stroke endings and beginnings in medial, final, and initial letterforms. The `entry` anchor of the instroke will be connected to the `exit` anchor of

the preceding outstroke. Preview cursive attachment immediately in Edit View when right to left typesetting is enabled (see p. 46).

4.4.5 Ligature Carets



Ligature carets define the positions where a text cursor should be placed on a ligature glyph. In a ligature glyph, these positions are defined by special anchors on the baseline. They must be named ‘caret’, followed by an underscore suffix, for example, `caret_1`, `caret_2`, ... The suffix needs to be different for each anchor because anchor names must be unique inside a glyph layer. The numbering order is not significant and is exclusively used for differentiating anchors.

Glyph > Set Anchors (Cmd-U) will insert appropriate caret anchors in properly named ligature glyphs. Most ligature glyphs are named with their individual glyph names joined by underscores, such as `s_t` or `f_f_l`. For the glyph naming convention employed by Glyphs, see section 6.4, ‘Names & Unicode’ (p. 77).

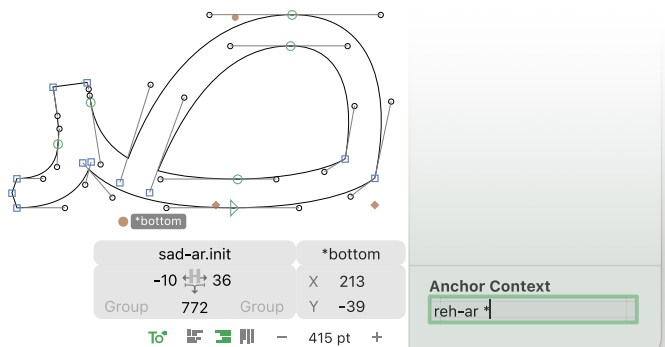
At export, Glyphs will use the caret information to build so-called `LigatureCaretByPos` instructions in the GDEF OpenType table. At the time of this writing, the only known software supporting ligature caret positioning are Mac applications that make use of the Cocoa text engine. Adobe and Microsoft apps ignore this information.

4.4.6 Contextual Mark Attachment

Make an anchor contextual by prefixing its name with an asterisk (*). Select the anchor and edit the context in which it should overwrite the unprefix counterpart in the *Anchor Context* field:

Moving the `bottom` anchor to the left in case the current glyph (`sad-ar.init`) is preceded by `reh-ar`.

The original `bottom` anchor (next to `▷`) is used for mark attachment unless the glyph is in a context that matches the *Anchor Context* of `*bottom`.



The context field is located at the bottom of the Palette (*Window > Palette*, Cmd-Opt-P) and is shown when *View > Show Info* (Cmd-Shift-I) is checked. Write OpenType feature code into the *Anchor Context* field with *** representing the current glyph (the base glyph). Classes and tokens may also be used in the context.

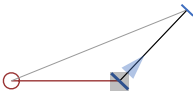
Additionally, the *&* symbol can be written in the context code as a placeholder for the mark glyph. This is optional, but needed when the mark does not directly follow the base glyph.

Write `lookupflag ...;` before the context pattern code to specify lookup flags. This is often useful with the `UseMarkFilteringSet` flag to limit the anchor context to a subset of marks.

Differentiate multiple context anchors by appending an arbitrary suffix to their names. For example, the anchor `bottom` might have two context anchors: `*bottom.noon` and `*bottom.reh`, which have different anchor contexts.

4.5 GUIDES


4.5.1 Magnetic Guides




When dragging a node selection across the canvas, red lines will appear, indicating when the selection is aligned with other nodes or a vertical metric. Deactivate magnetic guides temporarily by holding down the Control key.


Likewise, a node or any other object being dragged will snap to all nodes and handles on paths, as well as in components. When moving close to a node in a component while dragging, Glyphs will fade in small representations of the nodes inside the component. Disable node snapping by holding down the Control key.


4.5.2 Local & Global Guides



Add a local guide to the currently displayed glyph layer by Control-clicking or right-clicking to open the context menu and choosing *Add Guide*. A local horizontal guide  will be added at the click position. If two nodes are selected while adding the guide, it will be laid through the nodes. Toggle the display of guides in Edit View by choosing *View > Show Guides* (Cmd-Shift-L).

Select a guide by clicking anywhere on it. A filled knob indicates a selected guide. Move a selected guide by dragging its

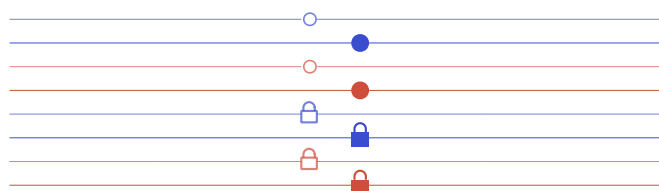
Pro Tip: Quickly create a guide using the Measurement tool  by simultaneously holding down Cmd-Ctrl-Opt and pressing the G key after starting to drag a measurement line.

knob. Double-click the knob to turn it perpendicular  to its current orientation. Quickly duplicate one or more guides by selecting them and holding down the Option key while dragging them to a new position.

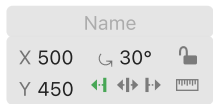
Local guides are blue and visible only on the layer on which they have been placed. Global guides  are red and visible throughout a master. Create a global guide by holding down the Option key while navigating the context menu and choosing *Add Global Guide*. Toggle a guide between local and global by selecting it, and from the context menu, choose *Make Global Guide* or *Make Local Guide*.



Lock one or more guides by choosing *Lock Guides* from their context menu. A locked guide cannot be selected and displays a lock / icon instead of its knob. Unlock a guide by Control-clicking or right-clicking the knob and choosing *Unlock Guide* from the context menu.




From top to bottom
(unselected and selected):
local guide, global guide,
locked local guide, locked
global guide.




Press the Tab key to quickly select the next, or Shift-Tab to select the previous guide. When a guide is selected, move it using the arrow keys (add Shift or Command for larger increments) or drag its knob with the mouse, just like a regular node.



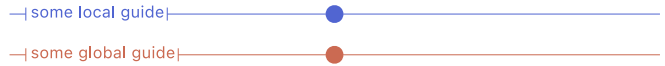
Change the angle by dragging the guide anywhere outside the knob. Enter values for its position and its angle in the Info box (Cmd-Shift-I). Click the lock  icon next to the angle in the Info box to maintain the guide at the set angle until the angle is unlocked  again.

By default, a guide will be positioned relative to the left sidebearing. Click  in the Info box to position it relative to the right sidebearing or  to position it relative to both sidebearings. Click  to use the left sidebearing again. This can be useful for slanted guides, especially when they are global or frequently changing the right side.

Select a guide and click the measurement  icon in the Info box to turn the guide into a measurement guide. For more details, see section 4.10.3, 'Measurement Guides' (p. 50).

Tip: Use *View > Show Metrics Names* to label the vertical metrics. No guides are needed.

Click *Name* in the Info box of a selected guide to name it. Guide names are displayed on the left end of a guide:



Global guides are shown on all glyphs by default. Select a global guide and choose *Edit > Info for Selection* (Cmd-Opt-I) to limit the global guide to a subset of glyphs. The rules for defining the scope of a global guide work the same as Smart Filters. See section 6.6.4, ‘Smart Filters’ (p. 86) for details.

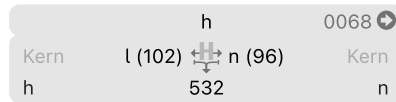
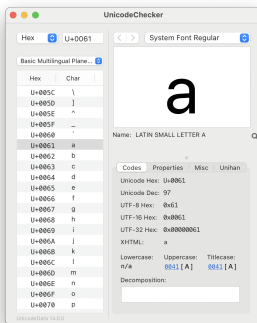
4.5.3 Glyph-Specific Undo History

In Edit and Font View, the Undo mechanism works on a glyph level. That means that every glyph has its own undo history.

This also implies that certain global actions, especially manipulating global guides, cannot be undone. That is because global guides are associated with a master and not a glyph, and therefore are ignored by the glyph-level undo history.

4.6 INFO BOX

An Info box is displayed below the current glyph. The current glyph is the glyph currently being edited on, or, in text mode, the glyph following the text cursor. The Info box shows general information about the glyph, including as its name, Unicode value, metrics, and kerning.




The glyph name is shown centered at the top of the Info box. Its Unicode value, if any, is shown in the top-right of the box. Click the arrow button next to the Unicode value to open it in UnicodeChecker. If UnicodeChecker is not installed, Glyphs will present a download link. When opening UnicodeChecker with the arrow button for the first time, the Mac will ask for permission. Grant it so that Glyphs can launch UnicodeChecker and set the current Unicode value to match the one from Edit View.

*UnicodeChecker*¹ is a Mac app by earthlingssoft. Use it to explore and convert Unicode values.

1 earthlingssoft.net/UnicodeChecker

4.6.1 Horizontal Layout

The lower half of the glyph Info box manages metrics and kerning. To the left and right of the horizontal metrics  icon are left and right sidebearings. Below the icon is the width of the glyph. LSB, RSB, and width values can be numbers or metrics keys. See section 9.1.3, ‘Metrics Keys’ (p. 124) for more details.

Kerning values and kerning groups are at the left and right edge of the Info box. See section 9.2, ‘Kerning’ (p. 126) for more information.

4.6.2 Vertical Layout


For glyphs written in a vertical layout, the vertical Info box is displayed instead. (See section 4.9.2, ‘Writing Direction’, p. 46 for more information on vertical layout.)



T and B are the top and bottom sidebearings of the glyph, respectively. O is the vertical origin of the glyph. H is the vertical width (or height) of the glyph. K and G define the kerning and kerning groups, both for the top and the bottom of the glyph.

4.7 GLYPH DISPLAY

4.7.1 Zooming

There are many ways to zoom in and out of Edit View. With a trackpad, zoom using **pinch and stretch gestures**. Or hold down the Option key and use a scroll gesture or the scroll wheel of a mouse. Or activate the **Zoom tool**  (shortcut Z) and click on the canvas to zoom in, Option-click to zoom out. Alternatively, click and drag across an area, and it will be zoomed to fill the window. Temporarily activate the Zoom tool by holding down Cmd-Space for zooming in or Cmd-Opt-Space for zooming out. If Cmd-Space collides with another shortcut, try pressing Space before adding the Command key.

Or use the **zoom commands** from the *View* menu: *Zoom In* (Cmd-Plus) and *Zoom Out* (Cmd-Minus). *Zoom to Active Layer* (Cmd-Zero) will maximize the area between ascender and descender in the window. *Zoom to Actual Size* (Cmd-Opt-Zero) will zoom one font unit to the size of one screen point. (One

Tip: The Spotlight shortcut in the System Settings may need to be changed for the Cmd-Space shortcut to work in Glyphs.

screen point is one pixel on a classic low-resolution screen and two pixels on new high-resolution Retina screens.)




Or use the **zoom — / + buttons** in the bottom-right corner of the window. Alternatively, set the zoom value numerically by entering a point height in the field between the buttons. The zoom value is the number of display points at which 1000 font units are shown. The size of a display point depends on the physical display that the Mac uses. Display points are independent of pixels, so one point might correspond to one pixel on a low-resolution display or two pixels on a high-resolution display. For example, at a zoom value of 300, a 1000 font units long path would be displayed at 300 display points, and a path of length 500 font units would be displayed at 150 display points.

4.7.2 Panning

On a trackpad, drag two fingers to pan around Edit View. Scroll on a mouse to pan vertically, hold down Shift to scroll horizontally. Or, drag the scroll bars on the right and bottom edges of the displayed canvas.

Tip: The Spotlight shortcut in the System Settings may need to be changed for the Cmd-Space shortcut to work in Glyphs.

Alternatively, switch to the Hand tool  (shortcut H) and drag the canvas around, or simply hold down the Space bar to temporarily switch to the Hand tool. When in text mode, pressing the Space bar would add a space to the text. Press Cmd-Space and subsequently release the Command key to avoid inserting a space.

4.7.3 View Options

Toggle settings that influence the glyph display in Edit View from the *View* menu:

Show Nodes displays the on-curve and off-curve points of a glyph. Control the display of points on the foreground layer, the background layer, and Extra Nodes (see p. 150).

Show Metrics shows the vertical and horizontal metrics as well as the alignment zones of the glyph.

Show Metric Names adds labels to the metrics in Edit View.

Show Hints displays PostScript hints. Use the TrueType Instructor

tool for TrueType hints.

Show Anchors shows the anchors of the current glyph. See section 4.4, ‘Anchors’ (p. 35).

Show Info displays the Info box as well as contextual controls at the bottom of the Palette. See section 4.6, ‘Info Box’ (p. 40).

Show Background displays the paths and components of the background layer as outlines in the foreground layer. See section 4.8, ‘Background’ (p. 44).

Show Image shows images placed on a glyph layer. See section 4.12, ‘Images’ (p. 54).

Show Guides displays local and global guides. See section 4.5, ‘Guides’ (p. 38).

Show Measurement Line shows horizontal and vertical measurement lines with numbers for the amount of space between the glyphs. Only visible when the Text tool is active. See section 4.10.4, ‘Measurement Line’ (p. 51).

Show Annotations shows annotations placed with the Annotation tool. See section 4.11, ‘Annotating’ (p. 52).

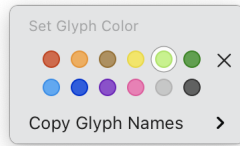
Show Bounding Box displays the bounding box of a selection with transformation knobs on all four sides and corners. See section 4.2.6, ‘Scaling & Rotating’ (p. 26).

Fill Preview fills closed paths with the foreground color. The current glyph is not filled unless the Text, Hand, or Zoom tool is selected. The foreground color can be changed in the settings. See section 3.2, ‘Appearance’ (p. 14) for details.

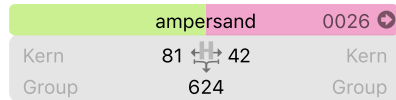
Many third-party reporter plug-ins are available for changing or enhancing the glyph display in Edit View. After they are installed, they will also show up in the *View* menu. See section 17.3, ‘Plug-ins’ (p. 237) for more details.

4.7.4 Glyph & Layer Colors

Label colors can be set both glyph-wide and layer-specific from the context menu. Control-click or right-click anywhere on the canvas of an active glyph and pick the color from the context menu.



Hold down Option and choose *Set Layer Color* to pick a layer color rather than a glyph color. Both glyph and layer colors will be displayed in the Info box (*View > Show Info*, Cmd-Shift-I):



The glyph color is displayed on the left half, the layer color on the right half, which corresponds to the display of label colors in Font View. See section 6.3.5, ‘Color Label’ (p. 72) for managing color labels in Font View.

4.8 BACKGROUND

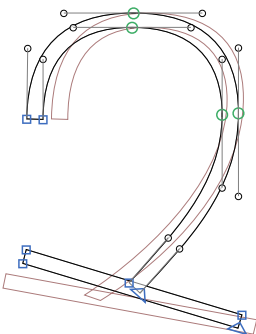
Each layer has a background layer, usually simply referred to as ‘background’. The background is useful for temporarily storing a path or tracking changes and comparing outlines before and after a manipulation. Some filters, such as *Filter > Hatch Outline*, use the background as a backup layer to work non-destructively.

While working in the foreground, objects on the background are displayed as a subtle red outline with *View > Show Background* (Cmd-Shift-B). If this option is active while in the background, the foreground objects will be displayed in the same way. Choose *View > Show Nodes > In Background* to show the on- and off-curve points of the background. When the background is displayed, snapping will also work with objects on the background layer.

Switch to the background by choosing *Path > Edit Background* (Cmd-B). The window display will darken slightly to indicate that the background layer is active. This command is a toggle; to switch back to the foreground, use Cmd-B again.

Path > Selection to Background (Cmd-J) replaces the current content of the background with the active selection; this works in reverse when the background is active. Simultaneously holding down the Option key changes the command to *Add Selection to Background* (Cmd-Opt-J) and adds the current selection to what is already in the background. *Path > Swap with Background*

Tip: Change the color of background outlines in the settings. See section 3.2, ‘Appearance’ (p. 14).



(Cmd-Ctrl-J) will exchange the foreground with the background. Empty the background layers of selected glyphs by holding down the Option key and choosing *Path > Clear Background*.

Copy the outlines of another font file using *Path > Assign Background* into the background layer of all selected glyphs. Put the same font into its own background to keep track of any other changes. Selecting all glyphs and choosing *Path > Selection to Background* (Cmd-J) has the same effect.

4.9 ENTERING TEXT

Edit View can display multiple glyphs to provide a context of words and sentences.

When using the Text tool (T), enter characters using the keyboard. Insert non-alphabetic characters with the Character Picker (*Edit > Emoji & Symbols*, Cmd-Ctrl-Space). Edit View has a preset line width. Set the maximum line width in the settings. See section 3.2, ‘Appearance’ (p. 14).

Tip: Quickly switch between text entry and editing the current glyph by pressing the Escape key.

4.9.1 Text Tool

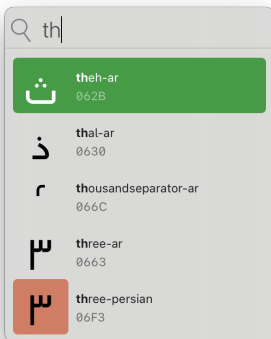
Select the Text tool **T** (shortcut T) to switch to text mode and start typing. Enter a single character, a word, a sentence, or multiple lines of text. Copy and paste text to and from Edit View. Use all familiar text editing controls such as the arrow keys, the *Edit* menu, and macOS Application Services (*Glyphs > Services*). In text mode, the current glyph is the one after the text cursor.

Find Glyphs to Insert

Insert a glyph by name with *Edit > Find > Find...* (Cmd-F). Results are shown while typing. If the search term contains spaces, glyphs matching all the space-separated terms will be shown. For example, ‘ha cy’ will find all glyphs that have ‘ha’ and ‘cy’ in their name, such as Ha-cy, Sha-cy, and sha-cy.loclBGR. Use a Unicode character as a search term to insert its glyph (or type the character directly into Edit View).

Press Return to insert the selected glyphs into Edit View. By default, the first result is selected. Shift-click to select a range of results, or Command-click to select results individually.

Click the magnifying glass **Q** to configure the search. Choose *Name* to search by glyph name, *Unicode* to search by Unicode value, and *All* to search by both. The Unicode search matches the



entered search query with the hexadecimal Unicode values of the glyphs (for example, ‘228E’).

Change the Current Glyph

Tip: On keyboards without Home and End keys, press Fn-Left Arrow and Fn-Right Arrow.

Switch to the previous or next glyph in the font by pressing the Home and End key, respectively. Add Shift to advance through the glyphs as they are currently visible in the Font View. These shortcuts are useful when filtering glyphs in the Font View and stepping through them in Edit View.

Placeholder Glyphs

Edit > Add Placeholder (Cmd-Opt-Shift-P) inserts a placeholder for the current glyph. Placeholders are dynamically replaced by the currently selected glyph. Multiple placeholders can be placed in Edit View to all reflect the same glyph. Edit a glyph for all placeholders to mirror it. Placeholders are helpful when spacing a glyph; for example, quickly switch from ‘ononnoon’ to ‘omommmoom’ if the n glyphs are placeholders.

4.9.2 Writing Direction

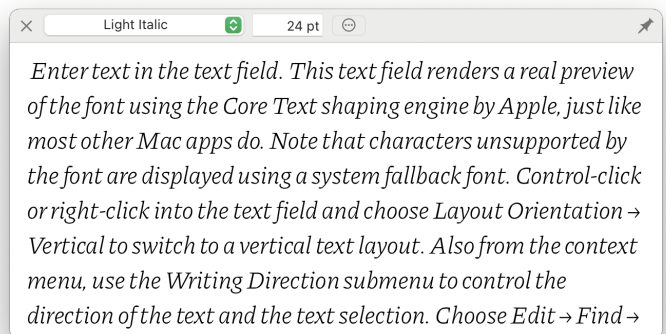


Switch between left to right, right to left, and top to bottom layout with the respective alignment buttons in the bottom-right corner of the Edit View window.

4.9.3 Text Preview


Keeping a lot of text in Edit View will slow down the interface. Instead, use the Text Preview (*Window > Text Preview*) for reviewing longer passages of text.


Tip: The Text Preview performs well with any amount of text, even the contents of an entire book.



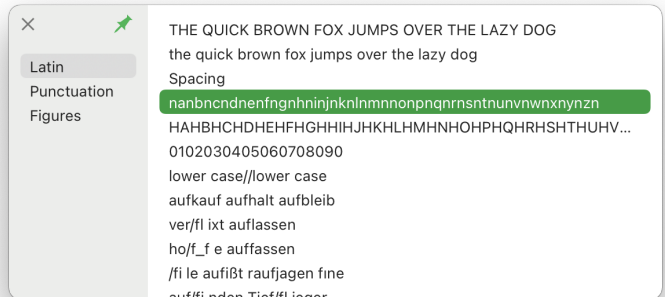
Enter text in the text field. This text field renders a real preview of the font using the Core Text shaping engine by Apple, just like most other Mac apps do. Note that characters unsupported by the font are displayed using a system fallback font.

Control-click or right-click into the text field and choose *Layout Orientation > Vertical* to switch to a vertical text layout. Also from the context menu, use the *Writing Direction* submenu to control the direction of the text and the text selection.


Choose *Edit > Find > Find...* (Cmd-F) to search for text or replace text in the text field. Click the magnifying glass  icon to set options for the search. Choose *Insert Pattern* or press Cmd-Ctrl-Opt-P to search for patterns such as spaces or digits.

Use the pop-up menu in the top-left of the window to pick the preview font style. The field to the right controls the font size at which the text preview is displayed. Click the pin  icon to toggle the pinned state of the window. A pinned window stays on top of the font window, allowing to continue editing glyphs while still keeping an eye on the text preview.

4.9.4 Sample Strings



Edit and store sample strings in *Glyphs > Settings... > Sample Strings* (see section 3.4, ‘Sample Strings’, p. 16). Insert a sample string by choosing *Edit > Select Sample Text...* (Cmd-Opt-F). Use the arrow keys or click to choose a string.

Switch to the next or previous sample string without the dialog by choosing *Edit > Other > Select Next Sample String* or *Select Previous Sample String*, respectively. Keyboard shortcuts to these commands can be assigned in the settings (see p. 20). Click the pin  button to keep the sample strings window open.

4.10 MEASURING

Glyphs offers several ways to determine coordinates and to measure distances between points and curves.

4.10.1 Measuring with the Info box

Toggle the display of the Info box with *View > Show Info* (Cmd-Shift-I). The Info box always displays data relevant to the current selection. If there is exactly one node selected, its coordinates will be displayed.




X 377
Y 610

Select a handle (aka off-curve point, or Bézier control point), and the Info box will also show its delta values (ΔX and ΔY difference to the on-curve point) and the total length of the handle (distance to the on-curve point).



X 233 ΔX 89
Y 144 ΔY 55 L 104.6

Tip: Quickly and precisely measure a stem or bowl width by selecting two nodes that indicate the width, and see what the Info box displays next to the width symbol.

The X and Y coordinates describe the position of the selection bounding box. The position is measured from the layer origin at (0, 0) to the part of the selection indicated by the blue point in the Info box. For example,  measures from the origin to the selection center.



X 620 \leftrightarrow 128 □ 38
Y 0 \updownarrow 458 ■ 5

See section 4.2.6, ‘Scaling & Rotating’ (p. 26) for more on the selection Info box.

Values in the Info box can be edited by clicking the number. Confirm a new value by pressing Return or by exiting the field. Use the Tab key to exit the current field and edit the next value in the Info box. Go back by pressing Shift-Tab. Pressing Escape exits the current field without entering a different field. Use the Up and Down arrow keys to increment or decrement the value of a field. Hold down Shift for increments of 10.

When a component is selected, the Info box of the base letter appears to the right. The base letter is the original glyph the component points to. The base letter Info box displays the glyph name, its X and Y offset, its horizontal \leftrightarrow and vertical \updownarrow scale in


percent, its slant to the right \nearrow , and counterclockwise rotation angle \curvearrowright . The arrow \rightarrow button in the top-right corner will insert the original glyph in the Edit tab string to the left of the current glyph and activate it for editing.

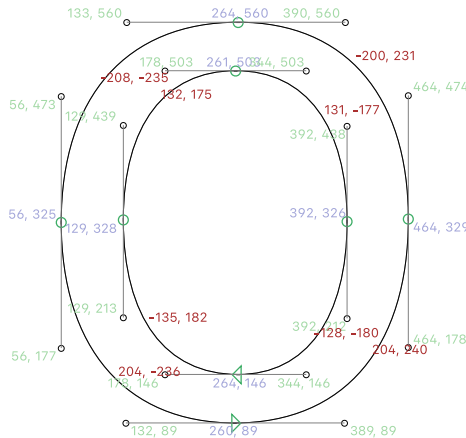


Change the glyph that the component points to by clicking its name in the Info box and choosing another glyph from the glyph list in the subsequent pop-up window. For more details on working with components, see section 10.1, ‘Components’ (p. 133).

Changing the position only has an effect if the component is not automatically aligned. For more details on automatic alignment, see section 10.1.8, ‘Automatic Alignment’ (p. 137).

4.10.2 Measurement Tool

Switch to the Measurement tool  (shortcut L) to see all coordinates of all nodes and anchors at once.

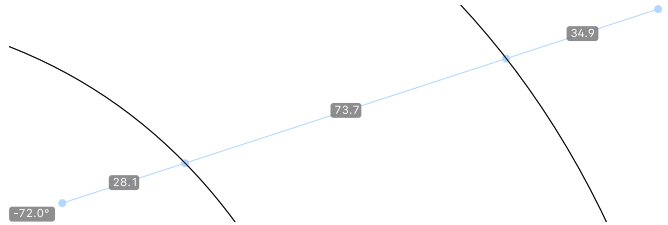


The blue numbers are the X and Y coordinates of the on-curve points, the green numbers are the coordinates of off-curve points, and the red numbers are the x and y delta values between the on-curve points.

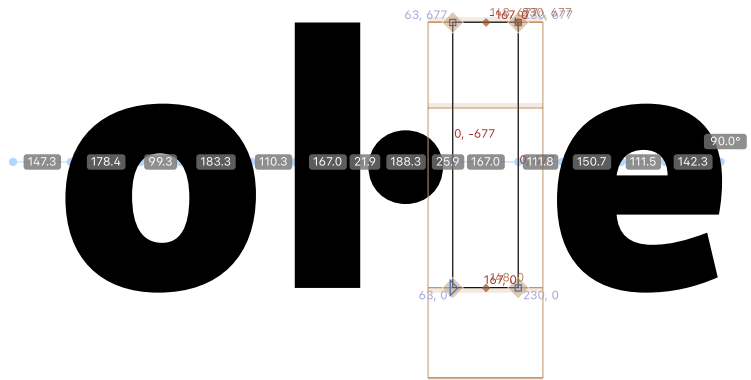
Tip: The X delta values respect the italic angle set in Font Info. So, an X delta of zero indicates a line exactly in the italic angle.

Clicking and dragging draws a ruler that displays precise measures between all of its intersections with the outlines. Add Shift to drag a horizontal or vertical ruler. At the end of the ruler, its angle is displayed in counterclockwise degrees, where zero

degrees corresponds to dragging the ruler perfectly vertically towards the top.




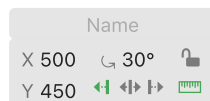
The Measurement tool works on all visible glyphs in Edit View. This also applies to the measurement line:



Temporarily activate the ruler and the display of point coordinates by simultaneously holding down Cmd-Ctrl-Opt. Press the G key while dragging a ruler to add a guide in measurement mode. Holding down Command temporarily switches to the Select tool.

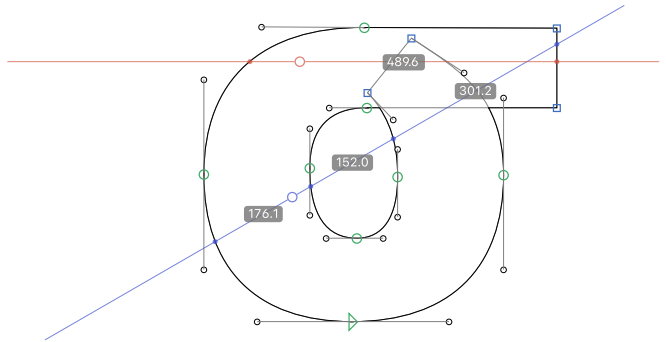
4.10.3 Measurement Guides

Local and global guides can be turned into a measurement guide. Click on the guide to select it and click the measurement symbol  in the Info box. See section 4.5, 'Guides' (p. 38) for more details.



Like the Measurement tool, guides in measurement mode will display the distance between their intersections with outlines or

components. Contrary to the tool, they always do so as long as guides are shown, regardless of which tool is active.

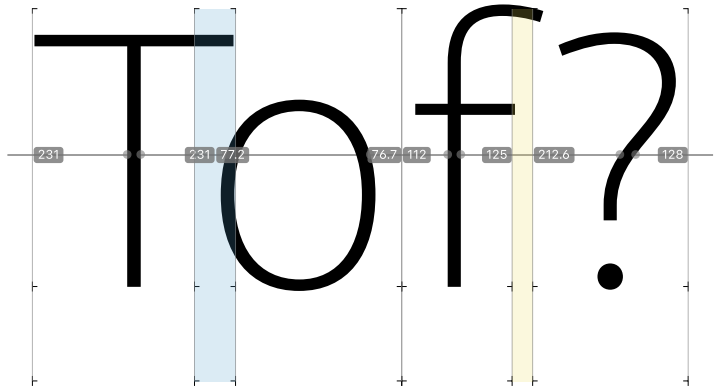


Note that the overlapping paths (such as the curl in the sigma pictured above) are ignored by the Measurement tool or measurement guides. Such overlapping paths are drawn with a light gray color and removed when exporting a font with the Remove Overlap filter enabled (see section 12.2.10, 'Remove Overlap', p. 187).


4.10.4 Measurement Line

When in text mode, enter measurement mode by choosing *View > Show Measurement Line*. The measurement line will display the sidebearings at a given height, ignoring the shape of the glyph at other positions. More precisely, the numbers displayed indicate the distance between the left or right sidebearing and the point where the measurement line first crosses the glyph outline. Alter the height of the measurement line by *Cmd-Ctrl-Opt-clicking* or *Cmd-Ctrl-Opt-dragging*. Or switch to the Measurement tool (L) and drag it to the desired height.

In measurement mode, thin gray lines indicate the widths of the glyphs. Kernings are color coded: Negative kerning is displayed as light blue, positive kerning as yellow. The colors can be changed in the settings (see p. 14).




4.11 ANNOTATING

The Annotation tool  (shortcut A) adds notes and correction marks to glyph layers. When the tool is active, the Info box (*View > Show Info*, Cmd-Shift-I) turns into a small palette holding a range of annotation tools.




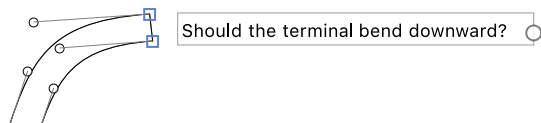
Choose *Edit > Select All* (Cmd-A) to select all annotations in the currently active glyph layer. Move selected annotations with the arrow keys. Hold down Shift for increments of 10, and Command for increments of 100 units. Press the Delete key to remove all selected annotations.

4.11.1 Annotation Cursor


The first tool in the Annotation palette is the Annotation Selection tool . Use it to click-select annotations. Shift-click to select multiple annotations at once.

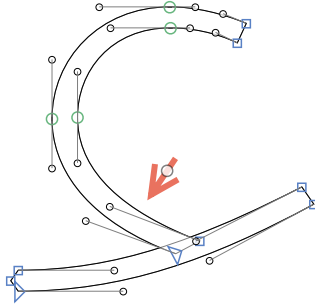
4.11.2 Annotation Text

Place snippets of text with the Text Annotation tool . Click to add a new text annotation, double-click an annotation to edit its text. A text snippet can span across multiple lines. The handle on the right controls the width of the text box.




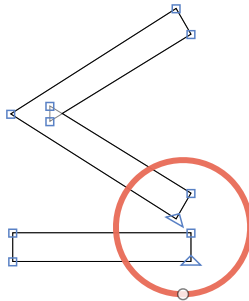
4.11.3 Annotation Arrow

The Arrow Annotation tool  adds red arrows onto the canvas. Use the handle on an arrow stem to control its rotation.





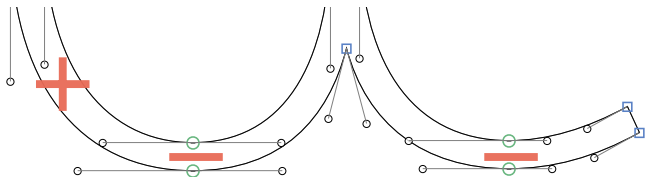
4.11.4 Annotation Circle

The Circle Annotation tool  adds red circles onto the canvas. The handle at the bottom of the circle controls its diameter.



4.11.5 Plus & Minus Annotations

Many designers use plus and minus signs to indicate that a counter, a bowl, or a stem should be thickened or thinned. Click the plus  or minus  button and then click on the canvas to add the symbols to the relevant area.



4.12 IMAGES

4.12.1 Adding Images

All image formats supported by macOS can be added to a glyph layer. These include JPEG, PNG, PDF, TIFF, and PSD files. Add an image by dragging the image file over a glyph cell in Font View or over a glyph in Edit View. Alternatively, choose *Glyph > Add Image...* to insert an image file.

In a Glyphs document, only the relative path to an image file is stored. Thus, it is a good idea to keep images in a subfolder next to the Glyphs file. If the path to the placed image is outdated or broken, it will be indicated with a missing image symbol:





Toggle the display of images with *View > Show Image*. Images are always displayed for empty glyphs (glyphs without paths or components).

Image files are ignored at export unless exporting a bitmap image font, like an sbix color font. See section 16.4, ‘sbix Fonts’ (p. 229) for details.



4.12.2 Manipulating Images



By default, images are scaled to a size where one DTP point corresponds to one font unit and placed at the origin point of the layer. Consider scaling images before importing them into Glyphs when preparing scans.

Move an image by dragging it to the desired position. When an image is selected, resize it with the bounding box (Cmd-Opt-Shift-B) or the Scale tool  (S). Rotate an image with the Rotate tool  (R). The *Transformations* palette also works for images.



The Info box (Cmd-Shift-I) offers controls for the image position (X and Y) and dimensions (↔ width and ↕ height) of the selected image. Rotate the image by changing the degree figure next to

the curved arrow . Clicking the right-pointing arrow  reveals the original image file in Finder.

Click the lock  icon to lock  the image. A locked image cannot be selected or manipulated. Unlock an image by Control-clicking or right-clicking the image on the canvas and choosing *Unlock Image*. Choose *Set Crop to Layer Bounds* from the image context menu to hide the image outside the layer bounds defined by its width, descender, and ascender. Choose *Reset Crop* to undo such a crop.

4.13 PREVIEWING & TESTING

4.13.1 Previewing Kerning

Kerning can be previewed with three modes: no kerning **To**, kerning **To⁺**, and locked kerning **To⁺**. See section 9.2.1, ‘Kerning Modes’ (p. 127) for details.

4.13.2 Previewing Masters

The glyphs in Edit View are drawn for the currently selected master. Change the current master by clicking the icon for the master in the toolbar.

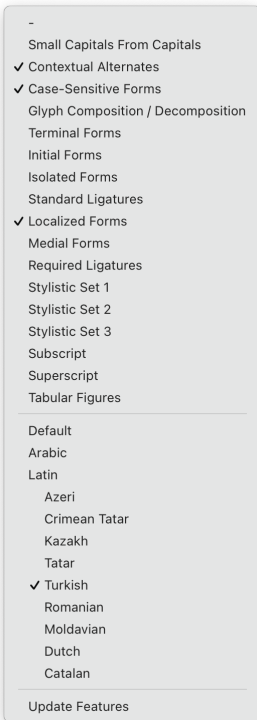


Set master icons in *File > Font Info... > Masters*. Switch to one of the first nine masters of the font by pressing Command and the number of the master: Cmd-1, Cmd-2, ..., Cmd-9. Change the master of the currently edited glyph or the selected glyphs in Edit View by selecting a master layer from the *Layers* palette. For more details on working with multiple font masters, see chapter 11, ‘Interpolation’ (p. 156).

4.13.3 Previewing OpenType Features

Toggle OpenType features from the *Features* menu at the bottom-left of the window in Edit View. When at least one feature is selected, the Features menu is highlighted in an accent color. The button will show the four-letter tags of the active features.

Activate a feature by selecting it from the Features menu. Multiple features may be active at the same time. Select a feature from the menu again to deactivate it in Edit View. Deactivate all features by choosing the dash (-) at the top of the menu.




Recompile the font features from the features editor (*File > Font Info... > Features*) to make newly added features appear in the menu. Quickly select an item from the menu by typing its name while the menu is open. This also works with the dash.



Preview language-specific forms by enabling *Localized Forms* from the menu. Pick the desired script and language from the bottom of the features menu. For this to work, a ‘locl’ feature with valid language-specific rules must be present in *File > Font Info... > Features*.

In Edit View, Glyphs previews substitution, kerning, and curvise attachment positioning features. Other positioning features may be handled differently in third-party applications and are not previewed in Edit View. Test these features directly in software by exporting the fonts. When testing in Adobe applications, export the fonts to the Adobe Fonts folder. For details, see section 4.13.6, ‘Previewing in Adobe Applications’ (p. 58). Note that in InDesign, the OpenType features may be interpreted differently by different compositors.

4.13.4 Previewing Interpolated Instances


Click the Preview button  in the bottom-left of the window to preview font instances. The window content will be split with Edit View above and Preview Area below. Drag the separator to adjust the size of the Preview Area.

Alternatively, open a separate window which can also be placed on a second display via *Window > Preview Panel*. The Preview Panel displays the Edit View glyphs of the current font file. When no Edit View tab is currently active, the Preview Panel stays blank. Click an Edit View tab, and the Preview Panel will immediately update to show the respective glyphs.

Select the instance to be previewed from the pop-up menu in the bottom-left of the preview. *Show All Instances* displays the current glyph across all instances of the font. Toggle the display of individual instances in *Show All Instances* by clicking the eye /  icon next to an instance name. Select the dash (-) for the preview to mirror the master used in Edit View.

Except for the *Show All Instances* option, the Preview Area and Preview Panel render the complete text of the current Edit View tab on a single line. The current glyph is centered by default. Drag the text horizontally in the Preview Area to reposition it. Double-click a previewed glyph to edit it. The rendering respects

some custom parameters, as well as intermediate and alternate layers. Control-click or right-click the Preview Area and choose *Always Center Active Glyph* to keep the active glyph centered. If deactivated, Glyphs will try to fill the Preview Area as best as possible, keeping the text flush left or flush right.

Switch between black-on-white and white-on-black with the Invert  button next to the instances pop-up menu. With the Flip **F/E** button, flip the Preview upside-down. Flipping text can be helpful when testing the spacing of a font. Test the legibility of the font by blurring the font sample in the Preview with the slider next to the Flip button.

4.13.5 Previewing on macOS

Applications on the Mac cache font files. Caching fonts is an optimization that makes sense for most Mac users, since those fonts are typically installed just once and accessed many times by many applications. However, when creating fonts with Glyphs, the fonts that have already been cached prevent their subsequent exported versions from showing up in apps. There are multiple approaches to bypass font caching.

Firstly, quit and relaunch the app in which the fonts are previewed. For example, when testing a font in Pages, save the document and quit Pages. Export the new version of the font and relaunch Pages.

If this approach does not work, check *Test Install* in the OTF export dialog. This option writes the font data directly to the system memory without creating a new font file. Quitting and relaunching may still be required for the new font version to show up.

If the new font versions still do not show up, restart the Mac. In case that does not fix the issue either, uninstall the affected fonts from the system by deleting them in Font Book. Then, quit all applications and launch the Terminal application. Enter the following lines and press the Return key after each line. If prompted for the user password, note that entering the password does not display the typical ••• bullet points. Press Return to confirm the password.

```
sudo atsutil databases -remove
atsutil server -shutdown
atsutil server -ping
```

After running the three lines above, restart the Mac for the changes to take effect.

If the problem persists, reboot the Mac in safe mode by pressing `Cmd-S` at startup until the Apple logo appears. Following that, recreate the caches by holding down `Shift` while the computer completes the restart and keep holding it while logging in.

4.13.6 Previewing in Adobe Applications

For a complete font preview, including positioning features and menu order, open the export dialog (*File > Export*, `Cmd-E`) and choose the OTF export.

Set the path in *Export Destination* to `/Library/Application Support/Adobe/Fonts`. For this, first, click the folder path to open the folder browser. Then, press `Cmd-Shift-G` and enter the path of the Adobe Fonts folder exactly as above.

Click *Go* and then click *Open* to choose the Adobe Fonts folder and export the fonts by clicking *Next...*

The font becomes immediately available in all Adobe applications. Glyphs will overwrite any previously saved instance of the font in this folder. The font will not be available outside Adobe apps, but this is a convenient way to circumvent any font cache problems in macOS.

Adobe apps may need to be closed and relaunched the first time the Adobe Fonts folder is used. After that, the Adobe font menus will update immediately every time the font is exported.

If the error message 'The folder can't be found.' appears, remove the `/Fonts` at the end of the text field and press *Go* again. Now create the folder by clicking *New Folder* or by pressing `Cmd-Shift-N`. Enter 'Fonts' and confirm with *Create*.

4.13.7 Previewing in Web Browsers

In the export dialog (*File > Export*, `Cmd-E`), check the `.woff2` option to export to WOFF2 font files. A `.woff` option for older browsers is also available.


These webfont files can be loaded into an HTML webpage and previewed in a web browser. Reload the webpage to see the new versions of the exported fonts. The font cache of many browsers can be bypassed by holding down the `Shift` key while reloading.

A simple HTML file might look like this:

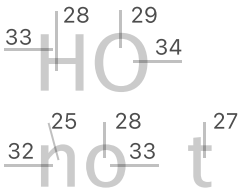
```
<!DOCTYPE html>
<html lang="en">
  <meta charset="utf-8">
  <title>Webfont Preview</title>
  <meta name="viewport" content="width=device-width">
```

```
<style>
  @font-face {
    font-family: 'Some Font Name';
    src: url(some-font-name.woff2) format('woff2');
  }
  html {
    font-family: 'Some Font Name';
  }
</style>
  This is the test text for the webfont preview.
</html>
```

5 Palette

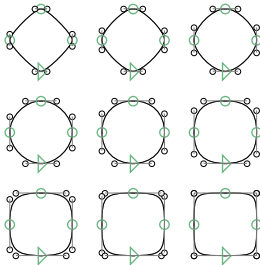
Open the Palette sidebar with the sidebar button  in the top-right corner of the window, or choose *Window > Palette* (Cmd-Opt-P). Glyphs includes four sections: *Dimensions*, *Fit Curve*, *Layers*, and *Transformations*. Plug-ins can add additional sections to the Palette (see p. 237). Collapse or expand a section by clicking the chevron on the left side of the section name.

5.1 DIMENSIONS




The *Dimensions* section does not affect the font but serves as a notepad where the dimensions of common font features can be written down and referenced. Enter a value by clicking a number or an empty field ('--'). Values are stored per master. The fields change according to the script attributed to the glyph. It can be changed through *Edit > Info for Selection* (Cmd-Shift-I).



5.2 FIT CURVE



Fit Curve from 20 % to 100 %



The *Fit Curve* section helps to create curves with a smooth curvature. Clicking one of the eight round buttons  changes the length of the selected handles. The leftmost button sets the handles to the length specified in the left field, and the rightmost button matches them to the right field. The intermediate six buttons set the handles to intervals of even lengths. Alternatively, activate the eight buttons with the shortcuts Ctrl-Opt-1 through Ctrl-Opt-8.



The plus  and minus  buttons uniformly increase or decrease the handle's length of the selected curve segments. *Fit Curve* always works on both handles of a segment, even if only one is selected.


When a handle is selected, a small gray indicator located under the buttons shows the current handle length. The minimum value is 1 percent, the maximum value is 100 percent. A length of 55 % is the closest approximation to an elliptical curvature or a circle. Such a curve segment appears equally curved across its entire length. Curves with longer handle lengths (above 55 %) appear flattened towards their on-curve points, making them more apt for connecting to line segments. Curves with shorter handle lengths (below 55 %) appear flat in the middle and highly curved towards the ends.

Add a ‘Fit Curve Panel Settings’ custom parameter to the font to set per-document minimum and maximum values. The value are two numbers, separated by a comma, like ‘55, 85’.

5.3 LAYERS

A layer contains the outline of a glyph. A single glyph can have multiple layers, at least one for each master.

Layers are shown in the *Layers* section of the Palette. The layers are sorted by their master. Drag the handle at the bottom of the *Layers* section to resize it. Click the eye symbol  next to a layer to toggle its display. A visible  layer is displayed with a faint blue line when a different layer of the glyph is active.


Change the color in the settings (see p. 14). Use the Select All Layers tool  (shortcut Shift-V) to edit all visible layers at once. See section 11.8.1, ‘Select All Layers Tool’ (p. 169) for details.

Shift-click to select a range of layers, or Command-click to make a non-contiguous selection. Reorder layers by dragging a layer to a new position. Layers can only be reordered within a master.



Glyphs differentiates between three types of layers: *master* layers, *backup* layers, and *special* layers.

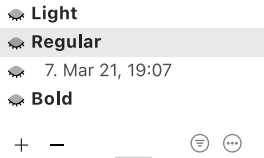
5.3.1 Master Layers

Master layers are required for interpolating instances. Every glyph has a master layer for every font master. In the *Layers* palette, master layers are labeled with the master name and are set in bold. Master layers cannot be added, deleted, or renamed from the *Layers* section. Instead, open *File > Font Info... > Masters* to manage font masters.

Choose *Only Show Layers from Current Master* from the filter  menu to hide all layers not belonging to the current master. This option can clean up the layers list and is particularly helpful when working on fonts with many masters.

5.3.2 Backup Layers

Backup layers are used to keep a copy of previous drawings. Click the plus  button in the bottom-left of the *Layers* section to create a new backup layer. By default, a backup layer is labeled with its creation date and time. Double-click the layer name to rename it. Click the minus  button to delete the selected layer. Backup layers are indented below their master layer and set in a



regular weight. A glyph can have any number of backup layers. Choose *Hide Backup Layers* from the filter (☹) menu to hide all backup layers.

Revert to a backup layer by choosing *Use as Master* from the actions (☹) menu. The backup layer will be deleted, its contents will be placed on the master layer, and the contents of the current master layer will be placed on a new backup layer. Alternatively, drag a backup layer onto a master layer to use it instead as a master layer.

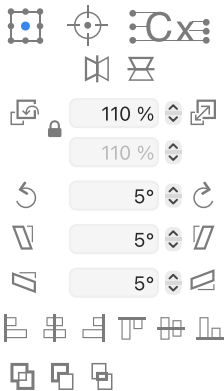
5.3.3 Special Layers

Special layers can be intermediate and alternate layers for interpolation or any kind of color layer. Special layers are indented below their master layer. The label reflects their type and is set in bold.

Intermediate and alternate layers are used in Multiple Master setups. See chapter 11, ‘Interpolation’ (p. 156) for more details. Color layers are used for color fonts. See chapter 16, ‘Color Fonts’ (p. 224) for more details. Plug-ins may define their own special layers. Refer to their documentation for more information.

5.4 TRANSFORMATIONS

The *Transformations* section transforms points, paths, components, anchors, guides, images, and more. The following transformations are supported:



- ▶ mirror the selection horizontally or vertically
- ▶ scale or reverse scale the selection;
- ▶ rotate the selection clockwise or counterclockwise
- ▶ slant the selection left , right , down or up
- ▶ align the selection left , right , to the top or bottom , center it vertically or horizontally
- ▶ perform boolean operations (union, subtraction, intersection) on paths.


Transformations can be applied in both Font View and Edit View. According to the selection, the transformation applies to segments, paths, or complete glyph layers.




The buttons to the left and right of text fields perform opposite transformations. For example, clicking undoes the transformation caused by clicking . These opposing



transformations are not subject to rounding errors that might be introduced by the unit grid. See section 7.5.1, ‘Grid Spacing & Subdivision’ (p. 107) for details on the unit grid.

5.4.1 Transformation Origin




All transformations (except for alignment) are performed with respect to a transformation origin. The top row of the *Transformation* palette controls this origin point.

The **transformation box**  places the transformation origin with respect to the selection bounding box. Either one of the four corners, or the center of one of the four edges, or the center of the selection box can be defined as the transformation origin.

The **reference point**  uses the transformation origin as defined by the Scale tool (S) or the Rotate tool (R). Using one of those tools, click on the canvas to define the transformation origin. The point is indicated as a red ring  with a crosshair when using the Scale/Rotate tool, and just as a crosshair  when using any other tool.

The **metrics point**  places the transformation origin at one of the metrics defined in *File > Font Info... > Masters > Metrics*. Choose either the baseline, half or full x-height, or half or full cap height. For CJK glyphs, the layer dimensions control  is displayed instead. It places the transformation origin at the center of the layer (half of the width and centered between the ascender and the descender).

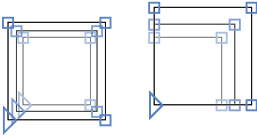
5.4.2 Mirroring



The mirror buttons / reflect the selection to the other side of the transformation origin. Horizontal  mirroring reflects across the horizontal center of the layer when a metrics point defines the transformation origin.



Mirror with the transformation origin positioned in the metrics point control at either half the x-height or half the cap height to preserve overshoots. For example, mirroring an n that overshoots the x-height will result in an u shape that overshoots the baseline.






Mirroring a corner component (see section 10.3, ‘Corner Components’, p. 145) will turn a left corner into a right corner and vice versa. Mirroring a glyph component upside-down will reverse the assignment of top and bottom anchors in the original glyph. This way, a top mark can be used as a bottom mark by mirroring the component.

5.4.3 Scaling









Scaling by 120 % using an origin of  and . Reverse scaling is helpful since a scale of 120 % would not be undone by simply scaling to 80 %, but by scaling to 83.3... % (= 100 %/120 %).

Click the scale  button to scale the selection by the specified percentage. Percentages larger than 100 % increase the selection size, percentages smaller than 100 % decrease the size, and a percentage of 100 % does not change the size. The scaling factor along the horizontal axis (top field) and the vertical axis (bottom field) can be defined independently. The top field is used for both axes if the lock  icon is locked.







The reverse scale  button undoes a scaling operation. This is not the same as scaling down: Scaling with  by a percentage smaller than 100 % (for example, 75 %) shrinks the selection. Scaling with  by the same percentage returns the selection to its original size. Similarly, a selection that was enlarged to a value larger than 100 % with  can be undone using .

5.4.4 Rotating & Slanting

Slanting is sometimes also referred to as *skewing*.




Rotate the selection with  and . Slant the selection with , , , and . Note that slanting a selection twice by a certain amount does *not* yield the same result as slanting it once with double the amount.

5.4.5 Aligning

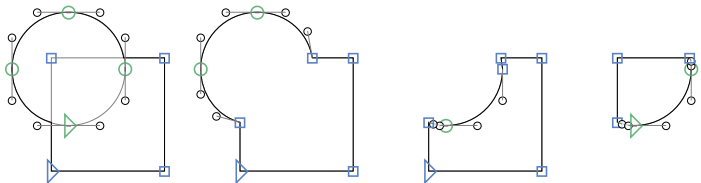
The align buttons (, , , , , ) can be used on points, complete and partial paths, anchors, and components. Aligning is always done relative to the bounding box of the selection.

Quickly align the selected points with *Path > Align Selection* (Cmd-Shift-A). This command respects the setting for the transformation origin. See section 4.2, 'Editing Paths' (p. 23) for more details.

5.4.6 Boolean Operations

The bottom row of buttons combines closed paths with boolean operations: , , and .

From left to right: two overlapping paths, union, subtraction, and intersection.





Union removes the overlaps in all selected paths (or all paths of the layer if there is no selection). **Subtraction** removes the selected paths from the unselected paths. Paths involved in the subtract operation are merged first to achieve consistent results. **Intersection** keeps only the parts where the selected paths overlap the unselected paths. Both subtraction and intersection use the frontmost path as the selected path if there is no selection.

A union operation can be applied to all glyphs on export using the *Remove Overlap* option. See section 8.1.1, ‘Options’ (p. 111) for details.

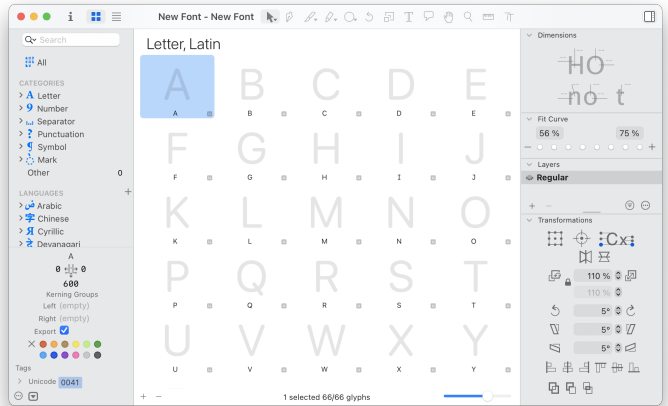
6 Font View



Font View provides an overview of all glyphs of a font. It is displayed when a new Glyphs file is created. If multiple tabs are open, jump to Font View by clicking the leftmost tab or pressing `Cmd-Opt-1`.

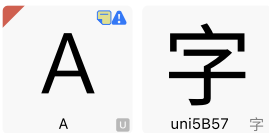
6.1 VIEWING GLYPHS


Font View is available in two viewing modes: Grid View and List View. Toggle between the modes with the grid  and list  buttons in the top-left of the font window.

6.1.1 Grid View








In Grid View , the glyphs of the font are displayed in a grid of glyph cells. The glyph outlines reflect the currently selected master. Use the slider  in the bottom-right to control the cell size. Large glyph cells display the glyph outlines, the glyph name, and the Unicode value of the glyph. Small cells show only the outlines.



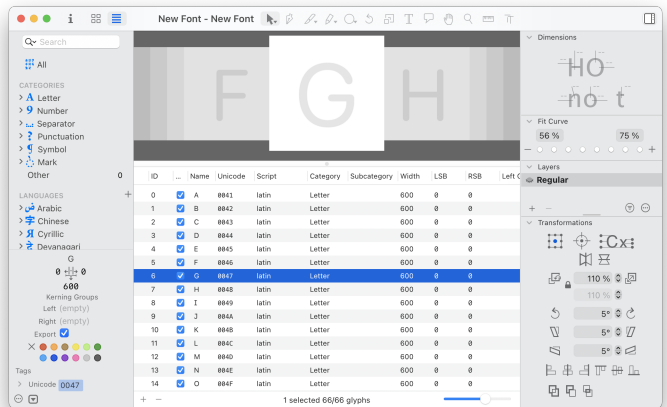
The Unicode value is shown in the bottom-right next to the glyph name. By default, a small Unicode indicator  is displayed. The Unicode indicator is replaced by a small rendering of the glyph set in the system font for CJK glyphs. Enable *Display Unicode Value* in settings to show the full Unicode value (see section 3.2, 'Appearance', p. 14). Alternatively, select a glyph cell to show its Unicode values in the inspector located in the


bottom-left of the window (see section 6.3.7, ‘Unicode’, p. 73 for details).

A warning  triangle in the top-right corner of a glyph cell indicates that the glyph has out-of-sync metrics keys (see p. 124). A yellow note  indicates that the glyph has an annotation on the active master layer. A black info  indicates that the glyph uses custom glyph properties (see p. 71). A red top-left corner  in a glyph cell indicates that not all layers are compatible for interpolation (see p. 162). A red ring with stroke  marks a non-exporting glyph (see p. 98).

A glyph cell displays its glyph and layer colors. See section 4.7.4, ‘Glyph & Layer Colors’ (p. 43) for details.

6.1.2 List View



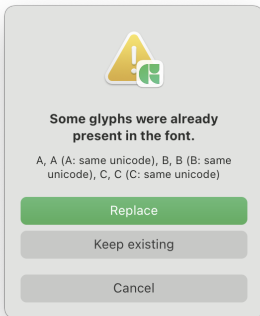
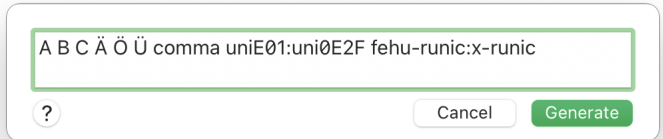
Switch to List View by selecting the list  icon located in the top-left of the font window. List View shows a table where each row corresponds to a glyph and each column to a glyph property. Click a column header to sort by that column. Click a header again to reverse the sort order. Click and drag a column header to rearrange columns. Control-click or right-click a header to hide or show columns. See section 6.3, ‘Glyph Properties’ (p. 71) for a description of all glyph properties.

6.2 MANAGING THE GLYPH SET

6.2.1 Adding New Glyphs

New glyphs can be added to a font in various ways:

Adding Multiple Glyphs *Glyph > Add Glyphs...* (Cmd-Shift-G) opens a dialog window for adding glyphs. Write glyph names (Aacute, ampersand, noon-ar.init), characters (Á, &, ÿ), or Unicode values (uni00C1, uni0026, uni0064) separated by spaces into the text field. Specify character ranges by placing a colon between two glyphs: ‘uni01FC:uni01FF’ or ‘Abold-math:Zbold-math’. The text field also accepts glyph recipes. See section 10.1.3, ‘Recipes’ (p. 134) for details.

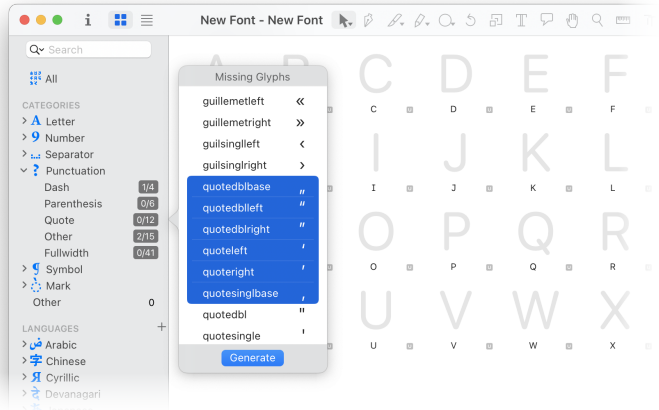


When adding glyphs that are already in the font, they will not be added again. Instead, a dialog is shown listing the glyphs that are already part of the font. Choose *Replace* to replace the existing glyphs with new, empty glyphs. *Keep Existing* ignores the duplicate glyphs and only adds the new glyphs. Choose *Cancel* to not add any new glyphs.

Adding a Single Glyph *Glyph > New Glyph* (Cmd-Opt-Shift-N) or the plus $+$ button located in the bottom-left of Font View adds a new empty glyph named ‘newGlyph’.

Adding From the Glyph Info Database Choose *Window > Glyph Info* to show a list of all glyphs known to Glyphs. Click *Add to Font* to add the selected glyphs to the current font. See section 6.4.1, ‘Glyph Info Database’ (p. 77) for details.

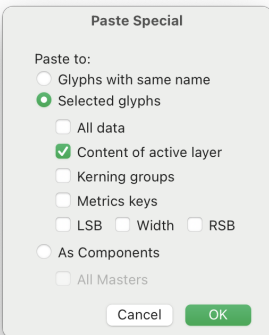
Adding From the Sidebar Some entries in the sidebar show a badge with the total glyph count for the entry and the count of glyphs that are already part of the font. Control-click or right-click a sidebar entry to show a list with the missing glyphs. Select glyphs from the list individually, or select all by pressing Cmd-A. Add the selected glyphs to the font by clicking *Generate*.



6.2.2 Copying Glyphs Between Files



Glyphs can be copied across font files. In Font View, copy the selected glyphs with *Edit > Copy* (Cmd-C) and paste them into another file with *Edit > Paste* (Cmd-V). Components are re-linked to the glyphs in the target font. When a linked glyph does not exist in the target font, the component cannot be re-linked and displays as a ‘no base glyph’ triangle placeholder instead. If a glyph with the same name already exists in the target font, an incrementing suffix like ‘A.001’, ‘A.002’, ... will be added to the name of the pasted glyph.



Overwrite existing glyphs with *Edit > Paste Special* (hold down Option while the menu is open or press Cmd-Opt-V). The *Paste Special* dialog presents three paste modes:

Glyphs With Same Name overwrites the glyphs that have the same names as the copied glyphs. Any glyphs that do not yet exist in the target font are also added.

Selected Glyphs overwrites all currently selected glyphs in the target font with the glyphs copied from the source font. The names of the selected glyphs are kept. If more glyphs were copied, then there are selected glyphs in the target font, the additional glyphs are ignored.

As Components pastes the copied glyphs as components into the selected glyphs. Choose *All Masters* to paste on all masters. Existing paths and components are preserved.

For the options *Glyphs With Same Name* and *Selected Glyphs*,

glyphs can be partially pasted by selecting the desired parts: *All Data* pastes the entire glyph, including shapes, anchors, layer attributes, and metrics. This option includes all the options below it.

Content of Active Layer pastes the currently displayed layer.

Kerning Groups pastes the left and right kerning groups. This option is useful for reduplicating kerning between similar fonts.

Metric Keys pastes the sidebearing and width formulas.

Recalculate them after pasting with *Glyph > Update Metrics* (Cmd-Ctrl-M) or across all masters with *Glyph > Update Metrics for all Masters* (Cmd-Ctrl-Opt-M).

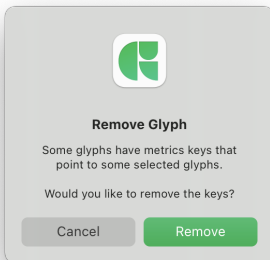
LSB, *RSB*, *Width* paste the left sidebearing, right sidebearing, and width metrics of a glyph. If both *RSB* and *Width* are selected, the right sidebearing value takes precedence.

6.2.3 Deleting Glyphs

Glyph > Remove Glyph (Cmd-Delete) deletes the selected glyphs from the font. This command also works on the current glyph in Edit View and selected glyphs with the Text tool active. In Font View, the minus — button in the lower-left of the window does the same. Before the selected glyphs are deleted, a confirmation dialog is shown. Confirm with *Remove* or keep the glyphs with *Cancel*.

If some of the glyphs are used as components, the dialog offers to decompose the components before deleting the glyphs. Click *Remove* to delete the glyphs and all components referencing them, or click *Decompose* to delete the glyphs after decomposing all components of those glyphs to outlines.

If the metrics keys of other glyphs were linked to the metrics of a deleted glyph, a dialog offers to unlink those metrics. For example, in a font with two glyphs A and AE (Æ), where the left sidebearing of AE is linked to the one of A, deleting A means that AE can no longer reference the metrics of A. The dialog appears with two choices: *Remove* dissolves the metrics keys linked to the deleted glyphs. *Cancel* keeps the metrics keys, even though they are no longer linked to a glyph. Such unlinked keys are set in a turquoise color:





Fix unlinked metrics keys by linking them to a different glyph, replacing the keys with numeric values, or creating the glyph they are pointing to. See section 9.1.3, ‘Metrics Keys’ (p. 124) for more details.

6.3 GLYPH PROPERTIES

Glyphs have multiple properties associated with them, such as their name, metrics, Unicode values, and whether the glyph will be exported. In Font View, glyph properties are shown in the bottom-left corner of the window in the glyph inspector. If the inspector is closed, open it with the disclosure button. Glyph properties are also shown in the columns of List View (see p. 67) and the Selection Info window (*Edit > Info for Selection*, *Cmd-Opt-I*). Some properties are also accessible from the context menu on a glyph in Font View and Edit View.

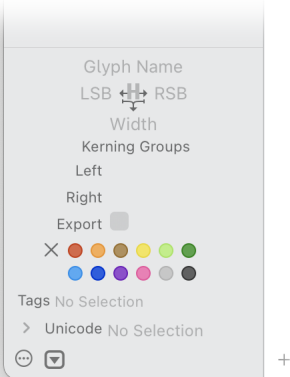
The glyph inspector and the Selection Info window show the properties of the selected glyphs. Click the disclosure chevron **>** next to the *Unicode* field in the glyph inspector to show additional controls for the production name, script, category, and subcategory.

When multiple glyphs are selected with different values for the same property, a mixed value is displayed in the inspector. For text fields, *Multiple Values* is shown in gray. Editing such a text field will overwrite the current value for all selected glyphs with the new value. The export checkbox uses a line icon **—**, indicating that some selected glyphs export and some do not.

6.3.1 Glyph Name



Glyphs are identified by their name. Therefore, a font cannot contain two glyphs with the same name. The glyph name (or *nice name*) is used within Glyphs, while a separate *production name* is used for exported fonts. See section 6.3.8, ‘Production Name’ (p. 74) for details on production names and section 6.4.2, ‘Naming Glyphs’ (p. 78) for general information about the glyph naming scheme.

Select glyphs one at a time to display and change their names, or batch rename glyphs with *Edit > Find > Find and Replace*



(Cmd-Shift-F). See section 6.5.3, ‘Batch-Renaming Glyphs’ (p. 83) for more details.

6.3.2 Metrics

The metric fields show the horizontal  metrics (LSB, RSB, width) or the vertical  metrics (TSB, BSB, vertical origin, vertical width) of a glyph. Click the icon in the middle to switch between the horizontal and vertical metrics. See chapter 9, ‘Spacing & Kerning’ (p. 123) for information on metrics.

6.3.3 Kerning Groups

Use kerning groups to kern multiple glyphs by the same amount. See section 9.2.4, ‘Kerning Groups’ (p. 128) for details.

6.3.4 Exports

The *Exports* checkbox controls whether or not a glyph should be included in exported font files. Disable this option for component glyphs and any other glyphs that should not be exported. Use the ‘Remove Glyphs’ custom parameter on an instance to remove additional glyphs for that instance only. A glyph is exported by default unless its name starts with an underscore (‘_’).

6.3.5 Color Label

From left to right:

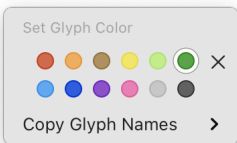
glyph color only (schwa),
no glyph or layer color (f),
glyph and layer color (g),
layer color only (gbreve).



Glyphs and glyph layers can be marked with color labels. Color labels do not affect the exported fonts but are used to organize glyphs during font development.

Control-click or right-click a glyph to apply one of the twelve predefined colors. Click the cross **X** button to remove the color from a label. Hold down the Option key to define or clear the color label of the current layer.

The glyph color spans across the entire glyph cell. The layer color is drawn on the right half of the cell if a glyph color is set, or across the entire cell with a cut-out in the top-left if no glyph color is set. Filter glyphs with a particular color using Smart Filters. See section 6.6.4, ‘Smart Filters’ (p. 86) for details.



In some cases, such as Smart Filters or scripts, color labels are addressed by name or number. The following names and numbers are used: ● 0 Red, ● 1 Orange, ● 2 Brown, ● 3 Yellow, ● 4 Light Green, ● 5 Dark Green, ● 6 Cyan, ● 7 Blue, ● 8 Purple, ● 9 Pink, ● 10 Light Gray, ● 11 Dark Gray.

6.3.6 Tags

has hook narrow rounded

Tags are short descriptions that can be attached to a glyph. A glyph can have multiple tags. Edit tags in the *Tags* field in the inspector located in the bottom-left of the font window. Tags can include letters, numbers, spaces, and other punctuation marks. Only the comma character (,) is special: Typing a comma or pressing Return will add the tag to the selected glyphs and display it in a blue capsule shape. The *Tags* field shows the tags of all selected glyphs if they have the same tags. Otherwise, the text *Multiple Values* is displayed.

Control-click or right-click a glyph in Font View and choose the *Tags* menu. It shows all tags used within the font. If no tags are used in the font, then the *Tags* menu is not shown. Tags included in any of the selected glyphs are marked with a checkmark ✓ next to the tag name. A dash indicates that only some of the selected glyphs include that tag. Select a tag from the menu to add it to all selected glyphs.

Tags can be used to filter glyphs, including in Smart Filters (see p. 86), metric scopes (see p. 95), and in OpenType feature tokens (see p. 189).

6.3.7 Unicode

A Unicode value (or *code point*) is a unit of text typed by a user on a keyboard or stored in a text file. The Unicode standard defines a code point for most written symbols used across the globe. Unicode values are written as ‘U+’ followed by a hexadecimal number (digits 0–9 and A–F). For example, U+0041 is the Latin capital letter A, U+1E900 is the Adlam capital letter Alif, and U+061F is the Arabic question mark. The font defines which glyph is displayed for a specific Unicode value.

		
A U+0041	Alif-adlam U+1E900	question-ar U+061F

Typically, one glyph has one Unicode value. In an all-caps font without lowercase letters, the A glyph might have two Unicode values: U+0041 and U+0061 (uppercase and lowercase Latin letter A). This way, typing either ‘A’ or ‘a’ displays the A glyph. Some glyphs have no Unicode values and are only accessible

using OpenType features (see p. 101). Glyphs that are solely used as components also have no Unicode values (see section 10.1, ‘Components’, p. 133).

Glyphs are assigned Unicode values based on their name. The mapping from glyph names to Unicode values is stored in the glyph info database. See section 6.4, ‘Names & Unicode’ (p. 77) for more details.

The Unicode values of a glyph can also be customized per Glyphs file. Select a glyph and edit Unicode values in the *Unicode* field of the inspector. In List View, the code points can also be edited in the *Unicode* column. Alternatively, choose *Edit > Info for Selection* (Cmd-Opt-I) and set the Unicode values of the selected glyph there.

For custom glyphs that are not defined in the glyph info database, choose a fitting name (see p. 78), and set its Unicode value manually. If the Unicode standard does not offer a suitable code point for a glyph, consider encoding it with a Private Use Area code point. When adding a custom glyph, make sure not to use the name of a registered glyph from the glyph info database.

Glyphs without a well-defined name (such as most Chinese characters) are named after their Unicode value: ‘uni’ followed by four hexadecimal digits or ‘u’ followed by five or six digits. For example, uni5B57 for U+5B57 and u20547 for U+20547.

Unicode defines some ranges (such as U+E000–U+F8FF) as *Private Use Area* (PUA) where the Unicode values have no predefined meaning. PUA code points are generally used for languages that are not in Unicode or icon fonts.

6.3.8 Production Name

Not all nice names used within Glyphs (see p. 71) are compatible with every app. Instead, production glyph names are used for exported font files. These production names are based on the Adobe Glyph List Specification (AGL).¹ Many production names follow the naming scheme as described in section 6.3.7, ‘Unicode’, p. 73: ‘uni’ followed by four hexadecimal digits or ‘u’ followed by five or six digits.

Production names are used automatically on export, while nice names are used in the Glyphs interface. All glyphs have a default production name, but it can be changed in the expanded inspector (expand it by clicking the > disclosure chevron) or by choosing *Edit > Info for Selection* (Cmd-Opt-I).

1 github.com/adobe-type-tools/agl-specification

6.3.9 Script

The script property defines the writing script the glyph belongs to. The glyph G is ‘latin’, Ya-cy is ‘cyrillic’, noon-ar is ‘arabic’, koKai-thai is ‘thai’, and so on. Glyphs belonging to multiple scripts—like digits (one, two, ...), punctuation (comma, hyphen, ...), or spaces—do not have a script value.

6.3.10 Category & Subcategory

The category and subcategory of a glyph groups it with related glyphs. All exporting glyphs belong to a category by default, while non-exporting glyphs such as components do not. The glyph T belongs to the category *Letter*, six to *Number*, and plus to *Symbol*. Some glyphs additionally have a subcategory for finer differentiation.

- ▶ guillemetright (‘»’) is defined as *Punctuation, Quote*
- ▶ parenleft (‘(’) is defined as *Punctuation, Parenthesis*
- ▶ hyphen (‘-’) is defined as *Punctuation, Dash*

6.3.11 Case

Some scripts differentiate glyphs by case. The case of a glyph can be *uppercase*, *lowercase*, *small capital*, *minor*, or no case. *Uppercase* and *lowercase* are used for letters in bicameral scripts such as Greek or Latin, but also for punctuation (for example, question (?) is uppercase and questiondown (¿) is lowercase) and marks (lowercase: acutecomb, uppercase: acutecomb.case). Glyphs ending in ‘.sc’ or ‘.smcp’ are assigned the *small capital* case. The *minor* case is used for subscript and superscript numbers (five.sups), letters (ainferior), and punctuation (equalinferior). All other glyphs have no case, including glyphs from scripts without case (Arabic or CJK) and most punctuation glyphs (for instance, comma or euro).

6.3.12 Writing Direction

A glyph has a writing direction of either left to right (LTR), right to left (RTL), or bidirectional (BiDi). Many punctuation glyphs are bidirectional (for example, hyphen, asterisk, and underscore). See section 4.9.2, ‘Writing Direction’ (p. 46) for setting the writing direction in Edit View.

6.3.13 Sort Name

The *Sort Name* property of the selected glyph can be edited in the Selection Info window (*Edit > Info for Selection*, Cmd-Opt-I). It defines the name by which the glyph is sorted in Font View and the exported font files. An empty *Sort Name* field indicates that the glyph name is used for sorting instead. Select the checkbox next to the sort name field to edit it. Sort names are not used in list filters (see p. 88) or when using a custom glyphs order (see p. 89).

6.3.14 ID

In List View, the *ID* column sorts glyphs in glyph order. This is the order in which glyphs are sorted in exported fonts. Change the order by setting the sort name of glyphs or using the ‘glyphOrder’ custom parameter in *File > Font Info... > Font*.

6.3.15 Char

In List View, the *Char* column displays the Unicode character using the system font. If a glyph has multiple Unicode values, only the first one is displayed.

6.3.16 Note

Notes are accessible from List View in the *Note* column. A glyph note can contain any text. Search for glyph notes with the search field in Font View (see p. 85).

6.3.17 Components

In List View, the components column shows a list of the components used in a glyph. The components cell in List View can be edited to add or remove components to or from a glyph. See section 6.1.2, ‘List View’ (p. 67) for hiding and showing columns.

6.3.18 Last Changed

The *Last Changed* column in List View is updated every time a glyph is modified. Add a ‘Write lastChange’ custom parameter in *File > Font Info... > Font* and disable it to prevent Glyphs from updating this property, which might be desirable when keeping the Glyphs file under version control.

6.4 NAMES & UNICODE

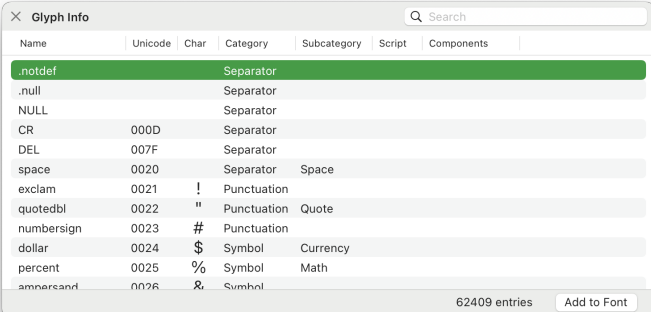
6.4.1 Glyph Info Database

Glyphs contains a glyph info database. For each glyph, the database defines:

- ▶ the nice glyph name that is used within Glyphs;
- ▶ the production name that is used on export;
- ▶ the associated Unicode value, when there is one;
- ▶ the components, in the case of composite glyphs;
- ▶ a default set of anchors;
- ▶ the associated marks displayed in the mark cloud (see p. 14);
- ▶ the script of the glyph;
- ▶ the category and subcategory of the glyph.

Glyphs uses two kinds of names: nice names and production names. Nice names are used within Glyphs and are intended to be readable for humans. Production names are used in exported font files and are intended to be read by computers. See section 6.3.1, ‘Glyph Name’ (p. 71) and section 6.3.8, ‘Production Name’ (p. 74) for details on both.

Glyphs uses its glyph info database to convert between nice names, production names, and Unicode values. This glyph info database can be viewed by choosing *Window > Glyph Info*. All glyphs stored in the database are displayed in a searchable table. Click a column header to sort the table by that column.



The screenshot shows the 'Glyph Info' window with a search bar and a table of glyph properties. The table has columns for Name, Unicode, Char, Category, Subcategory, Script, and Components. The first row is highlighted in green.

Name	Unicode	Char	Category	Subcategory	Script	Components
.notdef			Separator			
.null			Separator			
NULL			Separator			
CR	000D		Separator			
DEL	007F		Separator			
space	0020		Separator	Space		
exclam	0021	!	Punctuation			
quotedbl	0022	"	Punctuation	Quote		
numbersign	0023	#	Punctuation			
dollar	0024	\$	Symbol	Currency		
percent	0025	%	Symbol	Math		
ampersand	0026	&	Symbol			

62409 entries Add to Font

Use the methods described in section 6.3, ‘Glyph Properties’ (p. 71) to override the glyph properties of individual glyphs for a single Glyphs file. Adjust glyph properties for all Glyphs files by

creating a custom GlyphData.xml file.²

Suffixed glyphs inherit the glyphs info properties of their unsuffixed counterpart. For example, A.alt, A.ss01, and A.001 all use the same glyph properties as the unsuffixed glyph A. However, names like Aalt, Ass01, or A001 would not inherit any properties since they are missing the dot between the base name and the suffix. While such glyph names are valid, they require all properties to be set up manually. If possible, it is best to use glyph names from the glyph info database and use dot suffixes for glyph variants. See section 18.3, ‘Automatic Feature Generation’ (p. 242) for details on how specific dot suffixes can help generate OpenType features, automatically.

6.4.2 Naming Glyphs

Glyph names appear underneath their glyph cell in Grid View or their column in List View. Edit a glyph name by clicking it once. When adding a glyph by character (à) or by Unicode name (‘uni00E4’), Glyphs will automatically assign a nice name to the glyph (adieresis in this case).

Based on the glyph name, Glyphs can automatically assign Unicode values, a glyph category, subcategory, and generate OpenType features. See section 6.2.1, ‘Adding New Glyphs’ (p. 67) for more details.

This automated glyph renaming can be disabled for a font in *File > Font Info... > Other > Use Custom Naming*. Note that automatic feature code generation or automated assignment of categories does not work with custom names. Using a custom glyph info database (see p. 77) can provide automated behavior like assigning Unicode values or categories based on glyph names.

Unicode-value-based glyph names start with ‘uni’ followed by four hexadecimal digits or ‘u’ followed by five or six digits. See section 6.3.7, ‘Unicode’ (p. 73) for details.

6.4.3 Glyph Naming Rules

A font containing glyphs with invalid names cannot be exported. Ensure that every glyph name

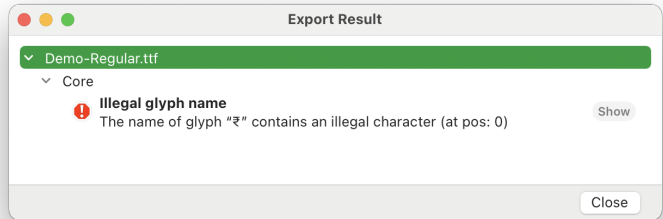
- ▶ only contains letters A–Z and a–z, numerals 0–9, underscore (‘_’), hyphen (‘-’) or period (‘.’);

² glyphsapp.com/learn/roll-your-own-glyph-data

- ▶ starts with a letter (except for non-exporting glyphs and the special `.notdef` glyph).

In particular, glyph names may not contain white space characters (space, tab, line break) or non-ASCII characters.

In this export results dialog, an error for the Glyph named ‘₹’ is shown. The glyph should instead use the name ‘rupeeIndian’.



Not adhering to these guidelines yields an error message at export time stating ‘Illegal glyph name’, followed by an explanation of where the invalid character is located in the glyph name. Click the *Show* button to open the invalid glyph in a new tab. Such error messages can also occur when trying to compile the OpenType feature code in *File > Font Info... > Features*.

A glyph whose name starts with an underscore is not exported by default. These names are used by component glyphs such as `_part.something`, `_smart.something`, `_cap.something`, or `_corner.something`. See section 10.1, ‘Components’ (p. 133) for details on component glyphs and their naming rules.

Hyphens are used to indicate the script a glyph belongs to. Most Latin and Greek glyphs have no hyphen suffix. This way, the glyph named `A` represents the Latin A, `A-cy` represents the Cyrillic A, and `Alpha` represents the Greek A. Arabic uses the suffix ‘-ar’, Hebrew uses ‘-hb’, Korean uses ‘-ko’, and Thai uses ‘-thai’. Open the Glyph Info window (see p. 77) for all available script suffixes. Special glyphs used in mathematics (‘-math’), Braille (‘-braille’), Fraktur (‘-fraktur’), or music (‘-musical’) also use this naming scheme. Some glyphs, like the Han glyphs used in Chinese, do not have a nice name and use the Unicode name instead.

Alternate glyphs such as small caps and initial/medial/final forms use a dot suffix: The name is the same as the regular glyph but followed by a period (.) and an identifier. For example, the small caps alternate of the glyph `x` is named `x.sc` and the initial form of `noon-ar` is `noon-ar.init`. Some suffixes are recognized by Glyphs and used to generate OpenType features automatically. See section 18.3, ‘Automatic Feature Generation’ (p. 242) for a list

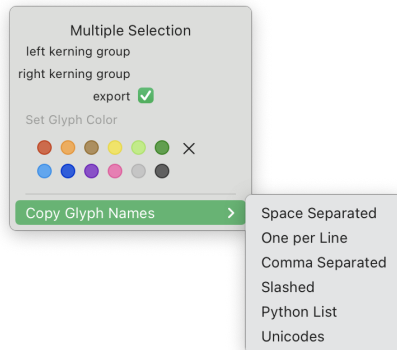
of all special glyph name suffixes.

Dot-suffixes can be chained. For the automatic feature generation to work, suffixes should be ordered in the same order as the features in *File > Font Info... > Features*. For example, `x.sc.ss01` is the name of the glyph `x` as a small cap with the first stylistic set enabled, since stylistic sets are typically ordered after small caps.

Ligature glyphs combine the names of their components with an underscore. For example, `f_f_l` is the `ffl` Ligature. Script and alternate suffixes can also be applied to ligature names. Suffixes must be added only at the end of the name and not for every component. For example, the ligature of `lam-ar` and `alif-ar` is named `lam_alif-ar`, and its final form is named `lam_alif-ar.fina`.

6.4.4 Copy Glyph Names

Control-click or right-click a glyph or a selection of multiple glyphs and choose from the *Copy Glyph Names* menu how to write the glyph names to the pasteboard.



The glyph names can be written in a variety of formats to the pasteboard:

Space Separated The names are formatted on a single line and joined by single space characters. Example: ‘one two three.’

One per Line Copies the glyph names with each name on its own line and no other separator characters.

Comma Separated Formats the glyph names on a single line with a comma and a space after each glyph name. Example: ‘one, two, three.’

Slashed Prefixes each glyph name with a slash ‘/’ and joins them

together on a single line. Example: `‘/one/two/three’`. This format can be pasted into Edit View.

Python List Formats the names as items of a Python list. Each name is enclosed in double quotes (") and separated by commas and line breaks. This format is useful when working with the Macro Panel or when writing scripts. Example:

```
"one",  
"two",  
"three",
```

Unicode Writes the Unicode values of the glyphs as hexadecimal numbers to the pasteboard, one per line. Values are padded to at least four digits with leading zeros. Glyphs without Unicode values are written as `‘- # someGlyph’`. If a glyph has multiple Unicode values, they are written on the same line and separated by a comma and a space. Example:

```
005B  
005D  
- # bracketleft.case  
- # bracketright.case  
27E8, 3008  
27E9, 3009
```

6.4.5 Renaming Glyphs

Click a glyph name to edit it. Changing a glyph name will also change some of its properties, such as the Unicode value or the category. If only the suffix after the period is changed, no properties will be modified since the base name stays the same. For example, renaming eacute to plus will change the glyph properties, while renaming eacute.sc to eacute.pc will not.

See section 6.5.3, ‘Batch-Renaming Glyphs’ (p. 83) for details on renaming multiple glyphs at once.

6.4.6 CID Mapping

CJK fonts use CID mapping, where glyphs are not accessed by glyph name but by a character identifier (CID). Glyphs can map nice glyph names to CIDs. A ROS (Registry, Ordering, Supplement) determines which glyph is assigned to which CID. Exporting glyphs that the ROS does not cover are added to the end of the CID mapping at export.

6.5 BATCH-PROCESSING

Use batch-processing operations to make changes to one or multiple glyphs at once.

6.5.1 Selecting Glyphs

Batch-processing commands operate on all selected glyphs. Select all glyphs currently shown in Font View by choosing *Edit > Select All* (Cmd-A). Choose *Edit > Deselect All* (Cmd-Opt-A) to cancel the selection. *Edit > Invert Selection* (Cmd-Opt-Shift-I) selects only the glyphs that are currently unselected. Inverting the selection is useful to select all glyphs, except for a few exceptions: Select the exceptions first and then invert the selection.

6.5.2 Batch Commands

The following commands from the *Glyph* and *Path* menus are applied to all selected glyphs. Commands with an asterisk (*) can be applied to all masters at once by holding down the Option key.

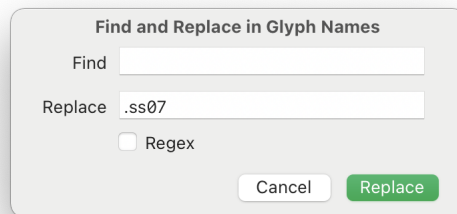
- ▶ *Glyph > Remove Glyph* (Cmd-Delete)
- ▶ *Glyph > Duplicate Glyph* (Cmd-D)
- ▶ *Glyph > Update Glyph Info*
- ▶ *Glyph > Update Metrics ** (Cmd-Ctrl-M)
- ▶ *Glyph > Transform Metrics* (see section 12.2.8, ‘Transform Metrics’, p. 184)
- ▶ *Glyph > Add Component ** (Cmd-Shift-C)
- ▶ *Glyph > Create Composite ** (Cmd-Ctrl-C)
- ▶ *Glyph > Decompose Components ** (Cmd-Shift-D)
- ▶ *Glyph > Add Image...* (see section 6.8, ‘Images’, p. 90)
- ▶ *Glyph > Set Anchors ** (Cmd-U)
- ▶ *Glyph > Reset Anchors ** (Cmd-Shift-U)
- ▶ *Path > Reverse Contours* (Cmd-Ctrl-Opt-R)
- ▶ *Path > Correct Path Direction ** (Cmd-Shift-R)
- ▶ *Path > Round Coordinates **
- ▶ *Path > Tidy up Paths ** (Cmd-Shift-T)
- ▶ *Path > Add Extremes*, or hold down Option for *Force Extremes* (see section 4.2.14, ‘Extremes & Inflections’, p. 32)
- ▶ *Path > Remove Overlap ** (Cmd-Shift-O)

- ▶ *Path > Transformations* (see section 12.2.8, ‘Transformations’, p. 185)
- ▶ *Path > Selection to Background* (Cmd-J), or hold down Option for *Add Selection to Background* (Cmd-Opt-J)
- ▶ *Path > Assign Background...*, or hold down Option for *Clear Background*.
- ▶ *Path > Swap with Background* (Cmd-Ctrl-J)
- ▶ *Path > Interpolate with Background* (see section 12.2.8, ‘Interpolate with Background’, p. 186)
- ▶ *Path > Other > Convert to Cubic*
- ▶ *Path > Other > Convert to Quadratic*

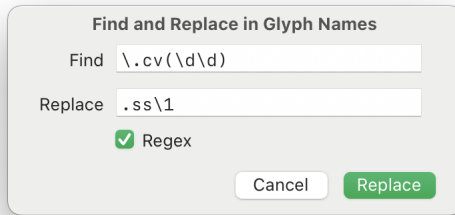
6.5.3 Batch-Renaming Glyphs

Rename glyphs by searching and replacing parts of glyph names. Select the glyphs for which parts of their name should be replaced, choose *Edit > Find > Find and Replace...* (Cmd-Shift-F), and enter the text to be replaced in the *Find* field. Then, enter the replacement in the *Replace* field and confirm by clicking *Replace*. The *Replace* text will replace all occurrences of the *Find* text in any of the selected glyph names. Leave the *Find* field empty to append the *Replace* text to all selected glyph names. Select the *Regex* checkbox to enable regular expression matching. Regular expressions can find and replace patterns of characters.

Add the suffix ‘.ss07’ to all selected glyphs by leaving the *Find* field empty and writing ‘.ss07’ into the *Replace* field:



Use the regex mode to replace patterns of glyph names. The dot ‘.’ has a special meaning in regular expressions and is thus preceded by a backslash ‘\’ in the *Find* field. ‘\d’ matches any digits from 0 to 9, and the parentheses around the ‘\d\d’ allow the two digits to be references in the *Replace* field by ‘\1’.



This would replace ‘someGlyph.cv03’ by ‘someGlyph.ss03’ or ‘otherglyph.cv15.alt’ by ‘otherglyph.ss15.alt’.

See section 18.1, ‘Regular Expressions’ (p. 240) for details.

6.5.4 Filters

Filters can be applied to all selected glyphs from Font View. See chapter 12, ‘Filters’ (p. 177) for details on how to apply filters.

6.5.5 Palette

Many of the controls in the Palette (Cmd-Opt-P) can be used from Font View. Both *Fit Curve* and *Transformations* operations can be applied to all selected glyphs. Palette sections provided by third-party plug-ins may also allow batch editing in Font View. Controls that do not operate on the entire glyph but only the selected nodes (such as *Fit Curve*) use the selection set in Edit View. See chapter 5, ‘Palette’ (p. 60) for details on the individual Palette sections.

6.5.6 Plug-ins & Scripts

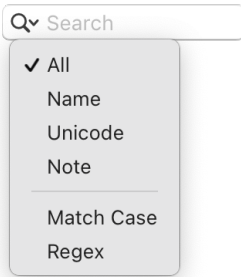
Some plug-ins and scripts can be applied to all selected glyphs at once. Refer to their documentation for guidance on batch editing.


6.6 FILTERING FONT VIEW

Font View can be filtered only to show a subset of glyphs. Filter glyphs using search, glyph categories, scripts, languages, and custom filter rules.

At the bottom-center of Font View, the number of selected, visible, and total glyphs is shown. The number of visible glyphs depends on the selected sidebar filter and the search query. Select *All* at the top of the left sidebar and clear the search field to show all glyphs of the font.

6.6.1 Search Field



In the top-left of Font View is the glyph search field (Cmd-F). Enter text to search by glyph name ('thorn'), Unicode value ('00FE'), Unicode character (p), or glyph note (see p. 76). Click the search glass  icon to search only by name, Unicode value, or note.






Select *Match Case* to search only for glyphs matching the capitalization of the search term. With this option selected, searching for 'Aring' would find Aring and Aringacute, but not aring or aringacute.

Select *Regex* to search using regular expressions. Regular expressions match patterns of characters.

- ▶ `'.'` matches any character and thereby finds all glyphs with a single character name
- ▶ `'..'` finds glyphs with two-character names
- ▶ `'.*\..sc'` finds glyphs ending in `'..sc'`
- ▶ `'[Aa].*-cy'` finds glyphs starting with 'A' or 'a' and ending in `'-cy'`


See section 18.1, 'Regular Expressions' (p. 240) for details.

6.6.2 Categories

Select a category filter in the left sidebar to show only the glyphs belonging to the selected category. Click the disclosure chevron  next to a category name to show its subcategories. Select a subcategory to show only those glyphs. The number of glyphs in a subcategory is shown next to the subcategory name. If Glyphs can infer that some glyphs in the subcategory might be missing, the count is replaced by a gray badge  showing both the current count  and the total count  of glyphs in the subcategory. Add the missing glyphs as described in section 6.2.1, 'Adding From the Sidebar' (p. 68). If all glyphs have been added, a checkmark  is displayed instead of a gray badge.

When multiple categories are selected by Command-clicking, glyphs from any of the selected categories are shown. If a *Categories* entry is selected together with a *Languages* or *Filters* entry, only glyphs that are part of both are displayed.



6.6.3 Languages

The *Languages* section contains script and language filters. Expand a language filter by clicking the disclosure chevron  next to the filter name. Glyph counts, badges, and checkmarks

appear the same as in categories (see above). Command-click script names to filter for multiple scripts at the same time, or Command-click other filters to combine them with the selected language filters.

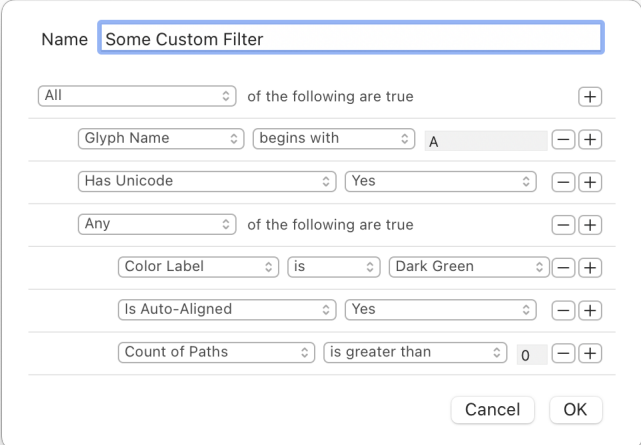
Only a few scripts are shown by default in the sidebar to keep the list short and tidy. Click the plus \oplus button next to the *Languages* label to edit the default list of scripts. A list opens where individual scripts can be shown or hidden by toggling the checkbox next to their name. A script entry is automatically added to the sidebar if the font contains glyphs of that script.

6.6.4 Smart Filters

Smart filters show glyphs that match a set of rules. They appear in the *Filters* section of the left sidebar and are identified by a gear  icon. Add a Smart Filter by clicking the actions menu  button located in the bottom-left of the font window and choosing *Add Smart Filter*. Edit an existing Smart Filter by double-clicking its gear icon, or choose *Edit Filter* from the actions menu for a selected filter.

Adding or editing a Smart Filter opens the Smart Filter editor window. It controls the name and the rules of a Smart Filter. The name of the Smart Filter will appear in the *Filters* sidebar. The rules define which glyphs to include in the filtered group.

When adding a new filter, only one rule is shown in the editor. Add additional rules by clicking a plus \oplus button, and remove a rule by clicking the minus \ominus button next to it.



The image shows a screenshot of the Smart Filter editor window. At the top, there is a text field labeled "Name" containing the text "Some Custom Filter". Below this, there are several rows of filter rules. Each row starts with a dropdown menu (currently set to "All" or "Any"), followed by a relationship dropdown (like "of the following are true", "begins with", "is", "is greater than"), and then a value field. Each rule row has minus and plus buttons on the right side. The rules shown are: "All of the following are true" with a sub-rule "Glyph Name begins with A"; "Has Unicode" set to "Yes"; "Any of the following are true" with sub-rules "Color Label is Dark Green" and "Is Auto-Aligned" set to "Yes"; and "Count of Paths is greater than 0". At the bottom right, there are "Cancel" and "OK" buttons.

The following filter rules are available:

Glyph Name The name (see p. 71) of the glyph.

Count of Paths The number of paths of the first master layer.

Count of Components The number of components (see p. 133) of the first master layer.

Tags The tags (see p. 73) of the glyph.

Script The script (see p. 75) to which the glyph belongs.

Category & Subcategory The category and subcategory (see p. 75) of the glyph.

Case Whether the glyph is uppercase, lowercase, small capital, or has no associated case.

Master Compatible Whether the masters can be interpolated (see p. 156).

Export Glyph Whether the glyph is included in exported font files (see p. 98).

Has Unicode Whether the glyph has at least one Unicode value (see p. 73).

Has Components Whether the layer of the selected master of the glyph contains components (see p. 133). Use *Contains Component* to check if a specific component is on the layer.

Has Hints Whether the layer of the selected master has any manually placed hints.

Has PostScript Hints Whether the layer of the selected master has any manually placed PostScript hints (see p. 195).

Has TrueType Hints Whether the layer of the selected master has any manually placed TrueType hints (see p. 204).

Has Corners Whether the layer of the selected master contains corner components (see p. 145).

Has Anchors Whether the layer of the selected master contains any anchors (see p. 35).

Contains Component Whether the layer of the selected master contains a specific component (see p. 133). Use *Has Components* to check whether there are any components on the layer.

Has Special Layers Whether the glyph has any special layers (see p. 62).

- Has Custom Glyph Info** Whether the glyph uses custom glyph info deviating from the glyph info database (see p. 77).
- Has Annotations** Whether the layer of the selected master contains annotations (see p. 52).
- Is Auto-Aligned** Whether the layer of the selected master is automatically aligned (see p. 137).
- Has Metrics Keys** Whether any of the glyph metrics use metrics keys (see p. 124).
- Metrics Keys Out of Sync** Whether any metrics keys (see p. 124) are out of sync and need to be updated.
- Has Kerning Groups** Whether the glyph uses kerning groups (see p. 128).
- Color Label** The color label (see p. 72) of the glyph. Compare with *Not Set* to filter for glyphs without a color label.
- Layer Color Label** The color label (see p. 72) of the layer of the selected master. Compare with *Not Set* to filter for layers without a color label.
- Is Hangul Key** Whether the glyph is a Hangul key, according to the ‘Hangul Composition Groups’ custom parameter.
- Custom** This filter rule can contain a custom glyph predicate expression. Such an expression is formulated like a glyph predicate token; see section 13.1.3, ‘Glyph Predicate Tokens’ (p. 189) for details.

Rules can also be nested, such that *all* of the rules in a group must apply, *at least one* of the rules in a group must apply, or *none* of the rules in a group may apply for a glyph to be included in the filter. Create a group of rules by holding down the Option key and clicking the dots (⋮) button. Then, pick *Any* (at least one nested rule must match), *All* (all nested rules must match), or *None* (no nested rule may match).

6.6.5 List Filters



Tip: In Font View, make a selection of glyphs and then choose *Copy Glyph Names > One per Line* to copy the glyph names in a format suitable for list filters. See also section 6.4.4, ‘Copy Glyph Names’ (p. 80).



A list filter shows all glyphs that are specified in a list of glyph names. List filters appear in the *Filters* section in the left sidebar with a list ☰ icon. Add a list filter by clicking the actions menu (⋮) button located in the bottom-left of the font window and choosing *Add List Filter*. By default, all selected glyphs are added to the list filter. Glyph names are separated by spaces or line breaks. Edit an existing list filter by double-clicking its list icon or


choosing *Edit Filter* from the actions menu.

Change the name of a filter with the *Name* text field at the top of the list field editor window. Click *OK* to confirm, or click *Cancel* to discard any changes to the list filter. Select a list filter from the sidebar to see all glyphs of the font that are in the list. Glyphs are sorted in the order of the list. See section 6.2.1, ‘Adding From the Sidebar’ (p. 68) for details on adding glyphs from a list filter.

6.6.6 Managing Filters

Reorder filters by dragging them up and down in the filters list. Choose *Add Folder* from the actions menu  to create a filter folder . Click and drag to move a filter into a folder. Folders can also be put into other folders. Hold down the Command key to select multiple filters, or hold down Shift to select a range of filters. Font View will show only glyphs that are in all of the selected filters. Select a folder to show all glyphs belonging to any of the contained filters.

Quickly edit a Smart Filter or list filter by double-clicking its gear  or list  icon. Double-click a filter name to rename it without opening the editor window. Or, select a filter and press Return to rename it.

Delete a filter by choosing *Remove Filter* from the actions menu . This also works for filter folders. Note that removing a folder will also remove all filters inside it.

6.6.7 Custom Categories & Languages

Custom entries in the *Categories* and *Languages* sections can be added by creating a *Groups.plist* file in the Glyphs Info folder. See *Custom Sidebar Entries in Font View*³ for details.

6.7 GLYPH ORDER

Categories, languages, and filters do not influence the glyphs order in the exported font. Instead, glyphs are ordered by their sort name (see p. 76). Define a custom order for all glyphs of the font with the ‘glyphOrder’ custom parameter in *File > Font Info... > Font* (Cmd-I). The parameter takes a list of glyph names, one name per line. The order of glyphs is used both in Font View and in exported font files.

When opening an OTF or TTF font file, Glyphs preserves the glyph order of the file by automatically adding a ‘glyphOrder’

³ glyphsapp.com/learn/custom-sidebar-entries-in-font-view

custom parameter when *Keep Glyph Names from Imported Files* is selected. See section 3.3, ‘User Settings’ (p. 15) for details.

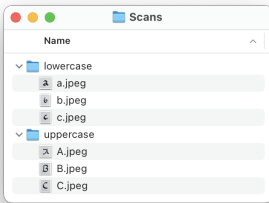
6.8 IMAGES

Add an image by dragging it from Finder onto a glyph cell. The image will be added to the currently active layer. Alternatively, choose *Glyph > Add Image...* to insert an image file to the selected glyph.

Add images to multiple glyphs at once by choosing *Glyph > Add Image...* and then selecting multiple image files. The images will be placed on the layer of the current master of the glyphs that correspond to an image name. For example, *Thorn.png* is added to the glyph *Thorn*, and *Djecy.jpeg* is added to *Dje-cy*.

Note that while glyph names are case-sensitive (*A* is different from *a*), file names on macOS are case-insensitive by default (*A.png* is considered the same file name as *a.png*). Uppercase and lowercase files thus cannot coexist in the same folder. As a workaround, place images for uppercase letters in a different folder than lowercase letters.

Images are shown in Font View when *View > Show Image* is selected. If a glyph is empty (no paths or components are placed in the glyph), the image is displayed regardless of the *Show Image* option. See section 4.12, ‘Images’ (p. 54) for information on working with images in Edit View.



7 Font Info

The Font Info window contains general information about a Glyphs file, such as the font family name, the masters, the exported font instances, and the OpenType features. Open Font Info by choosing *File > Font Info...* (Cmd-I) or click the Info **i** button located in the top-left of the main window.

Font Info is split into multiple tabs. The *Font*, *Masters*, and *Exports* tabs control the font metadata with rows of fields. Click the plus **+** button to the right of the bold headings to add additional fields. Click the minus **—** button to the right of a row to delete it. For rows with a plus button to the right, hold down the Option key to reveal the minus button. Alternatively, click a row to select it, then delete it by pressing Cmd-Delete.

Family Name	Handbook		+
Localized Family Names	Default	Handbook	+
	German	Handbuch	—
	French	Manuel	—
	Italian	Manuale	—
	Chinese (Simplified)	手册	—

Some fields can be localized into multiple languages. These display a language menu and a plus **+** button to the right of the field. Click the plus button to add additional localizations and pick the respective language from the languages menu.

Select one or multiple rows by Command-clicking and copy them by choosing *Edit > Copy* (Cmd-C). Paste rows with *Edit > Paste* (Cmd-V).

7.1 FONT

The *Font* tab contains information that applies to all exported font files. Click the plus **+** button to the right of the *General* heading to add additional fields.

7.1.1 Family Name

The ‘Family Name’ is the name given to the exported fonts. This name will appear in font menus. Fonts with the same family name will be grouped into the same styles submenu. A family name should only contain ASCII letters (a-z and A-Z), digits (0-9), and spaces. Any other characters may prevent the fonts from

exporting or installing.

Add the ‘Localized Family Names’ field to provide a family name containing non-ASCII characters or to localize the family name into multiple languages. Click the plus \oplus button to the right of the *General* heading, select ‘Localized Family Names’, and click *Add*.

Name IDs refer to the entries of the OpenType Naming Table stored in OTF and TTF fonts. See the OpenType name table specification¹ for a list of all possible name table entries.

Glyphs uses the family name to derive the file name and the entries for the Name IDs 1, 3, 4, and 6. In PostScript/CFF-based OpenType fonts, the family name is also used for the *FontName* and *FullName* in the CFF table. Add the ‘Font Name’ and ‘Full Name’ fields in *File > Font Info... > Font > General* to control those values independently of the family name.

The font name can be overwritten by instances using the ‘Localized Family Names’ field.

7.1.2 Units per Em

The ‘Units per Em’ (UPM) determines the number of coordinate units in the em square of each glyph. Increasing the UPM can improve the representation of fine details, as it increases the placement options of the nodes. 1000 UPM is the default for new Glyphs files. The OpenType specification allows values between 16 (2^4) and 16,384 (2^{14}), but values above 5000 can lead to problems in Adobe InDesign and Illustrator. Some applications have issues with values above 3000 UPM.

Additionally, the coordinates of points (nodes and handles) may not exceed $\pm 32,768$, and glyph widths in PostScript/CFF-flavor fonts can be problematic beyond 4096 UPM (2^{12}). Thus, if the design of the font requires higher precision, it may be better to change the *Grid Spacing* and *Subdivision* values. See section 7.5.1, ‘Grid Spacing & Subdivision’ (p. 107) for more details.

Click the double-arrow $\left\langle \right\rangle$ button next to the text field to scale the entire font to a different UPM. Scaling changes the UPM and glyph outlines together, keeping the apparent size of the glyph outlines the same. Enlarge all glyph outlines of the font by setting the UPM to a smaller value without scaling (for example, from 1000 to 800) and then scale $\left\langle \right\rangle$ the UPM back to its original value (for example, 1000). Change the UPM to a smaller value and scale back to the original UPM for smaller outlines.

Consider using the ‘Scale to UPM’ custom parameter on an

1 docs.microsoft.com/typography/opentype/spec/name

instance to change the UPM value for exported files, scaling glyph outlines and metrics to fit the new UPM value. Use the ‘Units per Em’ custom parameter to change the UPM without scaling outlines or metrics.

7.1.3 Version


The ‘Version’ field is split into a major and minor version, separated by a period. The version of a new Glyphs file is ‘1.000’. Click the stepper buttons to change the minor version.

See the OpenType head specification² for more details.


The major and minor versions of the font are written as the ‘fontRevision’ entry in the Header table (head) of the font. Additionally, Glyphs derives the Version String (Name ID 5) from the *Version* entry by appending its own version to the font version before writing the version string. For example, ‘Version 1.000;Glyphs 3.2 (3245)’.

Use the ‘Version String’ parameter to define a custom version string. Note that some applications require this version string to begin with ‘Version’ followed by the major version, a period, and the minor version. Consult the OpenType specification for details.

7.1.4 Creation Date

The ‘Creation Date’ field is used for the ‘created’ and ‘modified’ dates in the head table. Update its value to the current date and time by clicking the update  button.

7.1.5 Axes

The *Axes* define design axes for interpolation and variable fonts. Click the plus  button to the right of the *Axes* heading to add a new axis. An axis has a name, a four-character tag, and can be hidden or not.

Glyphs offers a range of predefined axis names. Selecting one of the predefined names will change the tag. Select the *Hidden* checkbox to hide the axis from font users.

See chapter 11, ‘Interpolation’ (p. 156) for details on working with axes across multiple masters, and specifically section 11.2, ‘Setting up Axes’ (p. 158) for details on the usage of axes in variable fonts.

² docs.microsoft.com/typography/opentype/spec/head

7.1.6 Custom Parameters

Additional font configurations can be specified using custom parameters. Click the plus $+$ button to the right of the *Custom Parameters* heading to add a new parameter. A list of custom parameters is shown. Use the search or scroll the list to find the desired parameter. Click a parameter name in the list to get a description, and click *Add* to add the selected parameters to the font.

7.2 MASTERS

A master is a set of specifically designed and configured glyph outlines and metrics. From the master setup in a Glyphs file, font instances are calculated and exported as OpenType font files. See section 7.3, ‘Exports’ (p. 98) for details.

7.2.1 Managing Masters

When a new Glyphs file is created, the font has one *Regular* master. At the bottom of the *Masters* tab are buttons for adding and deleting masters. Click the plus $+$ button to add a new master using one of the following options:

Add Master adds a new, empty master. All glyph layers are empty, and the metrics are set to their default values.

Add Other Font shows a list with all masters of the fonts currently open in Glyphs. Select one or more masters and import them by clicking *OK*. Click *Cancel* to not import any masters.

Duplicate Selected duplicates the masters that are currently selected in the *Masters* tab. The new masters will have the same name, glyph layers, and metrics as the selected masters. Consider renaming the duplicate masters to distinguish them from the original masters.

Click the minus $-$ button located in the bottom-left of the Font Info window to delete the selected masters.

Reorder masters by dragging them to the desired position in the masters list. The first master in the list has special relevance for glyph-level hinting information (unless the ‘Get Hints From Master’ parameter is set). See section 14.3, ‘Manual hinting’ (p. 200) and chapter 15, ‘TrueType Hinting’ (p. 204).

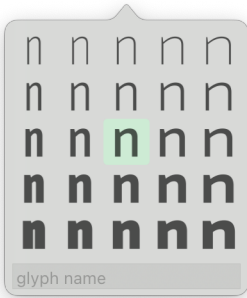
Master properties such as the name, metrics, and custom parameters can also be edited for several selected masters at

once. Properties that do not share a common value are indicated with a gray *Multiple Values* text.

7.2.2 General

In *File > Font Info... > Masters > General*, define the name of a master and its icon. The master name is only used internally by Glyphs and not by exported font files. Use instances (see p. 98) to define the exported font files.

The master icon can be a glyph from the font or one of 25 predefined lowercase n glyphs in various widths or weights. Click the master icon to choose a predefined image, or write a glyph name into the text field at the bottom of the icon picker to use that glyph as the icon.



7.2.3 Axes Coordinates

The *Axes Coordinates* indicate the position of a master in the designspace. The designspace is defined by the axes in the *Font* tab (see p. 93). For example, a font with two axes—width and weight—has a two-dimensional designspace. A master can then be placed anywhere within that designspace. See section 11.3, ‘Setting up Masters’ (p. 158) for details on placing masters in a designspace.

7.2.4 Metrics & Alignment Zones

Master metrics define the baseline of the font, offsets from that baseline, and other values such as the angle of slope of italic glyphs. In Edit View, the metrics are displayed as orange alignment lines (when *View > Show Metrics* is selected). The default metrics are set according to the script and language selection. For example, Latin fonts have the following default vertical metrics:

Ascender The height reached by the ascenders of letters such as b, d, f, h, k, l, p, or β.

Cap Height The height of capital letters such as H, T, or W without overshoot.

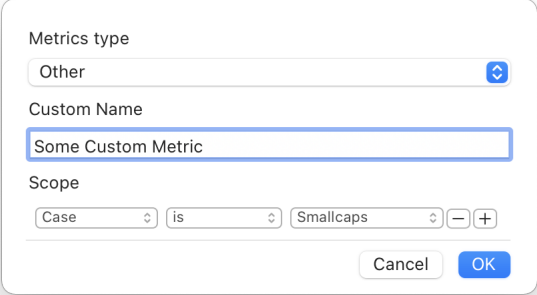
x-Height The height of short lowercase letters such as x, a, c, e, n without overshoot.

Baseline The position on which letters are placed, always 0.

Descender The depth reached by descending letters such as Q, g, j, p, or q. This value is negative since it is below the baseline.

Italic Angle The angle of slope of the italic glyphs stems. This value affects several elements, including the sidebearings calculation, the anchors alignment between selected nodes, and transformations that take the italic angle into account. The value is in degrees clockwise.

Other scripts use different default metrics. For example, an Arabic font has the default metrics of *Ascender*, *Baseline*, *Descender*, *Alef Height*, *Joining Line*, and *Meem Depth*.



Metrics type
Other

Custom Name
Some Custom Metric

Scope
Case is Smallcaps - +

Cancel OK

Click the name of a metric to edit its category, name, and scope. The category is the metric type, and the name can be chosen freely to identify the metric. The scope limits the metric to a subset of glyphs. For example, scoping a metric to glyphs with a *Devanagari* script hides its alignment line in all other glyphs. See section 6.6.4, ‘Smart Filters’ (p. 86) for details on glyph scopes.

800	16	Ascender
700	16	Cap Height
500	16	x-Height
0	-16	Baseline
-200	-16	Descender
0°		Italic Angle
550	16	Some Custom Metric Smallcaps

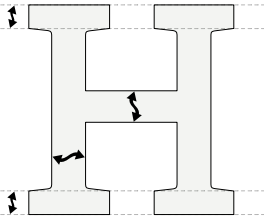
The left number field of a metric defines the offset from the baseline. (The *Italic Angle* metric has one field only.) The right number field is the size of the alignment zone and starts at the offset of the metric. Nodes and anchors placed in alignment zones are highlighted. See section 4.2.5, ‘Nodes in Alignment Zones’ (p. 25) and section 4.4.1, ‘Adding, Editing, & Removing Anchors’ (p. 35).

Alignment zones are used by the automatic hinting process, which ensures that overshoots snap to even metric lines at small font sizes or low screen resolutions. See section 14.1.2, ‘Alignment Zones’ (p. 197) for details on PostScript hinting with alignment zones. For TrueType hinting, separate zones may be configured (see section 15.2.1, ‘TrueType Zones’, p. 205).

7.2.5 Stems

Font designs typically conform to standard stems. For example, the capital I has a vertical stem, and the capital H has two vertical stems and one horizontal stem. A rounded shape like the letter O has left and right vertical stems and top and bottom horizontal stems. Serifs and crossbars, such as in the lowercase f and t, are also stems.

Frequently used stem widths are defined in *File > Font Info... > Masters > Stems*. The autohinter uses these stem values to add hints to vertical and horizontal stems. See section 14.2, ‘Autohinting’ (p. 199) for details on hinting with the autohinter. For TrueType hinting, separate stems may be configured (see p. 208).



Click the plus $+$ button located to the right of the *Stems* heading to add a new stem definition. A stem can either be vertical \rightarrow (measured left to right) or horizontal \downarrow (measured top to bottom). The name of a stem is used to describe it and for interpolation: When interpolating between masters, stems with the same name in both masters are used to interpolate the in-between stem value. The stem value should be as close to as many master stems as possible. See chapter 14, ‘PostScript Hinting’ (p. 195) for more information on good stem values.

Click the actions \odot button located on the right of a stem entry line to restrict it to a specific subset of glyphs. This scoping works the same as in master metrics (see p. 95) or Smart Filters (see p. 86).

7.2.6 Custom Parameters

Masters can have custom parameters, just like the *Font* tab (see section 7.1.6, ‘Custom Parameters’, p. 94). A custom parameter value is interpolated if it is defined in all masters. Some custom parameters exist on the font and the master level. A parameter on a master will overwrite such a font parameter. Instances can also have custom parameters, which overwrite both font and

master parameters.

7.2.7 Number Values

Tokens can use Number Values in OpenType feature code to insert numbers that interpolate between masters. Click the plus \oplus button located to the right of the *Number Values* heading to add a new number value. See section 13.1.1, ‘Number Value Tokens’ (p. 188) for more information.

7.3 EXPORTS

The *Exports* tab contains the static instances and variable fonts that are created when exporting with *File > Export...* (Cmd-E). Click the plus \oplus button located in the bottom-left of the *Exports* tab to add a new static or variable instance. Four options are available:

- ▶ *Add Instance* adds a new instance named ‘Regular’.
- ▶ *Add Instance for Each Master* adds instances at the designspace locations of each master. For example, if the font has two masters, *Thin* with a weight of 100 and *Bold* with a weight of 700, a *Thin* and a *Bold* instance will be added with the same axis values as the respective masters.
- ▶ *Instance as Master* adds a new master to the *Masters* tab using the outlines and metrics of the currently selected instance. For example, consider a font with two masters—*Thin* and *Bold*—and five instances—*Thin*, *Light*, *Regular*, *Semibold*, and *Bold*—where the *Light*, *Regular*, and *Semibold* instances are interpolations between the two masters. Select the *Regular* instance and choose *Instance as Master* to add a *Regular* master with the outlines and metrics taken from the instance. Now the *Light* instance interpolates between the *Thin* and *Regular* masters, the *Regular* instance uses the *Regular* master without interpolation, and the *Semibold* instance interpolates between the *Regular* and *Bold* masters.
- ▶ *Add Variable Font Setting* adds a variable font instance. See section 8.1.2, ‘Exporting Variable Fonts’ (p. 112) for more information.

Click the minus \ominus button to delete the selected instances. Select multiple instances by Command-clicking entries in the sidebar, or Shift-click to select a range of instances. Reorder instances by dragging them to a new position. The order of

instances is just used for organization within Glyphs and does not affect the exported font files. For more information on interpolating instances, see chapter 11, ‘Interpolation’ (p. 156).

7.3.1 Active

The *Active* setting controls whether an instance is exported with *File > Export...* (Cmd-E). Deactivated instances are shown with a gray icon in the sidebar.

7.3.2 Style Name


The style name appears in the font menu of an application. Examples include ‘Regular’, ‘Bold’, or ‘Light Italic Display’. The style name can include the letters A–Z and a–z, the digits 0–9, spaces, and some punctuation marks such as the hyphen. Characters outside the ASCII range are not compatible with all software. Instead, add the ‘Localized Style Names’ custom parameter for style names with Unicode characters.

Some operating systems, applications, and printers do not work with fonts with long font names. A conservative limit is 20 characters for the overall name (family and style name). When the font name is too long, consider adding the ‘Font Name’ and ‘Full Name’ custom parameters with abbreviated font names.

See the Naming tutorial³ for more information on good style names and how to shorten them for maximum software compatibility.

7.3.3 Weight & Width

The *Weight Class* and *Width Class* values are used to categorize an instance. Some applications use the weight and width classes to order fonts in a font menu. When defining the typography of a website with CSS, the weight class is used to access fonts of different weights.

Click the disclosure  button to select from a range of predefined values. These values follow the OpenType OS/2 specification:⁴ 100, 200, ..., 900 for standard weight classes and 1–9 for width classes. The weight class can be any value from 1 to 1000 if a font family needs more than the nine standard weight classes. Click a weight class field to manually enter a value ranging from 1 to 1000 (for example, 350 for a *Semi Light* style).

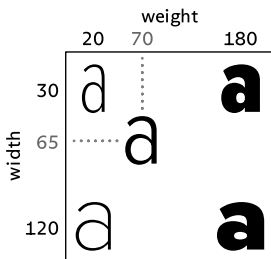
³ glyphsapp.com/learn/naming

⁴ docs.microsoft.com/typography/opentype/spec/os2

Note that some applications only work with fonts that use one of the predefined weight classes and reject fonts with in-between values.

Weight and width *classes* are not necessarily the same as the weight and width *axis coordinates* (if a font has such axes). The weight and width classes are used by applications other than Glyphs to order and categorize fonts. Axes are used internally in Glyphs for interpolation (though they can also be exposed, see section 11.3.1, ‘Axes Coordinates’, p. 159). Weight and width axes are not bound to the same value ranges as the weight and width classes.

7.3.4 Axes Coordinates



Axes coordinates define the location of an instance in the designspace. The axes of a font are defined in the *Font* tab (see section 7.1.5, ‘Axes’, p. 93), and masters define the outlines and metrics at specific locations in the designspace of those axes (see section 7.2.3, ‘Axes Coordinates’, p. 95).

An instance with axis coordinates between masters is *interpolated* and *extrapolated* when it is outside the designspace. For example, consider a single weight axis with two masters, *Light* at 40 and *Bold* at 120. An instance with a weight of 80 is interpolated. Instances with weights of 20 and 150 are both extrapolated. Extrapolated instances are often challenging to get right. Consider drawing masters at the axes extremes so that all instances are interpolated.

7.3.5 Style Linking

Use style linking to connect a bold, italic, or bold-italic instance to its regular instance. Style linking is used by applications like word processors to switch to the correct font when the user clicks the **Bold** button or presses Cmd-I to italicize the selected text.

Style Linking has two checkboxes—one for bold and one for italic—and a text field for the name of the base instance. For example, in a *Bold Italic* instance, select both checkboxes and enter the name of the normal instance (‘Regular’) into the text field.

Style linking only works between fonts with the same family name. By default, Glyphs uses the family name from the *Font* tab for all instances in the *Exports* tab. The family name can be overwritten for individual instances by adding a ‘Localized Family

Names' in *File > Font Info... > Exports > General*. Fonts generated from different Glyphs files can use the same family name. This is common when using separate Glyphs files for the Roman and the Italic, which still belong to the same family.

Some applications do not expose a complete list of all styles in a font. Instead, they only allow the user to select a family and its regular, bold, italic, and bold-italic styles using **Bold** and *Italic* buttons. In these applications, fonts that are not linked typically appear as separate entries in a font family picker. Consider using the following style linking strategy:

- ▶ The *Bold*, *Italic*, and *Bold Italic* instances of a font family should always be linked to the *Regular* instance.
- ▶ Other italic styles should always be linked to their non-italic counterpart. For example, *Medium Italic* is marked as the italic of *Medium*.

Some font designers also link bolds with other styles. For example, *Semibold* is marked as the bold of *Light*. This hides the *Semibold* style from the font family picker, making it only available when pressing the **Bold** button while the *Light* style is selected. Therefore, users may be unaware that the font family includes a *Semibold* style.

7.3.6 Custom Parameters

Instances can have custom parameters. Some custom parameters exist in the *Font*, *Masters*, and *Exports* tabs. A parameter in the *Exports* tab will overwrite values set in the *Masters* and *Font* tabs. For example, a trial version of an instance might add the word 'TRIAL' to the end of the family name.

7.4 FEATURES

The *Features* tab manages OpenType feature code that substitutes and positions glyphs. These features are described in the OpenType specification for the GSUB table⁵ (glyph substitution) and the GPOS table⁶ (glyph positioning).

Some GPOS features such as kerning, mark positioning, and cursive attachment are primarily defined in Edit View by editing the kerning values between glyphs and placing anchors. These implicit features do not appear in the *Feature* tab. See

⁵ docs.microsoft.com/typography/opentype/spec/gsub

⁶ docs.microsoft.com/typography/opentype/spec/gpos

section 7.4.5, ‘Implicit Features’ (p. 105) for details.

7.4.1 Editing Feature Code

OpenType features are defined using the AFDKO (Adobe Font Development Kit for OpenType) syntax. Glyphs also offers additional feature code functionality (see p. 188).

The *Features* sidebar is divided into three parts: *Prefix*, *Classes*, and *Features*. Click the plus \oplus button located to the right of each heading to add an entry. Select one or multiple sidebar entries and delete them by clicking the minus \ominus button located in the bottom-left of the window.

- ▶ *Prefix* entries contain arbitrary feature code, primarily used to define lookups that need to be placed outside any specific feature.
- ▶ *Classes* are lists of glyph names. The names are separated by a white space character, either a space or a line break.
- ▶ *Features* control the glyph substitution and positioning rules. A feature is identified by its four-character tag. Click the plus \oplus button next to the *Feature* heading and search by feature tag or feature name. Double-click a feature to add it, or navigate the picker list with Up and Down arrow keys and press Return when the desired feature is selected.

A *Spec* button is shown in the top-right of the window when a feature is selected. Click it to open the feature specification on the Microsoft OpenType developer website.


7.4.2 Automatic Feature Code

Glyphs can automatically generate feature code for many commonly used OpenType features. Click the \mathcal{C} *Update* button located in the bottom-left of the window to create and update automatically generated prefixes, classes, and features all at once.

Glyphs also looks for new classes and features to generate when updating. For example, suppose glyphs with an underscore in their name (such as `f_l`) were added since the last update. In that case, Glyphs spots those additions and automatically adds a ‘liga’ (Standard Ligatures) feature with the corresponding feature code to the sidebar.

Many automatic features use the dot suffix in glyph names. For example, glyphs ending in ‘.sc’ are put in the small caps feature (**sub a by a.sc**), or glyphs ending in ‘.tf’ are put into the

tabular figures feature (`sub zero by zero.tf`). This also works by appending the feature tag to the end of a glyph name. For example, the glyph `g.ss01` would substitute the glyph `g` when the first stylistic set (`'ss01'`) is enabled.

Automatic prefixes, classes, and features are indicated with an update  button to the right of their name. Click this button to update only that single entry. Select an entry from the sidebar and deselect the *Generate Feature Automatically* checkbox above the code editor to edit the code manually. Manual code is not updated by clicking the *Update* button. See section 7.4.3, ‘Manual Feature Code’ (p. 103) for details.

Some features and classes are not added automatically when pressing *Update*, even though Glyphs could auto-generate them. For example, the class `All`, which contains every exporting glyph, can be automatically generated, but not every font project needs it. The *Capital Spacing* feature (`'csp'`), which adds a bit of extra space between capital letters, can also be automatically generated but is not required by all designs, and some designers might want to write it manually.

Some features automatically generated by Glyphs do not appear in the *Features* tab. See section 7.4.5, ‘Implicit Features’ (p. 105) for details.

7.4.3 Manual Feature Code

Uncheck the *Generate Feature Automatically* checkbox located above the feature code text field to write the code manually. Code that cannot be generated automatically, such as the `'calt'` feature, is always in manual mode.

Feature code is written in the AFDKO syntax, as specified by the OpenType Feature File Specification.⁷ Glyphs additionally supports extended feature code syntax (see p. 188).

After editing feature code, click the *Compile* button located in the bottom-left of the window to preview it in Edit View. See section 4.13.3, ‘Previewing OpenType Features’ (p. 55) for details. Note that Edit View only previews manual substitution code, not manual positioning code.

Glyphs syntax highlights the feature code in the editor: Keywords (like `sub` or `lookup`) are formatted differently from glyph names (like `ka-deva` or `zero`) which are again formatted differently from numeric values, lookup names, and comments.

⁷ adobe-type-tools.github.io/afdko/OpenTypeFeatureFileSpecification.html



Trigger the autocomplete function by typing a glyph name, a class name, or a lookup name (for example, a glyph name after the keyword **sub** or a lookup name after the keyword **Lookup**). Glyphs will provide a menu of matching suggestions.

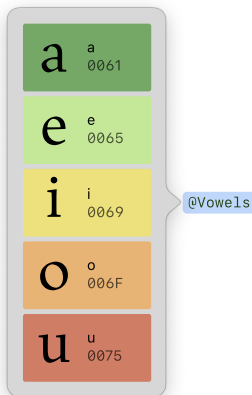
```
sub halant-deva by rak|
```

```
rakar-deva  
rakar_uMatra-deva  
rakar_uuMatra-deva  
rakar-deva.small
```

Navigate to the desired suggestion with the Up and Down arrow keys. If there are many matching suggestions, scroll until the desired suggestion is shown, or narrow down the suggestions list by typing a few more characters. Accept the highlighted suggestion by pressing the Return key.


Option-click a glyph or class name to show a visual preview. For glyph classes, a list of all member glyphs is shown.

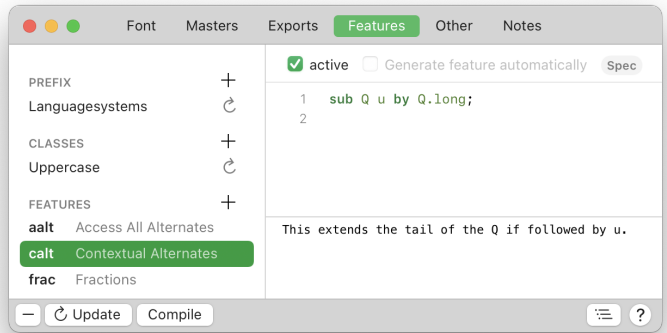
Glyphs shows error messages inline if the feature code is incorrect and cannot be compiled. Click the  error icon for an expanded description. Errors with a wrench  icon include a suggested fix. Click the *Fix* button to accept the suggestion.




```
sub @T adiaeresis' by adieresis.compact;
```

```
❗ Use of undefined class name '@T'  
🔧 Unknown glyph 'adiaeresis'  
Did you mean 'adieresis'?  
Replace 'adiaeresis' with 'adieresis' 
```

Prefixes, classes, and features with errors show an  icon in the sidebar. Click the error icon in the sidebar to jump directly to the erroneous part of the code.





The text field below the feature code can be used to write notes about a prefix, class, or feature. Change the height of the notes field by dragging the gray line above it up and down.

Click the snippets () button to insert predefined code snippets. Choose any of the snippets to insert them into the feature code. See section 18.2, ‘Custom Feature Code Snippets’ (p. 241) for details on adding custom feature code snippets.

7.4.4 Naming Stylistic Sets

Stylistic sets—the features with tags ‘ss01’ through ‘ss20’—can be given proper names. Applications supporting feature names will show those names instead of ‘Stylistic Set 1’, ..., ‘Stylistic Set 20’.

When a stylistic set is selected in the sidebar, rows for the feature name are shown above the code editor. Each row has a language picker and a text field for a localized name in that language. Click the plus  button located to the right of a row to add additional localizations of a feature name. Click the minus  button to remove a row.

Use the *Default* language to provide a fallback name if none of the other localized names match the user’s locale. Many systems assume the default language to be English. For example, to provide the name of the ‘ss07’ feature in English and German, use the *Default* language for the English feature name and the *German* language for the German name.

7.4.5 Implicit Features

Implicit features are automatically generated by Glyphs but do not appear in the *Features* tab. The kerning feature (‘kern’) is such an example. Kerning values are defined in Edit View and

managed by the Kerning window (see section 9.2, ‘Kerning’, p. 126). The ‘kern’ feature code is different for each master and instance, which is why it is not exposed in the *Features* tab and instead added silently in the background on export. The same is true for other features that are added implicitly:

Distance ‘dist’ works similarly to kerning, but, unlike kerning, it is not controllable by the user.

Mark to base ‘mark’ is generated if base glyphs have base anchors such as `top` or `bottom` and mark glyphs have matching mark anchors such as `_top` or `_bottom`. This also applies to the ‘abvm’ (above-base mark positioning) and ‘blwm’ (below-base mark positioning) features. See section 4.4.2, ‘Mark to Base Positioning’ (p. 36) for details.

Mark to mark ‘mkmk’ is generated similarly to the ‘mark’ feature. See section 4.4.3, ‘Mark to Mark Positioning’ (p. 36).

Cursive positioning ‘curs’ is also defined in Edit View by placing special `exit` and `entry` anchors. See section 4.4.4, ‘Cursive Attachment’ (p. 36).

Implicit features can also be extended manually. Click the plus `+` button next to the *Features* heading in the sidebar and search for a feature tag, for example, ‘kern’. Double-click the result or press Return to insert the feature.

In the feature code editor, insert the line `# Automatic Code` somewhere in the feature code. Any code above this line will be placed *before* the automatically generated feature code, and any code below it will be placed *after* the automatic feature code:

```
pos a b -20;
```

```
# Automatic Code
```

```
pos x y 15;
```

```
pos y z -5;
```

Insert the automatic code placeholder by clicking the snippets `☰` button located in the bottom-right of the window.

7.4.6 Export-Specific Features

Use the following custom parameters to customize the features for individual instances.

- ▶ ‘Add Prefix’, ‘Remove Prefixes’, and ‘Replace Prefix’

- ▶ ‘Add Class’, ‘Remove Classes’, and ‘Replace Class’
- ▶ ‘Add Feature’, ‘Remove Features’, and ‘Replace Feature’

Automatic classes and features do not need to be changed for different instances; they are adapted automatically for each instance.

7.5 OTHER SETTINGS

7.5.1 Grid Spacing & Subdivision

The points defining the outlines are placed on a grid, and the *Grid Spacing* defines how the coordinates of the points are rounded. The default value of a grid spacing of 1 rounds points to font units. For example, (3.142, 7.816) is rounded to (3, 7). A value of 5 rounds points to the nearest factor of 5; consequently (3.142, 7.816) is now rounded to (5, 10). Finally, a value of 0 does not round points and maintains (3.142, 7.816) as it is. Disabling rounding can be helpful for highly detailed designs such as ornaments or when glyphs are scaled to a new size. Higher values are useful in coarse designs such as pixel fonts.

Contrary to popular belief, decimal coordinates can be exported into CFF-flavor (PostScript) fonts.

All tools and modifications snap to the grid. Choose *Path > Round Coordinates* to round all selected points to the grid.

Use a *Subdivision* value above 1 to offer finer control while keeping a large grid spacing for the design coordinates. For example, a *Grid Spacing* of 100 and a *Subdivision* of 5 places points on a 20 unit subgrid. A default *Grid Spacing* of 1 with a *Subdivision* of 10 gives point coordinates one decimal point, such as (3.1, 7.8).

7.5.2 Keyboard Increments

Many places in Glyphs allow modifying values with the arrow keys. For example, points can be moved in Edit View, or numeric values in text fields can be incremented and decremented using the Up and Down arrow keys. Values are modified by 10 units when holding down the Shift key and by 100 units when holding down the Command key. Adjust the Shift and Command values as desired.

7.5.3 Use Custom Naming

Select the *Use Custom Naming* option to use custom glyph names that do not conform to the names from the built-in glyph database. Enable this option for workflows using custom glyph

names, thus preventing the automatic replacement of names.

Deactivating the *Use Custom Naming* option will not immediately rename the glyphs of the font. Instead, choose *Glyph > Update Glyph Info* to update the properties, including the glyph name, of all selected glyphs. Note that it may invalidate imported or manually written feature code, which will require manual code adjustments.

When importing OpenType fonts or UFO sources, glyph names are either kept or modified to conform with the built-in glyph database. Go to *Settings > User Settings* to change the setting of the option *Keep Glyph Names from Imported Files* (see section 3.3, ‘User Settings’, p. 15).

7.5.4 Disable Automatic Alignment

This option disables the automatic alignment of components and the automatic synchronization of metrics for composite glyphs such as diacritics. Lock the position of individual components by choosing *Lock Component* from the context menu in Edit View. Go to *Glyphs > Settings... > User Settings > Disable Automatic Alignment for Imported Files* for the setting of imported files (see section 3.3, ‘User Settings’, p. 15). See also section 10.1.8, ‘Automatic Alignment’ (p. 137).

7.5.5 Keep Alternates Next to Base Glyph

Select *Keep Alternates Next to Base Glyphs* to sort glyph variants with a dot suffix name directly after the base glyph without the dot suffix. For example, a.ss15, a.alt, and a.loclDEU will follow the a glyph instead of being placed after all base lowercase letters.

7.5.6 File Format Version

The format of a Glyphs file changed slightly from Glyphs 2 to Glyphs 3. For compatibility, Glyphs 3 can read and write the old Glyphs 2 file format.

When working on a Glyphs file created in Glyphs 2, Glyphs 3 will import it as a version 2 file. Some of the features new in Glyphs 3 cannot be stored in version 2 format files. Use all Glyphs features by changing this setting to *Version 3*. New Glyphs files are always created in version 3 format.

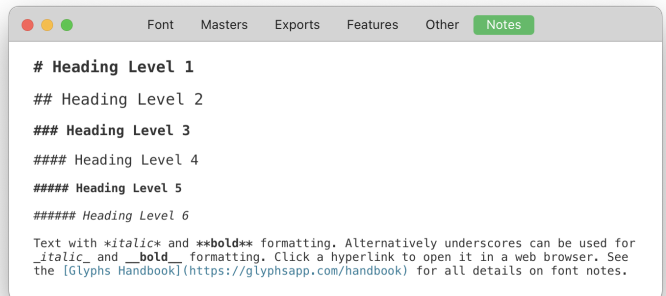
7.5.7 Font Type

The font type controls the type of interpolation used for static instances. *Default* interpolation is based on Adobe’s Multiple Master model. This is the interpolation that Glyphs exclusively used before variable fonts, and it continues to be the default for all new documents. *Variable* interpolation mimics the interpolation used by variable fonts. Using this interpolation for static instances might be helpful in cases where both static and variable fonts are exported and the differences between the two should be kept to a minimum.

7.6 NOTES

The *Notes* tab in Font View stores a text note. It is not exported and is used only for working in Glyphs. The font note corresponds to the ‘note’ custom parameter of the font.

The text field uses syntax highlighting for Markdown. Italic (**italic**, `_italic_`), bold (****bold****, `__bold__`), hyperlinks ([Link Text](https://example.org) & `<https://example.org>`), and heading (`#`, `##`, `###`, `####`, `#####`, `#####`) formatting are supported.



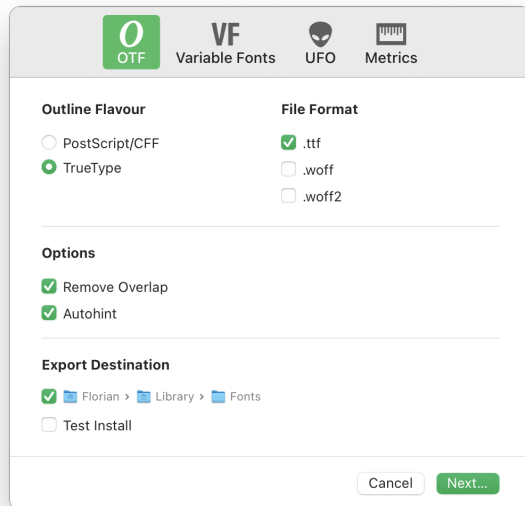
8 Import & Export

8.1 EXPORTING FONT FILES

Choose *File > Export...* (Cmd-E) to show the export dialog for the currently open Glyphs file. Hold down the Option key and choose *Export All* (Cmd-Opt-E) to export all open Glyphs files without showing an export dialog. Exported files overwrite existing files.

The export dialog offers different export formats. These are described in the following sections.

8.1.1 OpenType Export



The *OTF* tab exports static OpenType fonts. This export option uses the font instances (see p. 161) or, if none are configured, a default instance based on the first master.

Outline Flavor

There are two flavors of OpenType fonts. Their main difference is the types of glyph outlines that they contain.

- **PostScript/CFF** flavor files containing a CFF (Compact Font Format) table storing glyph outlines as cubic Bézier curves. Font files with this flavor typically end in '.otf'.

- ▶ **TrueType** flavor files containing a glyph table storing glyph outlines as quadratic Bézier curves and other tables related to the TrueType format. Font files with this flavor typically end in ‘.ttf’.

Pick one of the two outline flavors. Paths drawn in Glyphs are cubic, but Glyphs can also edit quadratic paths. When exporting to TrueType, Glyphs creates quadratic paths from the cubic paths stored in the Glyphs file. This might introduce minuscule differences between the drawn and the exported glyph outlines.

Besides the paths, the two flavors differ in their hinting capabilities; see chapter 14, ‘PostScript Hinting’ (p. 195) and chapter 15, ‘TrueType Hinting’ (p. 204) for details. Exporting to TrueType keeps components without converting them to outlines; see section 10.1, ‘Components’ (p. 133) for details.

File Format

The first option (*.otf* or *.ttf*) is generally used to install a font in an operating system and use the font in graphic design and word-processing applications. WOFF (*.woff*) and WOFF2 (*.woff2*) are intended for usage on the web (Web Open Font Format). Check all formats that should be exported.

Options

The *Remove Overlap* option unifies all paths and components in glyph layers as not to include overlapping paths. This is the same operation as *Path > Remove Overlap*; see section 12.2.10, ‘Remove Overlap’ (p. 187) for details. Release versions of fonts should have this option checked; uncheck it only during development for faster exports or if the *Remove Overlap* filter is already applied using custom parameters.

Check the *Autohint* option to apply automatic hinting. If the *PostScript/CFF* flavor is selected, autohinting is applied to all glyphs that do not contain manual PostScript hints; see section 14.2, ‘Autohinting’ (p. 199). For the *TrueType* flavor, autohinting is applied to the entire font, and any manually added TT hints will be ignored; see section 15.1, ‘Autohinting’ (p. 204).

Export Destination

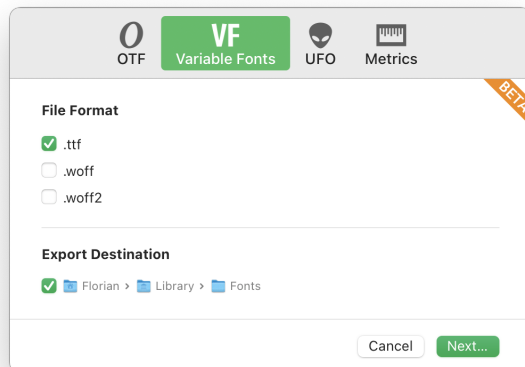
The export dialog offers two possible export destinations.

- ▶ Select the checkbox next to the file path to export to the chosen folder. Choose a different folder by clicking the file path and picking a folder from the file browser dialog. Common destinations are the system font library¹ and the Adobe fonts folder.²
- ▶ Select the *Test Install* checkbox to install the fonts directly on the Mac without exporting any font files. This writes the font data directly into the Mac font system. The exported fonts will be available in apps using the system font framework Core Text (such as Pages, Final Cut, and Pixelmator, but not Adobe or Affinity apps). Test installed fonts bypass many font caching issues, but they need to be re-installed after a reboot.

Click the *Next...* button to export to the selected destination. If neither export destination is checked, Glyphs will prompt for an export folder.

The export destination can be customized per instance in *File > Font Info... > Exports* by adding an ‘Export Folder’ custom parameter. The instance will be exported to that this folder relative to the selected export destination. The export folder may contain slashes (‘/’) to export to a nested folder structure (such as ‘Trial Versions/Webfonts’). Place the export folder outside the export destination by going up folder levels with two dots (‘..’, such as ‘../Trial Versions’).

8.1.2 Exporting Variable Fonts



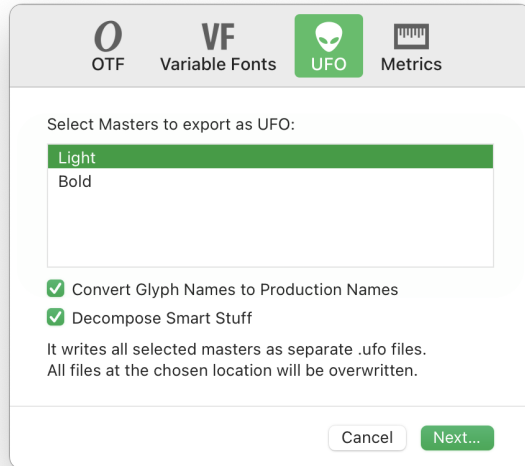
1 [\(Username\)](#) > Library > Fonts

2 [\(Username\)](#) > Library > Application Support > Adobe > Fonts

The *Variable Fonts* tab exports variable OpenType fonts. This export option uses the variable font settings (see p. 161) or, if none are configured, a default variable font setting.

Set the desired file formats and export destination like in the *OTF* tab. Variable fonts are always exported in the TrueType flavor.

8.1.3 Exporting UFO



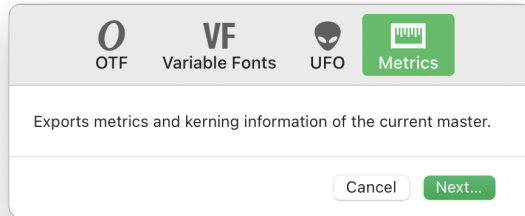
The *UFO* tab exports the selected masters as UFO (Unified Font Object) files. Select one or multiple masters to be exported to UFO by Command-clicking entries in the masters list. Select a range of masters by Shift-clicking or all by choosing *Edit > Select All* (Cmd-A).

Check *Convert Glyph Names to Production Names* to use the glyph production names instead of the nice names used inside Glyphs. This option is helpful when working on the exported UFO files with tools expecting a simplified naming scheme compared to the more expressive names used by Glyphs. See section 6.3.1, ‘Glyph Name’ (p. 71) for more on the difference between nice names and production names.

Decompose Smart Stuff handles Glyphs-specific features such as smart components or corner components to regular paths. The UFO format does not support these smart components. Check this option to convert unsupported Glyphs features to regular paths or leave it unchecked to remove them.

Click *Next...* to choose an export destination. UFO files are exported in the UFO version 3 format. See section 8.2.3, ‘Unified Font Object’ (p. 115) for details on working with UFO files.

8.1.4 Exporting Metrics



The *Metrics* tab exports the font metrics of the current master. The current master is the master that is currently selected in the toolbar (see p. 55).

Click *Next...* to choose the export destination and the export format. There are two metrics formats to choose from:

- ▶ **Metrics File** is a proprietary Glyphs metrics format containing all spacing and kerning information of the master.
- ▶ **AFM File** (Adobe Font Metrics) is an old-style format that is compatible with other font editors. However, it cannot contain all types of metrics information. For instance, AFM does not support group kerning or metrics keys.

See section 8.4.2, ‘Importing Metrics’ (p. 118) for details on importing these metrics files into a Glyphs file.

8.2 SOURCE FORMATS

A Glyphs document can be stored in one of three source file formats. Plug-ins can add support for additional source formats; see section 17.3, ‘Plug-ins’ (p. 237).

Pick the source file format when saving with *File > Save As...* (Cmd-Shift-S). Each format is described in the following sections.

8.2.1 Glyphs File

Pro Tip: .glyphs files are stored as NeXTSTEP-flavor property lists. They can be opened and edited in a text editor for advanced editing and inspection.

The *Glyphs File* format is the default format in which new Glyphs files are created. Glyphs files end in the ‘.glyphs’ file suffix. A Glyphs file is *flat*, meaning it is stored as a single file containing all information about the font.

When opening a Glyphs files created in Glyphs 2, Glyphs 3 will

keep the file in a Glyphs 2 compatible mode. This way, the file can still be opened in Glyphs 2 after editing it in Glyphs 3. However, not all features of Glyphs 3 can be stored in a version 2 file. In *File > Font Info... > Other > File Saving*, change the *File format version* to *Version 3*. New Glyphs files are created in the *Version 3* format. See also section 7.5.6, ‘File Format Version’ (p. 108).

8.2.2 Glyphs File Package

The *Glyphs File Package* format is identical to the *Glyphs File* format, except it splits parts of the font into separate files and groups those files into a folder with the ‘.glyphspackage’ suffix. Notably, every glyph is put into its own file.

In Finder, a .glyphspackage folder appears as a normal file that can be opened by double-clicking it like any other file. Inspect its inner contents by Control-clicking or right-clicking the package file in Finder and choosing *Show Package Contents* from the context menu.

A normal Glyphs file requires re-saving the entire file, even if only a single glyph is changed. Since a package contains a file per glyph, only the files that changed are modified when saving. While this difference is inconsequential for smaller fonts, the Glyphs package format is recommended for projects with many glyphs (such as CJK fonts).

Splitting every glyph into its own file also helps when managing the document with a version control system (VCS) such as Git. The frequently changing *display strings* representing the glyphs of the Edit View tabs are stored in a `UIState.plist` file inside the package. This makes it easy to ignore user interface changes in the VCS.

8.2.3 Unified Font Object

The *UFO* format is available both as a source file format and as an export format (see p. 113). When using UFO as a source format, not all features are available in Glyphs. Activating an unsupported feature will trigger a warning message.

Glyphs supports opening and saving UFO version 2 and version 3 formats. The version is maintained when opening a UFO file. New UFO files created using the UFO export are stored in version 3 format.

UFO files end in the ‘.ufo’ suffix. Like Glyphs packages, they are folders appearing as a normal file in Finder. Unlike Glyphs

files and Glyphs packages, a UFO stores only a single master.

UFO files are compatible with many other font development tools. Use UFO as a source format when working on a font together with others that use UFO-compatible software. However, when using UFO for a single-purpose tool (for example, a specialized kerning tool), consider working with a Glyphs file or Glyphs package and exporting to UFO. The results of the specialized tool can then be imported into the Glyphs file, for instance, by importing metrics (see p. 118).

See section 8.3.1, ‘Font File Importing Behaviors’ (p. 116) for file import configuration options. Configure these options for a specific UFO file in *File > Font Info... > Other*; see section 7.5.3, ‘Use Custom Naming’ (p. 107) and section 7.5.4, ‘Disable Automatic Alignment’ (p. 108).

8.3 OPENING FONT FILES

Prefer working with sources (‘.glyphs’, ‘.glyphspackage’, ‘.ufo’) as described in section 8.2, ‘Source Formats’ (p. 114).

Glyphs can open compiled font files of the following formats:

- ▶ OpenType, PostScript/CFF flavor (‘.otf’)
- ▶ OpenType, TrueType flavor (‘.ttf’)
- ▶ OpenType Collection, TrueType flavor (‘.ttc’)
- ▶ Adobe Type 1, PostScript Font Binary (‘.pfb’)

Glyphs cannot reverse-engineer all the information inside a compiled font file. Opening such a file and exporting it again will produce a font file different from the original. For example, some hinting information and OpenType tables are lost when importing a compiled font. Compiled fonts are opened in a view-only mode. Choose *File > Save As...* (Cmd-Shift-S) to save it to one of the source formats (see p. 114).

The order of the glyphs is written into a ‘glyphOrder’ custom parameter in *File > Font Info... > Font*. Plug-ins can add support for opening additional font formats; see section 17.3, ‘Plug-ins’ (p. 237).


8.3.1 Font File Importing Behaviors

When opening a font file, Glyphs will try to apply its built-in naming scheme and sync the metrics of composite glyphs with their base glyphs. These behaviors can be disabled in the settings; see section 3.3, ‘User Settings’ (p. 15) for *Keep Glyph Names from Imported Files* and *Disable Automatic Alignment for Imported Files*.

8.3.2 Opening TrueType Font

TrueType flavor OpenType fonts are imported, maintaining their quadratic Bézier curves. Glyphs can edit, but not create quadratic curves. Convert them to PostScript/CFF flavor cubic curves with *Path > Other > Convert to Cubic*. This command applies to all selected paths, layers, or glyphs. Components contained in TrueType fonts are kept.



8.3.3 Importing Multiple Fonts Files into a Glyphs File

Import fonts as masters by choosing *File > Font Info... > Masters >  > Add Other Font*. See section 7.2.1, ‘Managing Masters’ (p. 94) for details. If the newly added masters are interpolated, ensure they are compatible (see section 11.5, ‘Outline Compatibility’, p. 162).

8.3.4 Importing OpenType Features

Some OpenType features cannot be reconstructed when opening a compiled OpenType file. The features in *File > Font Info... > Features* show a list of features that should be imported. Lookups used from multiple features are placed in a prefix named ‘Prefix’. Kerning and kerning groups are mostly preserved, but contextual kerning is not.

8.3.5 Importing PostScript Hints

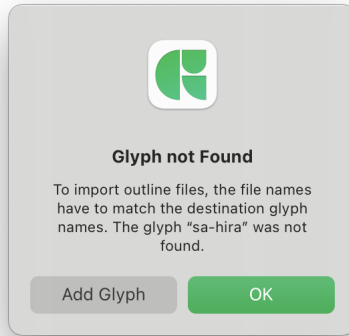
Most glyph-level PostScript hints are preserved. Alignment zones and standard stems are also preserved. However, most glyph-level hints are not connected to nodes anymore. In a Multiple Masters setup, drag the blue circle  and triangle  onto outline nodes that fit the hint. See section 14.3.1, ‘Stem Hints’ (p. 201) for details.

8.4 IMPORTING FONT DATA

8.4.1 Importing Outlines

File > Import > Outlines... imports PDF, EPS, or SVG files as outlines. In Font View, multiple files can be selected and imported into the glyphs with a matching name (for example, A.pdf into A and comma.svg into comma). If no glyph exists for the file, Glyphs prompts to create a matching glyph or ignore the file.

Click *OK* to skip the file, or *Add Glyph* to add a new glyph matching the filename to the font and import the outlines into it.



In Edit View, *File > Import > Outlines...* imports a single file into the current glyph.

Vector paths stored in the file are imported as glyph outlines. Only outlines are kept; the fill color and other styles are discarded. For PDF files, the stroke thickness is imported as a stroke style (see p. 34).

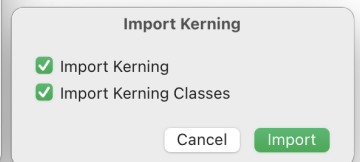
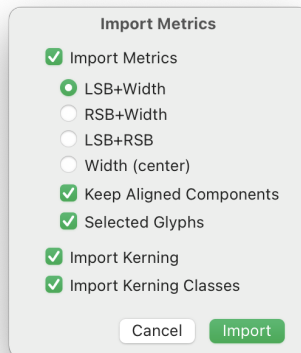
See section 8.5, ‘Vector Drawing Applications’ (p. 119) for information on importing outlines with copy and paste.

8.4.2 Importing Metrics

Glyphs can import the spacing and kerning of a master from *Metrics* and *AFM* files (both of which can be exported from Glyphs, see section 8.1.4, ‘Exporting Metrics’, p. 114) and from Glyphs and UFO files.

In Font View, choose *File > Import > Metrics...* to import metrics into the currently open Glyphs file. Select a file and click *Import*, then an import dialog will appear:

Depending on the selected file, one of the two import dialogs on the right is shown. Left: Metrics or AFM file. Right: Glyphs or UFO file.



Select whether to import kerning values and kerning classes with the respective checkboxes. When importing metrics from a Glyphs file, kerning values are not supported. Import kerning values from one Glyphs file into another by copying and pasting them in the Kerning window (*Window* → *Kerning*, Cmd-Opt-K, see section 9.2.6, ‘Kerning Window’, p. 129 for details).

Importing from a Metrics for AFM file offers controls for glyph metrics in addition to kerning. Check *Import Metrics* to choose from one of four import modes:

- ▶ import left sidebearing and width, adjust the right sidebearing;
- ▶ import right sidebearing and width, adjust the left sidebearing;
- ▶ import both sidebearings, adjust the width;
- ▶ import the width and center the outline (equal sidebearings).

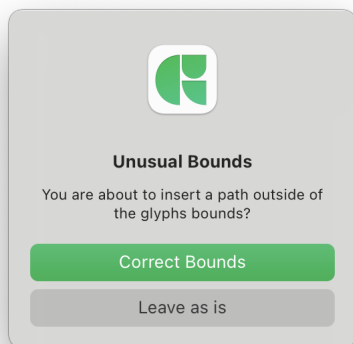
Automatic alignment is disabled for components where the metric values deviate from those derived from automatic alignment, unless *Keep Aligned Components* is checked. Metrics are imported for all glyphs by default. Check *Selected Glyphs* to import metrics only for the selected glyphs or all glyphs if none are selected.

8.4.3 Importing Feature Files

File > *Import* > *Features...* imports OpenType features written in the AFDKO feature language. Select a file ending in ‘.fea’ and click *Import* to add the contained features to *File* > *Font Info...* > *Features*. See section 7.4, ‘Features’ (p. 101) for details on OpenType features.

8.5 VECTOR DRAWING APPLICATIONS

Glyphs supports pasting outlines from most vector drawing applications. The pasted outline may be placed far outside the glyph box. Glyphs detects such situations and offers to correct the bounds of the pasted outline:



The following sections describe recommended setups for specific applications.

8.5.1 Adobe Illustrator

Tip: Quickly get the correct scale by drawing a rectangle with the height of the capital letters in Glyphs, copy and paste it into an Illustrator artboard and scale the drawings to fit the height of the rectangle.

Vector outlines can be imported from Illustrator with copy and paste. For best results, configure the Illustrator artboard such that one point corresponds to one font unit. A font with 1000 UPM would map to an Illustrator artboard 1000 pt high.

Copy closed paths in Illustrator and paste them with *Edit > Paste* (Cmd-V) in Glyphs onto the current glyph. Node coordinates are rounded unless the grid spacing (see p. 107) of the font is zero.

8.5.2 Affinity Designer

Affinity Designer places the copied outlines at the origin (0, 0), regardless of where the outline is placed in the Designer artboard. Glyphs supports both the standard outline copying mode and the SVG mode (*Affinity Designer > Settings... > General > Copy items as SVG*).

8.5.3 Sketch

In Sketch, set the contour to a 1 pt outline without fill. This will prevent double outlines in Glyphs.

8.6 FILE FORMAT INTEROPERABILITY

The UFO format is generally used when exchanging files with other font editors. See section 8.2.3, 'Unified Font Object' (p. 115) for details.

FontLab 7 imports and exports Glyphs files natively. This will preserve more details than exchanging UFO files with FontLab. For FontLab Studio 5, use the *Glyphs Import*³ and *Glyphs Export*⁴ macros. Install them in FontLab Studio 5 by opening Finder and choosing *Go > Go to Folder*. Enter the path of the FontLab Studio 5 macros folder⁵ and press Return. Place the two macro files there and relaunch FontLab Studio 5. The macros will be available in the macro toolbar.

Note that not all Glyphs features are supported in other font editors. Exchanging files may thus result in simplified data (for example, decomposed outlines) or missing font information. Consider keeping a copy of the original Glyphs file for safety.

8.7 PROJECTS

Projects are collections of instance definitions linked to a Glyphs font file but kept in a separate project file. This is useful for setting up different font versions without altering or compromising the original Glyphs file. Project files carry a ‘glyphsproject’ suffix.

8.7.1 Setting up a Project

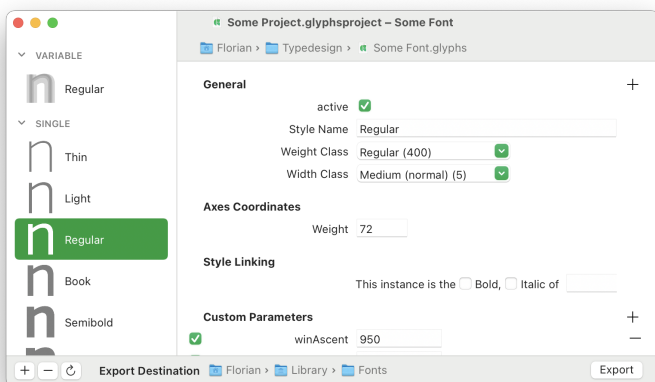
Create a new Glyphs project with *File > New Project*. Save the newly created project with *File > Save* (Cmd-S).

Click the *Choose* button located in the top-right of the project window, select a Glyphs file, and confirm with *Choose*. The selected font does not need to be open in Glyphs. This *links* the Glyphs file to the project. Click the file path to choose a different Glyphs file. The instances of the font are listed in the sidebar of the project window:

³ github.com/schriftgestalt/Glyphs-Scripts/blob/master/Glyphs%20Import.py

⁴ github.com/schriftgestalt/Glyphs-Scripts/blob/master/Glyphs%20Export.py

⁵ ~/Library/Application Support/FontLab/Studio 5/Macros/



Rearrange instances using drag and drop. Duplicate an instance by Option-dragging it to its new location in the list. Alternatively, copy (Cmd-C) and paste (Cmd-V) to duplicate the selected instances. Click the plus (+) button to add new instances and variable font settings. Delete the selected instances with the minus (-) button. Revert to the instances of the Glyphs file with the arrow (↺) button.

Edit the parameters of instances in the right half of the project window. See section 7.3, ‘Exports’ (p. 98) for details.

8.7.2 Exporting a Project

Ensure that the project is saved (*File > Save*, Cmd-S) and that the path to the linked font is valid (a path becomes invalid when the linked file is renamed, moved, or deleted). Click the *Export Destination* file path to set the export folder. Instances are exported into this folder. Customize the export folder of an instance with the ‘Export Folder’ custom parameter (see section 8.1.1, ‘Export Destination’, p. 111). Click the *Export* button located in the bottom-right of the project window to export all configured instances. The linked Glyphs file does not need to be open for export.

9 Spacing & Kerning


The space between glyphs is controlled by their spacing and kerning. *Spacing* defines the general white space surrounding a glyph, while *kerning* makes adjustments to specific glyph pairs.

9.1 SPACING

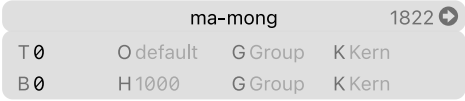
Spacing is the process of adjusting the sidebearings of glyphs to achieve an even rhythm in a line of text. There are no fixed rules for how large or small the sidebearings of a glyph should be. The amount of spacing depends on the style of the font and the script of the glyphs. In general, glyphs that share similar shapes share similar sidebearing values. For example, the left sidebearing of the Latin K is usually the same as the left sidebearing of the Latin H, while the right sidebearing of the Latin D and the Latin O should be the same or similar.

When the Text tool **T** is active and *View > Show Metrics* (Cmd-Shift-M) is checked, the sidebearing values and widths of all glyphs are displayed:

9.1.1 Info Box

The Info box (*View > Show Info*, Cmd-Shift-I) shows the sidebearings of the current glyph. In text mode, the current glyph is placed after the text cursor. For left to right and right to left scripts, the sidebearing values are shown to the left and right of the  metrics icon in the center of the Info box:

For a top to bottom script, the Info box shows the top and bottom sidebearings next to the T and B:



ma-mong		1822
T 0	O default	G Group K Kern
B 0	H 1000	G Group K Kern

Use the Up and Down arrow keys to increase and decrease a value in the Info box. Hold down Shift for steps of 10 and both Shift and Command for steps of 100.

Editing sidebearings in the Info box can be handy for quick adjustments. When editing the spacing of many glyphs, consider using the spacing shortcuts.

9.1.2 Spacing Shortcuts

Mnemonic: the Control key is located on the left, and the Command key is on the right.
Control: left sidebearing.
Command: right sidebearing.

Spacing shortcuts can be used with the Text tool to change the sidebearings of the current glyph. Hold down the Control key to change the left sidebearing (LSB) with the Left Arrow and Right Arrow keys (^◀ and ^▶). Hold down the Command key to change the right sidebearing (RSB) with the Left Arrow and Right Arrow keys (⌘◀ and ⌘▶). Hold down Shift for steps of 10.

Ctrl-Left	increase spacing on left side
Ctrl-Shift-Left	increase spacing on left side by 10 units
Ctrl-Right	decrease spacing on left side
Ctrl-Shift-Right	decrease spacing on left side by 10 units
Cmd-Right	increase spacing on right side
Cmd-Shift-Right	increase spacing on right side by 10 units
Cmd-Left	decrease spacing on right side
Cmd-Shift-Left	decrease spacing on right side by 10 units

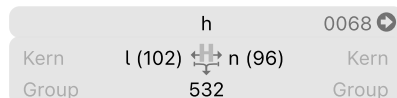
When both the Control and Command keys are held down, the LSB and RSB are changed simultaneously, shifting the glyph inside its width. This is particularly useful in monospaced designs, where the width of the glyph cannot be changed.


Cmd-Ctrl-Left	shift glyph outline left
Cmd-Ctrl-Shift-Left	shift glyph outline left by 10 units
Cmd-Ctrl-Right	shift glyph outline right
Cmd-Ctrl-Shift-Right	shift glyph outline right by 10 units

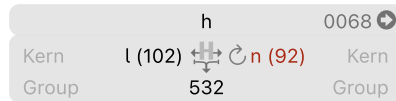
The Control and arrow key shortcuts (^◀ and ^▶) conflict with the default system shortcuts for Mission Control. If they do not work in Glyphs, disable or change them in *System Settings > Keyboard > Shortcuts > Mission Control*.


9.1.3 Metrics Keys


Metrics keys can be used instead of numeric metrics values to link the metrics—the LSB, RSB, and width—of one glyph to those of another. Write the name of a glyph in a sidebearing or width field in the Info box to use the metric value of that glyph. For example, the LSB of an h could be linked to the l and its RSB to the n:



Linked metrics are *not synced automatically* when the metrics of the linked glyphs are changed. Instead, in Edit View, out-of-sync metrics keys are displayed in red next to an update  button:



Sync a metric key by clicking its update  button. Fix all out-of-sync metrics on the current layer by choosing *Glyph > Update Metrics* (Cmd-Ctrl-M). Hold down the Option key to update the metrics on all masters (Cmd-Ctrl-Opt-M). These commands also work when multiple glyphs are selected.

In Font View, out-of-sync metrics are indicated by a warning  triangle in the top-right corner of the glyph cells. Consider using a Smart Filter (see p. 86) to reveal all of the glyphs with out-of-sync metrics.

Metrics Key Calculations The sidebearing fields can contain simple calculations following these rules: a calculation starts with an equal sign (=) and a glyph name, followed by a mathematical operator, + (plus, addition), - (minus, subtraction), * (asterisk, multiply), or / (forward slash, divide), and ends with a number. For example, ‘=n+10’ is the metric value of the n glyphs plus 10, and ‘=comma/2’ indicates half the metric value of the comma glyph. Decimal numbers can also be used: ‘=o*1.2’ or ‘=koKai-thai/1.5’. The result of a calculation is rounded to the nearest font unit.

Constant Metrics Keys Equating with a number such as ‘=50’ fixes a metric to that value. This is helpful if the metric should keep its value even when the outlines are changed. A calculation that only contains a glyph name (such as ‘=n’) has the same effect as writing just the glyph name (‘n’).

Mirrored Metrics Keys Place a pipe character (|) directly after the equals sign to reference the opposite sidebearing. For example, writing ‘=|n’ in the right sidebearing of the u uses the value of the left sidebearing of the n. Keep a glyph centered by writing ‘=|’ into one of its sidebearing fields. Such a value will reflect the opposite sidebearing of the same glyph, thus centering the glyph outlines inside its bounding box.

Local Metrics Keys By default, a metrics key applies to all masters of the font. Prefix a metrics key with two equals signs ('==') instead to specify a local metrics key. Local metrics keys do not affect other masters and can be used as exceptions. Examples: '==m' or '==hyphen+15'.

Metrics Keys at Baseline Offsets Add an at-sign (@) and a number at the end of a metrics key to measure a sidebearing at a specific offset from the baseline. For example, 'A@0' is the sidebearing of A on the baseline, and 'A@200' is the sidebearing value of A as measured 200 units above the baseline.

9.1.4 Metrics Keys & Automatic Alignment



The metrics fields in the Info box of an automatically aligned glyph (see p. 137) show the text 'auto' and the derived values in parentheses. Modify an automatic metric while keeping it automatically aligned by entering an equals sign, a plus or minus sign, and the offset value that should be added or removed. For example, '+25' or '-10'.

Modifying automatic metrics can be useful for glyphs where a mark reaches far outside the glyph box. Typical examples are idieresis (ï), dcaron (ď), and lcaron (ĺ).

9.2 KERNING

Carefully spaced glyphs should fit together well in words and sentences. Some glyph combinations, however, need specific adjustments to look good. Usually, glyphs with a lot of white space (such as A and V) need to be moved closer together. In contrast, glyphs that collide (such as f and ?) need to be moved further apart. Kerning is the adjustment of the distance between glyphs.

Note that left to right kerning is stored separately from right to left kerning. When working with a font that includes glyphs of both directions, make sure to select the correct writing direction before kerning. See section 4.9.2, 'Writing Direction' (p. 46) for details.

With the Text tool active, kerning between glyphs is highlighted in light blue  for negative kerning and yellow  for positive kerning. These colors can be changed in the Appearance settings (see p. 14).

9.2.1 Kerning Modes

Edit View has three kerning modes: no kerning **To**, kerning **To***, and locked kerning **To^l**. Cycle between the modes by clicking the icon located in the bottom-right of the Edit View canvas.


No kerning **To** disables the preview of kerning in Edit View. The kerning shortcuts (see p. 127) are also disabled in this mode. Kerning **To*** previews kerning pairs in Edit View.

Locked kerning **To^l** works like the normal kerning mode but disables the spacing shortcuts (see p. 124), preventing accidental spacing changes while kerning.

9.2.2 Info Box

In Edit View, the Info box (*View > Show Info*, Cmd-Shift-I) displays the kerning between the current glyph and the immediate glyph on each side. When no kerning is set, the gray placeholder ‘Kern’ is visible.

Click a kern field, enter an adjustment value, and confirm by pressing Return or by exiting the kern field. Clear a kern field to remove the kern between the two glyphs. Activate the Text tool to see a small Info box for the preceding glyph. The box provides the name of the preceding glyph, its lock status, and kerning group name:

See section 9.2.4, ‘Kerning Groups’ (p. 128) for details on the ‘Group’ fields in the Info box and section 9.2.5, ‘Kerning Group Exceptions’ (p. 129) for working with the kerning  locks.

Editing kerning in the Info box can be handy for quick adjustments. When kerning many glyph pairs, consider using the kerning shortcuts.

9.2.3 Kerning Shortcuts

Mnemonic: the Control key is located on the left, and the Command on the right.
Control: left glyph.
Command: right glyph.

The kerning shortcuts can be used with the Text tool. Type the two glyphs in Edit View, then place the cursor between them and hold down the Option and Control keys to change the kerning value between the current and the left glyph with the Left Arrow and Right Arrow keys ($\wedge\lrcorner\blacktriangleleft$ and $\wedge\lrcorner\blacktriangleright$). Hold down Option and Command to change the kerning between the current and the right glyph ($\lrcorner\text{⌘}\blacktriangleleft$ and $\lrcorner\text{⌘}\blacktriangleright$). Hold down Shift for steps of 10.

Ctrl-Opt-Left	increase kerning on left side
Ctrl-Opt-Shift-Left	increase kerning on left side by 10 units
Ctrl-Opt-Right	decrease kerning on left side

Ctrl-Opt-Shift-Right	decrease kerning on left side by 10 units
Cmd-Opt-Right	increase kerning on right side
Cmd-Opt-Shift-Right	increase kerning on right side by 10 units
Cmd-Opt-Left	decrease kerning on right side
Cmd-Opt-Shift-Left	decrease kerning on right side by 10 units

9.2.4 Kerning Groups

Many glyphs with similar forms should have the same kerning values. Kerning groups capture these similarities and reduce the number of pairs that need to be set manually. Kerning then applies not just to pairs of glyphs, but to groups of pairs of glyphs. For example, B, D, E, F, H, and so on may form a left side kerning group. Also, V, W, \acute{W} , \grave{W} , \tilde{W} , and so on might share both the left and the right kerning groups. For example, if A and \grave{A} are in the same kerning group, kerning A to V also kerns the \grave{A} -V pair. Additionally, if V is in a group with W, A-W and \grave{A} -W will also be kerned by the same amount. All members of a kerning group have the same kerning values unless a kerning exception is defined (see section 9.2.5, ‘Kerning Group Exceptions’, p. 129).

A glyph can have two kerning groups: one on the left and one on the right. Glyphs written from top to bottom have a top and bottom group instead.

The name of a kerning group is written into the ‘Group’ fields in the Info box (*View > Show Info*, Cmd-Shift-I). Edit the kerning groups of multiple glyphs in the inspector in Font View (see p. 72), or select multiple glyphs in Edit View. Group names can contain the letters a–z and A–Z, the digits 0–9, and an underscore (), period (.), or hyphen (-). Rename kerning groups in the Kerning Window (see p. 129).



It is common to name kerning groups after a base glyph. For example, R, \acute{R} , \grave{R} , and \tilde{R} may all have the right kerning group ‘R’. The left kerning group might be named ‘I’ or ‘H’. However, it can also be named ‘stem’ or any arbitrary name.

Another example: the glyphs c, d, e, o, and q could share a left kerning group named ‘o’. In that case, kerning ‘To’ will also kern all other group members. However, while ‘To’ might look good, kerning ‘Tö’ by the same amount will probably look too tight. There are two solutions for this issue: either ö could be placed in a different left kerning group than o, or kerning group exceptions (see p. 129) could be used for ö.

Show all kerning group members in the background by choosing *View > Show Group Members*. The group members are only shown for the current glyph and the one placed before it on the canvas. Only members of the respective kerning group are shown. For example, when editing the P glyph, only its left kerning group members are shown in the background. This display option is helpful to spot possible collisions.

9.2.5 Kerning Group Exceptions


A kerning group exception overwrites the kerning values of a group for a specific pair. For example, if o and ö have the same left kerning group, the kerning value of ‘To’ might be too tight for ‘Tö’.

Create a group exception for ‘Tö’ by selecting the Text tool, then place the text cursor between T and ö. In the Info box (*View > Show Info*, Cmd-Shift-I), open the lock  of the ö to mark it as an exception. Now, editing the kerning value between T and ö will not affect any of the other glyphs having the same left kerning group as ö. However, keep the lock of the T glyph locked  so that the new T-ö kerning exception also applies to other group members of the right ‘T’ kerning group, for example, Ṫ-ö , Ṭ-ö , and Ṱ-ö . Close the kerning lock to remove an exception.

9.2.6 Kerning Window

Open the Kerning Window by choosing *Window > Kerning* (Cmd-Opt-K). It provides a list of all kerning pairs for the currently selected writing direction. See section 4.9.2, ‘Writing Direction’ (p. 46) for details on changing the writing direction.

Viewing Kerning Pairs

Enter a search term in the search field to filter for kerning pairs containing it. Click the search icon  to configure the search options. Choose how the search term should be applied (kerning pair *contains*, *is exactly*, *begins with*, *ends with*, or *does not contain* the term), which pairs to show (*is a group* or *is not a group*), on which side of the pairs to search (*left* or *right*), or whether or not to show only glyphs that are selected in Font View. Choose *Reset* to reset all options back to their default.

Kerning groups are shown in blue with an at-sign (@) before their name, while single glyphs are shown in gold. Click a column header (*Left*, *Value*, *Right*) to sort by that column. Click again to

reverse the sort order. Sorting by value is helpful to spot particularly small or large kerning values.

Click a kerning pair to display it in Edit View. The current glyph pair in Edit View, if any, will be replaced by the selected pair. Use the Up and Down arrow keys to walk through the kerning pairs of the font. When selecting a row containing one or two kerning groups, only a single example is shown. Insert all kerning pairs belonging to the selected row into Edit View by choosing *Show All Glyphs* from the actions ☺ menu.

Editing Kerning Pairs

Double-click an entry in the Kerning Window to edit it. Edit a value to adjust the kerning between a pair. Edit a group name to rename the group. Renaming a group will prompt whether to change the group name for only the selected kerning pair, or rename the group across the entire font. Since left and right kerning groups are separate, renaming a group in the *Left* column will not affect any entries in the *Right* column and vice versa.

Copy the selected kerning pairs with *Edit > Copy* (Cmd-C). Select all kerning pairs by choosing *Edit > Select All* (Cmd-A). Switch to another font master and paste the copied pairs with *Edit > Paste* (Cmd-V). If pasting would overwrite existing kerning pairs, Glyphs will show a warning. Choose *Overwrite* to paste the copied pairs and overwrite any the conflicting kerning pairs. Choose *Keep Existing* to paste only non-conflicting pairs.

Delete all selected kerning pairs by pressing the minus — button in the bottom-left of the Kerning Window. Hold down the Option key to delete the pairs from all masters.

Maintaining Kerning Pairs

Kerning pairs can become obsolete when glyphs are removed from the font, or a kerning group no longer has any members. Delete all obsolete kerning pairs by choosing *Clean Up* from the actions ☺ menu.

Choose *Compress* from the actions ☺ menu to reduce the number of kerning pairs in the font. When a few glyphs were kerned with each other before the kerning groups were set, the singleton kerning pairs can be converted into group kerning with the *Compress* action. Compressing kerning changes kerning pairs between two glyphs or between a glyph and a group such that

Kerning groups are used wherever possible. Kerning exceptions are kept unless the exception has the same kerning value as the group kerning (meaning, it's not a real exception). The *Compress* action might need to be run multiple times to compress all kerning pairs fully.

For example, consider a kerning pair T–m, which is a kerning between the glyphs T (right group: @T) and m (left group: @n). Compressing once will turn it into the kerning pair @T–m. Compressing a second time will turn it into @T–@n. Now, this kerning pair applies to all glyphs in the @T right group and the @n left group, such as T̃–m, T–n, T–ŋ, and T̃–ŋ. However, it will keep exceptions such as T–ñ, provided it has a different value than T–n.

9.2.7 Manual Kerning Code

While kerning is implemented as an OpenType feature, it does not appear in the *Features* tab of the Font Info window. Instead, it is added as an implicit feature (see p. 105). However, an explicit ‘kern’ feature can also be added for performing contextual kerning.

In *File > Font Info... > Features*, add a new feature named ‘kern’. The ‘kern’ feature typically uses positioning (**pos**) rules to change the distance between glyphs.

Contextual kerning matches more than two glyphs and can be helpful when dealing with punctuation or spaces. For example, the f might get some extra spacing to the right when it is followed by a space glyph and a glyph that extends far to the left on the top, such as T or V.

```
pos f' 50 space [T V W Y];
```

Kerning groups can be reused in feature code. The name of a left kerning group is prefixed by @MMK_R_ while a right kerning group is prefixed by @MMK_L_. For example, the left kerning group @H has the feature class @MMK_R_H and the right group @quote has the class @MMK_L_quote. Note that the R and L in the prefixes are with respect to the other glyph of the kerning pair: a left kerning group is used on the *right* (R) of a pair while a right kerning group is used on the *left* (L) of a pair.

For example, consider the glyph sequence L–quoteright–A (L'A). With pair kerning, the quote might get pulled into the white space of the L and also in the white space of the A, making the L and A collide on the baseline: L'A. A contextual kern can fix

this situation by pulling the quoteright into the white space of the L (-50) while pushing the A to the right (70):

```
pos @MMK_L_L' -50 quoteright' 70 @MMK_R_A;
```

Manual kerning code can also make use of Tokens (see p. 188), which allow the glyph positioning to be based on Number Values of masters or the sidebearings of glyphs. See section 13.1, ‘Tokens’ (p. 188) for details.

10 Reusing Shapes

10.1 COMPONENTS

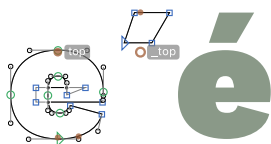


Components are glyphs used as shapes inside other glyphs. For example, the A glyph can be reused to build the Å, Ǻ, ǻ, Ǽ, Ǿ, and many more glyphs. Diacritics like ̈, ̉, ̊, ̋, and ̌ can also be used as components so that other characters like Ö, È, Ç, Ñ, Û, and many more can be formed.

The original glyph from which a component is derived is referred to as the *base glyph* of the component. A glyph that is built out of components is referred to as a *compound* or *composite glyph*. Changing a base glyph also changes all of its copies as a component. When changing a base glyph in Edit View, all related components will update live to reflect the newly made changes.

On export, components are decomposed to paths with the *Remove Overlaps* option selected. TrueType flavor fonts without the *Remove Overlaps* option active (such as variable fonts) keep non-overlapping components while decomposing overlapping components. Set the 'Keep Overlapping Components' custom parameter on an instance to also keep overlapping components in TrueType flavor fonts.

10.1.1 Building Composites



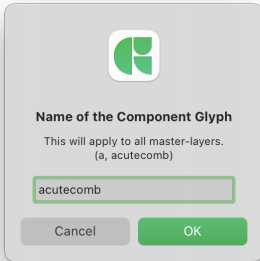
When adding a new composite glyph, Glyphs will automatically add the necessary components. For example, adding an eacute (é) glyph to a font will automatically build it with an e component and an acutecomb (◌́) component. Glyphs uses the glyph info database (see p. 77) to figure out which components to insert into which glyphs. This automatic insertion does not work if any of the required components do not exist in the font.

Convert an existing glyph to a composite by choosing *Glyph > Create Composite* (Cmd-Ctrl-C). This command will replace all paths and components already present on the current layer. Hold down the Option key to apply the command to all masters of the glyph (Cmd-Ctrl-Opt-C).

Add components by name or Unicode value by choosing *Glyph > Add Component* (Cmd-Shift-C). Hold down Option to add the component on all masters (Cmd-Opt-Shift-C). The component picker works the same as the glyph picker used to

insert glyphs in Edit View. See section 4.9.1, ‘Text Tool’ (p. 45) for details.

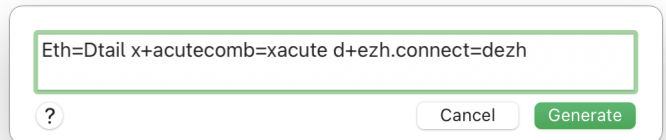
10.1.2 Turning Paths into Components



Create a new component by selecting paths and components on a glyph layer and choosing *Component from Selection* from the context menu or the *Glyph* menu. A dialog window appears, prompting for the glyph name of the new component. Glyphs will suggest a component name based on context, but the name can be changed to any other name as well. Prefix the name of a component with an underscore (`_`) if the component should only be used inside the Glyphs file and not be part of the exported glyphs. See also the exports glyph property (see p. 98).

Confirm the dialog with *OK*. Glyphs adds the newly created component to the current Edit View tab and activates it. The previously selected shapes will be replaced by the component.

10.1.3 Recipes



When adding glyphs to a font using *Glyph > Add Glyphs...*, recipes can be used to define which components should be used in which glyphs. A recipe is a formula of glyph names that is used to create a new glyph. The following recipe formulas can be used:

- ▶ `component = glyph`, for example, ‘Eth=Dtail’, which adds the Dtail glyph with the Eth as a single component.
- ▶ `base + mark = glyph`, for example, ‘x+acutecomb=xacute’, which adds multiple components (separated by a plus sign) to a new glyph. The mark glyphs are automatically aligned to the base glyph using anchors.
- ▶ `base + base = glyph`, for example, ‘d+ezh.connect=dezh’. Adding multiple base components to a glyph can be useful when building ligatures.

10.1.4 Editing Components

Click to select a component. When dragging with the Select tool, components are ignored unless the Option key is held down. Press the Tab key to select the next component, or Shift-Tab to select the previous one. When paths are present on a layer, the first *Edit > Select All* (Cmd-A) selects paths only. Press Cmd-A once again to also select all components on the layer.



With one or multiple components selected, a component Info box appears next to the glyph Info box. See section 4.6, ‘Info Box’ (p. 40) for details on the component Info box.

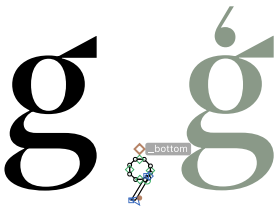
Transform components using the Info box, the bounding box (see p. 26), or the Palette (see p. 62). Flipping a component vertically switches its **top** and **bottom** anchors to be again on top and bottom of the component. This anchor switch is helpful, for example, when building the gcommaaccent (ğ) from a g and a flipped commaaccentcomb (̆).

Move a selected component with the mouse or the arrow keys. Auto-aligned components can either be moved along the vertical axis or not at all. See section 10.1.8, ‘Automatic Alignment’ (p. 137) for details. When moving with the arrow keys, hold down Shift for increments of 10 and Command for increments of 100. Option-drag a component to duplicate it. Delete all selected components by pressing the Delete key.

10.1.5 Moving between Base Glyphs and Composites

Double-click a component to edit its glyph. The component glyph will be placed next to the composite glyph in Edit View and activated for immediate editing. Alternatively, click the arrow button located in the top-left of the component Info box to edit its glyph. (For the Info box to be shown, the component must be selected and *View > Show Info* must be checked.)

Show all glyphs that use the current glyph as a component by Control-clicking or right-clicking the canvas and choosing *Show all glyphs that use this glyph as a component* from the context menu.



10.1.6 Component Placeholders

Component placeholders indicate a problem with a component. There are three types of problems.



An **empty base glyph** placeholder is shown if the glyph layer of a component is empty. Remove the component or fill the glyph layer.

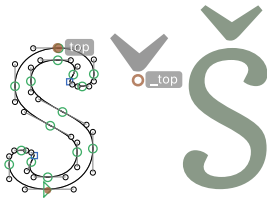
When **no base glyph** is shown, the component references a glyph that does not exist (or no longer exists). Remove the component or add the referenced glyph to the font. Click the placeholder to see the referenced glyph name in the Info box (Cmd-Shift-I).

A **bad reference** indicates a circular reference, where the component contains a composite glyph that contains that component.

10.1.7 Anchors

Glyphs uses anchors for OpenType features (such as mark-to-base positioning and cursive attachment, see section 4.4, ‘Anchors’, p. 35) and for arranging components. Often, the same anchors can be used for both features and for arranging components.

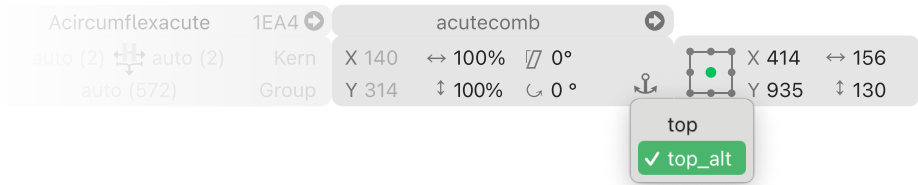
Unicode defines base characters like S (U+0053) and combining marks like ̣ (U+030C). Typing a base character followed by a combining mark will place the mark on the base glyph using the anchors. But Unicode also contains precomposed characters like Š (U+0160). Glyphs for these characters can be built by using the base and mark glyphs as components.




For example, a mark component with a `_top` anchor snaps onto the `top` anchor of a base component just like mark-to-base positioning would place a mark glyph atop a base glyph. See section 4.4.1, ‘Adding, Editing, & Removing Anchors’ (p. 35) for general information on working with anchors.

In some situations, a component might need multiple anchors of the same type. For example, it is common to use such alternative `top` anchors for Vietnamese diacritics. Alternative anchors share the name of the original anchor but use an underscore suffix for differentiation. So, there might be a `top`

and a `top_alt` or `top_viet` anchor on the same glyph layer. The suffix after the underscore can be chosen freely.



Switch between anchors by selecting an attached component and picking a different anchor from the  menu in the component Info box. Show the Info box with *View > Show Info* (Cmd-Shift-I). The anchor icon is only visible if multiple anchors are available.

10.1.8 Automatic Alignment

Glyphs built entirely from components are called composite or compound glyphs and can use automatic alignment to keep their metrics in sync with their base glyph. For example, changing the left sidebearing of the U glyph would automatically adjust the glyphs Ú, Û, Ü, Ũ, Ū, and all other composite glyphs built with the U component.

Automatic alignment is enabled by default if a glyph layer contains only component shapes (no path shapes). Control-click or right-click an auto-aligned component and choose *Disable Automatic Alignment* from the context menu to place the components manually. Choose *Enable Automatic Alignment* from the context menu on a component to enable it again. Automatic alignment can also be disabled for the entire font in *File > Font Info... > Other > Disable Automatic Alignment*. See also section 7.5.4, ‘Disable Automatic Alignment’ (p. 108).



Auto-aligned components are shown in green to differentiate them from normal, gray components. If a component has the category *Number* (for example, the glyphs zero–nine) and it is the only component in a composite glyph, it is shown in blue and can be shifted vertically while staying auto-aligned horizontally.

Metrics keys cannot be used for auto-aligned composite glyphs, since the sidebearings are derived from the base components. There is one exception: auto-aligned composite glyphs can add to and remove from the automatic sidebearing values using the `+=` and `-=` operators. For more information, see section 9.1.4, ‘Metrics Keys & Automatic Alignment’ (p. 126).

Automatic alignment is typically for single components, a base

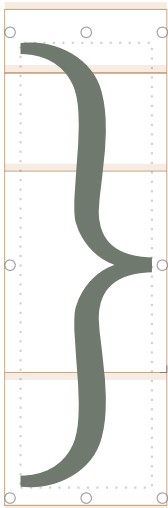
component with one or more mark components, or multiple base components. These three setups are described in more detail in the following sections.

Single Component

A composite glyph may use a single auto-aligned component. This setup is helpful for glyphs that look the same but should still be encoded as two separate glyphs. That might be the case with Greek, Latin, and Cyrillic capital letters sharing the same outlines, such as Alpha/A/A-cy, Beta/B/Ve-cy, and Rho/P/Er-cy. These glyphs belong to different scripts and may use separate kerning groups, so they should be encoded as separate glyphs instead of a single glyph with multiple Unicode values.

A single component might also be handy when constructing the glyph nine as a flipped six. As mentioned above, such a component will be shown in blue and can be shifted vertically while keeping the automatic horizontal alignment.

Punctuation marks such as < and >, ¿ and ?, { and }, and arrows such as ← and → can also often be built from a single flipped component.



braceleft →

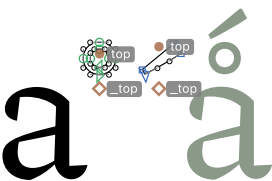
X 327	↔ -100%	↙ 0°
Y 0	↓ 100%	↻ 0°

Base Component & Mark Components

Build precomposed glyphs such as Ú, Ù, Ů, Ü, Ū from a base component (U in this case) and diacritical marks. First, add the base component, followed by combining mark components. For example, for Uacute (Ú), add the U component and then the acutecomb (◌́) component. If the glyph is known to the glyph info database—as is the case with Uacute—choose *Glyph > Create Composite* (Cmd-Ctrl-C), and all required components will be added in the correct order.

Use anchors to place marks atop, below, or over the base component. See section 10.1.7, ‘Anchors’ (p. 136) for details. When adding multiple combining marks to a composite glyph, marks stack in the order in which they are added. For this to work, mark glyphs need to contain both a base attachment anchor (such as `_top`) and a mark attachment anchor (such as `top`). Rearrange marks using the *Shape Order* filter (see section 12.2.1, ‘Shape Order’, p. 179).

Mark components positioned with anchors do not affect the sidebearings of the composite glyph. For example, placing a



macroncomb (̄) atop an idotless (i) may result in the macron reaching outside the glyph box of the imacron (i). In such a case, either design a narrower macron and use that for the imacron or add additional space to the sidebearings of the composite glyph (see section 9.1.4, ‘Metrics Keys & Automatic Alignment’, p. 126).

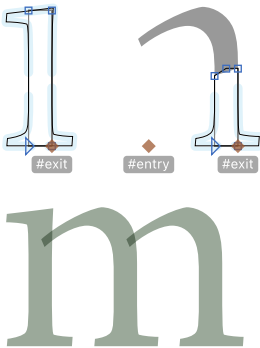
See the Diacritics tutorial¹ for a guided introduction to working with diacritics.

Multiple Base Components

A composite glyph may also contain multiple base components. They are placed next to each other as if they were typed together. Combining marks can be placed on each one of the base components individually; place a base component first, then all of its marks, then the next base component followed by its marks.

Adding multiple base components to a composite glyph is handy when building ligature glyphs. Using components for ligatures is particularly helpful for scripts like Arabic that make extensive use of ligatures.

Base components can be connected using the same anchors that are used for cursive attachment (see p. 36). This technique can build an n glyph from a `_part.stem` and a `_part.arch` component, where the `_part.stem` has an `#exit` anchor and `_part.arch` has an `#entry` anchor. Add both components to the n glyph, and they will connect such that the exit and entry anchors overlap. Add an `#exit` anchor to the `_part.arch` to build an m composite glyph from a `_part.stem` and two `_part.arch` components. See section 10.1.14, ‘Underscore Components’ (p. 141) for details on working with components with a name beginning with an underscore.



10.1.9 Locking Components

Lock components that are not automatically aligned by Control-clicking or right-clicking it and choosing *Lock Component* from the context menu. Locked components cannot be moved. Unlock a component by choosing *Unlock Component* from the context menu.

1 glyphsapp.com/learn/diacritics

10.1.10 Decomposing

Convert all components inside a glyph to paths by choosing *Glyph > Decompose Glyph* (Cmd-Shift-D). All components, including all nested components, will be decomposed into their paths and anchors.

Choose *Decompose* from the context menu on a component to only decompose the selected components. In this case, nested components will not be decomposed.

10.1.11 Combining Paths and Components

As soon as there is a path on a layer, automatic alignment is disabled. Therefore, be careful when combining components and paths because shifts may occur, especially if the base glyph of the component is changed. Instead, consider building the glyph from components only and connecting them with anchors as outlined in section 10.1.7, ‘Anchors’ (p. 136), thus enabling automatic alignment. Consider defining glyphs as non-exporting if they are only ever used as parts of other glyphs. See section 10.1.14, ‘Underscore Components’ (p. 141) and section 10.2, ‘Smart Components’ (p. 141) for working with non-exporting components.

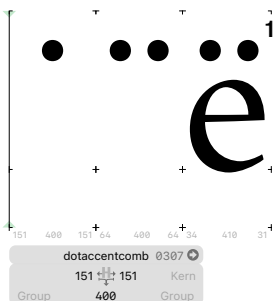
When dragging an element, such as a node or an anchor, the nodes inside components are highlighted. Dragging an element near such a highlighted point snaps it to the position.

Choosing *Path > Align Selection* (Cmd-Shift-A) while exactly one point and one component are selected will align the origin point of the component to the selected node. The node keeps its position. The origin point is where the baseline crosses the left sidebearing if the italic angle is zero. If the component glyph contains an **origin** anchor, it is used as the origin position instead.

10.1.12 Nesting Components

Glyphs allows components to be nested. For example, the dieresiscomb (◌̈) can be built from two dotaccentcomb (◌̇) components and then be used as a component in glyphs such as edieresis (ë).

top and **bottom** anchors propagate from a component to the glyph in which the component is placed. For example, consider a glyph E that has a **bottom** anchor. When a composite glyph, like Edieresis (Ë), uses E as a component, it



inherits the `bottom` anchor from the E. That way, bottom marks such as circumflexbelowcomb (◌̣̂) can attach to the bottom of Edieresis, even though it itself does not contain a `bottom` anchor. Propagated anchors are overwritten by anchors in the composite glyph. Continuing the example, if the Edieresis glyph had its own `bottom` anchor, then that would overwrite the `bottom` anchor of the E component.

Add a ‘Propagate Anchors’ custom parameter to the *Font* tab in *File > Font Info...* with its checkbox unselected to prevent anchor propagation.

10.1.13 Preferred Marks for Glyph Composition

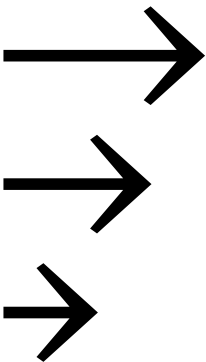
When automatically building composites, Glyphs prefers marks with the same name suffix as the composite. For instance, when composing `agrave.sc`, Glyphs prefers a mark glyph named `gravecomb.sc` over `gravecomb` if such a glyph exists. When building uppercase letters, marks with a ‘.case’ suffix are preferred. For example, when building the Agrave glyph, Glyphs will use the mark `gravecomb.case`—if it exists—over `gravecomb`. This is helpful when the ‘.case’ diacritical marks are flattened to accommodate the limited vertical space available in the uppercase letters. The glyphs `i` and `j` are built from `idotless` and `jdotless` with a `dotaccentcomb`. When building composite glyphs based on `i` and `j`—such as `imacron` (ï) or `jcircumflex` (ÿ)—Glyphs prefers mark glyphs ending in ‘.i’ or ‘.narrow’.

10.1.14 Underscore Components

By default, a glyph whose name starts with an underscore is not exported. This allows for glyphs solely used as components in other glyphs. Such glyphs might help to create ligature glyphs. For example, both glyphs `fl` and `f_f` may use an `_f.connect` component for the first `f`.

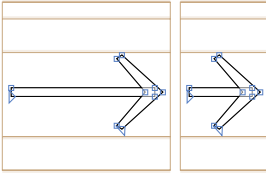
10.2 SMART COMPONENTS

Smart components allow for variation of design properties within a single component glyph. Each smart component can have its own set of variation properties such as *width*, *height*, *angle*, *roundness*, or any other arbitrary design feature that can vary along an axis. In this way, smart components work like multiple masters or a variable font, but with the variation axes exclusive to a component.



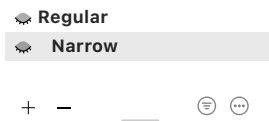
Smart components were initially designed for Asian scripts, where shapes are frequently reused with slight modifications, such as the strokes in CJK fonts or the components of the Tibetan script. This concept—reusable components with property variations—has proven useful for various scripts and glyphs.

10.2.1 Setting up Smart Glyphs



Above: The two layers of a `_part.arrow` smart glyph.

Below: The *Layers* palette for the smart glyph.



The glyph of a smart component is called a smart glyph. All CJK radical and Korean base glyphs are smart glyphs. In addition, all glyphs with a name starting with ‘`_smart.`’ or ‘`_part.`’ are smart glyphs.

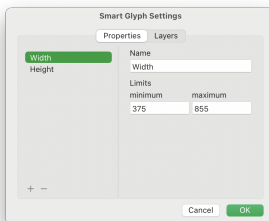
The different forms of a smart glyph (for example, *Narrow* and *Wide* or *Sharp* and *Rounded*) are placed on smart layers. Click the plus $+$ button in the bottom-left of the *Layers* palette to create a new layer. New layers are named after the current date and time, but that is not a good description for a smart layer. Double-click the smart layer name and change it to a more descriptive one such as ‘*Narrow*’, ‘*Low Contrast*’, ‘*Small*’, or any other name that best describes the variation. For instance, for a `_part.arrow` glyph with a *Width* property, draw a normal arrow on the *Regular* master layer and a narrow version of the arrow on an additional layer aptly named ‘*Narrow*’. Note that smart layers need to be compatible with the master layer. For more on layer compatibility, see section 11.5, ‘*Outline Compatibility*’ (p. 162).

Smart glyph properties control how the smart component interpolates between the different layers. Add a property by opening the *Smart Glyph Settings* with *Edit > Info For Selection* (Cmd-Opt-I) or by choosing *Show Smart Glyph Settings* from the component context menu. It is split into two tabs: *Properties* and *Layers*.

Smart Glyph Properties Settings

Create a new property by clicking the plus $+$ button located in the bottom-left of the *Properties* tab. Smart glyph properties are similar to font axes. They have a name as well as minimum and maximum values. The name can be chosen freely and may be unrelated to the layer names or axes names of the font. The names ‘*Width*’ and ‘*Height*’ have a special meaning; see section 10.2.3, ‘*Width & Height Properties*’ (p. 144) for details.

The minimum and maximum values can be any number range, but the minimum needs to be less than the maximum. Values can

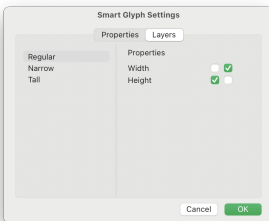


be negative. The default range of 0–100 works well for abstract properties such as ‘Contrast’ or ‘Curviness’. If the variation property is less abstract and related instead to font units—such as ‘Height’ or ‘Descender Depth’—consider using measurements as minimum and maximum values. Continuing the `_part.arrow` example, if the normal arrow is 855 units wide and the arrow on the ‘Narrow’ layer is 375 units wide, add a ‘Width’ property with a minimum of 375 and a maximum of 855.

Note that smart components can be interpolated *and extrapolated*. This allows the usage of values below the minimum and above the maximum. For example, a `_part.arrow` component with a ‘Width’ property ranging from 375 to 855 can also be used with a ‘Width’ value of 300 or 900. Extrapolation works well for simple variations such as the tail of an arrow getting longer and shorter, but might produce less desirable results for more complex smart components such as the arch of an `n` getting wider and narrower.

Use a minimum of 0 and a maximum of 1 for properties that should not interpolate and instead only be used as an on/off toggle.

Smart Glyph Layers Settings



The *Layers* tab connects smart properties with smart layers. The sidebar shows all smart layers of the glyph. Select a layer and set whether it belongs to the minimum or maximum value. If a layer is unrelated to a property, do not check any of the two values. For instance, for a glyph with two layers—‘Regular’ and ‘Narrow’—and a ‘Width’ property, check the minimum *Width* value for the *Narrow* layer and the maximum *Width* value for the *Regular* layer.

A setup with multiple properties does not require a layer at all extremes. For example, a smart glyph with two properties—‘Width’ (range 375–855) and ‘Height’ (range 80–360)—only requires the following three layers:

	Width	Height
Regular	375	80
Wide	855	80
Tall	375	360

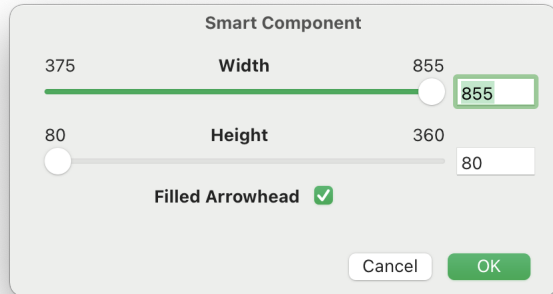
Note that a layer at both maximum values (*Width* of 855 and *Height* of 360) is not required. Glyphs infers it from the *Wide* and

Tall layers. Consider adding a layer at both extremes only if the inferred outlines are not as desired.

10.2.2 Using Smart Components

Add a smart component like any other component (as described in section 10.1.1, ‘Building Composites’, p. 133). A smart component is indicated by a *Smart* badge in its Info box.



Smart Component dialog as shown with *Edit > Info for Selection* (Cmd-Opt-I) for a selected component.



If the Info box is not visible, select the smart component and check *View > Show Info* (Cmd-Shift-I). Click the *Smart* badge in the Info box or choose *Edit > Info for Selection* (Cmd-Opt-I) to configure the selected smart components. If the Palette is visible (*Window > Palette*) and *View > Show Info* is checked, the smart component settings are also visible at the bottom of the Palette.

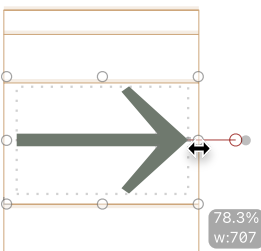
Smart Settings



Use the slider  to set the setting of a property anywhere from the minimum to the maximum value. Use the number field to input any value, which might also extrapolate beyond the minimum and maximum values. If the min/max values are 0/1, a checkbox  is displayed instead.

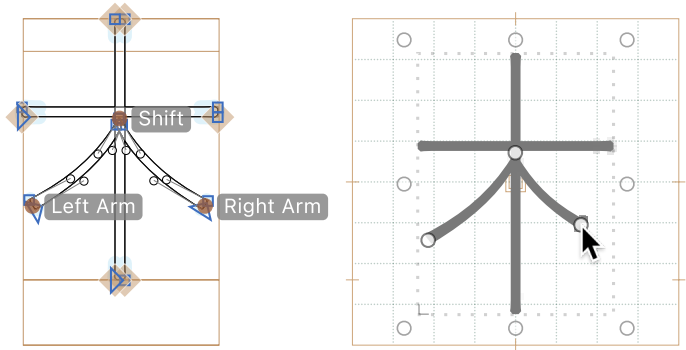
10.2.3 Width & Height Properties

Name smart glyph properties ‘Width’ or ‘Height’ (with a capital letter) to control them with the bounding box. Select *View > Show Bounding Box* (Cmd-Opt-Shift-B) and click on a smart component to show its bounding box. Resizing the bounding box horizontally changes the ‘Width’ property; resizing vertically changes the ‘Height’ property. Resizing the bounding box is impossible for a smart component with neither a *Width* nor a *Height* property.



10.2.4 Smart Handles

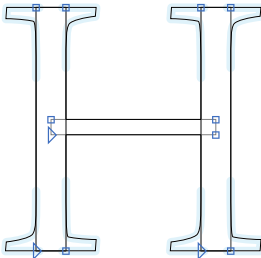
Left: Placing anchors with the names of properties on all layers. Right: Moving a knob to adjust smart settings.



Smart handles allow properties to be modified using control handles placed on the smart glyph. Place an anchor with the name of a smart glyph property on every smart glyph layer. The anchor names must match the name of the property exactly, including capitalization. On smart glyph layers affecting the property, move the anchor to match the modified outlines. These anchors do not need to be added for all properties.

When using the smart glyph as a component, gray handles will appear when the component is selected. Click and drag a handle to change its property value.

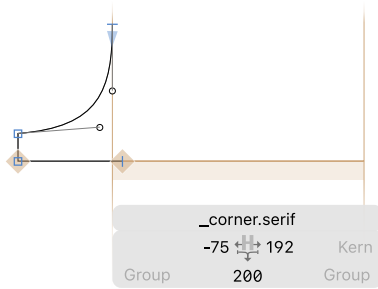
10.3 CORNER COMPONENTS



Corner components are open paths that can be fitted onto the corners of a path. The main usage of corner components is adding serifs to stems, but they can also be used in other circumstances.

10.3.1 Creating Corner Glyphs

The name of a corner glyph starts with ‘_corner.’ followed by an arbitrary corner name, for example, ‘_corner.serif’. A corner glyph contains an open path flanking the layer origin. The layer origin is at coordinates (0, 0). If an anchor named `origin` is placed on the layer, it is used as the origin point.

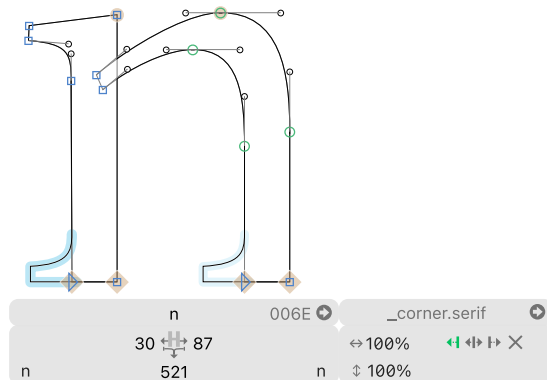


The open path of the corner component should follow the same direction as it would if it were part of a stem. Typically, the start node of the path is placed on the same vertical or horizontal axis as the origin point. Therefore, either the X or Y coordinate of the start node is the same as the X or Y of the layer origin.

The relative position of the end node defines the direction in which the corner expands from the stem. For instance, place the end node to the right of the layer origin if the corner should expand to the left of the stem.

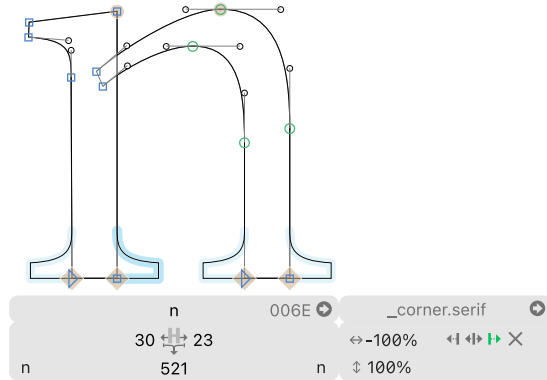
If the path direction is not set correctly, the corner will extend in the wrong direction (for example, to the bottom instead of to the left). Fix the direction by selecting the open path and choosing *Reverse Selected Contours* from the context menu.

10.3.2 Using Corner Components

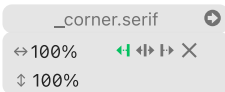


Add a corner component to a node in another glyph by selecting the node and choosing *Add Corner Component* from the context menu. A glyph picker will open, showing corner glyphs. Search for the desired corner and insert it by pressing Return.

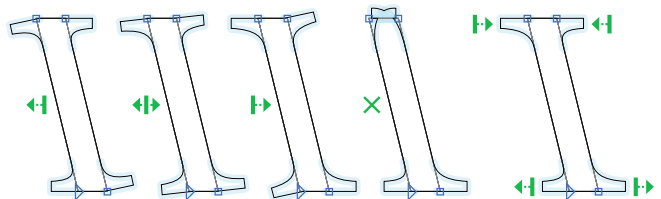
In Edit View, a corner component is highlighted with a subtle blue background. Click on a corner component to select it. Shift-click to select multiple corners. A selected corner can be copied with *Edit > Copy* (Cmd-C) and pasted onto other nodes with *Edit > Paste* (Cmd-V). Press the Delete key to remove the selected corner components. Control-click or right-click a corner and choose *Decompose Corner* to replace the component by its path.



Select a corner component to view its Info box. *View > Show Info* (Cmd-Shift-I) needs to be active. Click the component name in the Info box to replace it with a different component. Click the arrow button to add and activate the corner glyph in Edit View. Use the horizontal and vertical scale fields to change the size of the corner. This might be useful for increasing the size of serifs for capital letters or reducing it for small caps. Negative values (such as '-100') may be used to flip the component. This allows a left serif to be reused on the right or a bottom serif on the top. The mirror buttons from the Palette (see p. 62) can also be used to flip corner components.



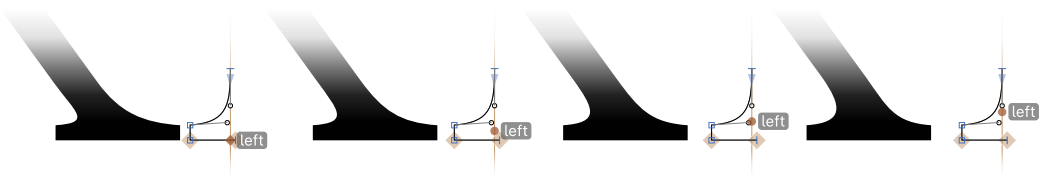
Control the alignment of the corner using the alignment modes in the Info box. This is relevant for diagonal stems:



- ▶ The **↵** left arrow aligns the start node to the diagonal, while the end node of the corner stays put. This mode is typically used for bottom-left and top-right serifs.
- ▶ The **➦** right arrow aligns the end node to the diagonal, while the start node of the corner stays put. This mode is typically used for bottom-right and top-left serifs.
- ▶ The **↔** left-right arrow is a blend of the left and right alignment. This mode is typically used for ink-traps and similar corners.
- ▶ The **✕** x-mark does not align the start or end node to the diagonal.

10.3.3 Adjusting the Entry Point of a Corner

The entry point of a corner component is where the stem connects to the corner. It can be adjusted by adding an anchor to the component glyph. This is particularly useful for attaching corners to diagonal stems:



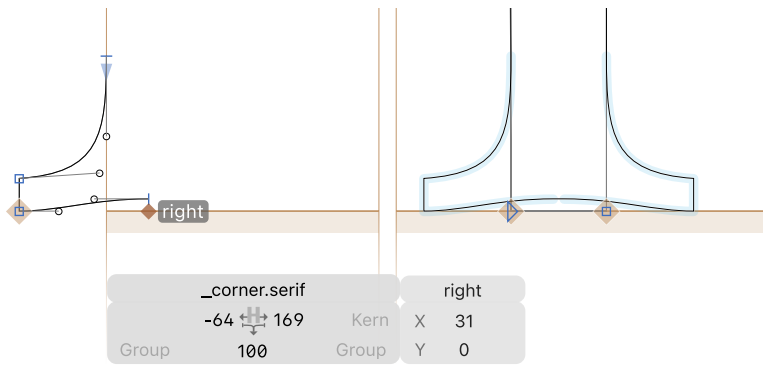
In a typical corner glyph setup with the layer origin at (0, 0) and the start node at X = 0, place the anchor somewhere at X = 0, as shown in the image above.

Add a **left** anchor to a corner glyph to control where the corner meets the diagonal. If the open path in the corner glyph is facing to the right, name the anchor **right** instead. The anchor must be placed on the line crossing the layer origin and the start node of the open path.

Move the anchor away from the start node towards the baseline to widen corners at the large angle of the diagonal and shorten the corner at the small angle of the diagonal (see the left side of the image above). Move the anchor in the other direction to achieve the opposite effect (see the right side of the image above). If no such anchor is placed on the layer, Glyphs assumes it sits at the layer origin (leftmost example in the image above).

10.3.4 Adjusting the Exit Point of a Corner

The exit point of a corner component is where the corner connects back to the stem. It allows the path after a corner to continue at a different position to the path of the stem. This technique is often used for cupped serifs:



Place a **right** anchor to redefine the end position of the corner. For a right-facing corner glyph, place a **left** anchor instead. This anchor is used instead of the end node of the open path as the end position of the corner, which allows the end node to be placed above or below the baseline.

10.3.5 Adjusting the Width and Height

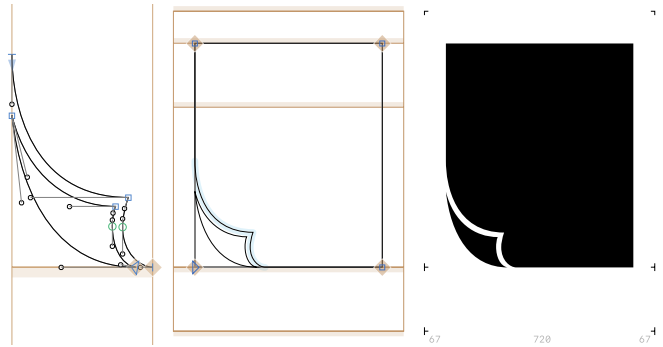
As described above, a corner component can be resized using the \leftrightarrow horizontal and \updownarrow vertical fields in the Info box. By default, the component is scaled by stretching the outlines. Customize the scaling by adding Intermediate layers (see p. 165) to the corner glyph. These Intermediate layers are then used to scale components by interpolation.

For example, add an Intermediate layer with a *Width* of 150 % and *Height* of 100 % to a corner glyph and make the outline of the corner wider. Then, change the \leftrightarrow horizontal scaling to, for example, 125 %. The resulting component outlines are interpolated. Changing the height still stretches the corner, in this example, as no Intermediate layer with a height different from 100 % was defined.

10.3.6 Closed Paths in Corners

Corner glyphs may contain closed paths in addition to the one open path. Closed paths add shapes that can be placed inside or outside the corner.

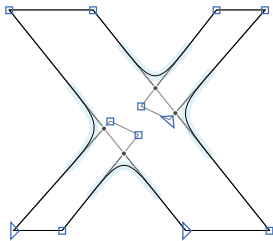
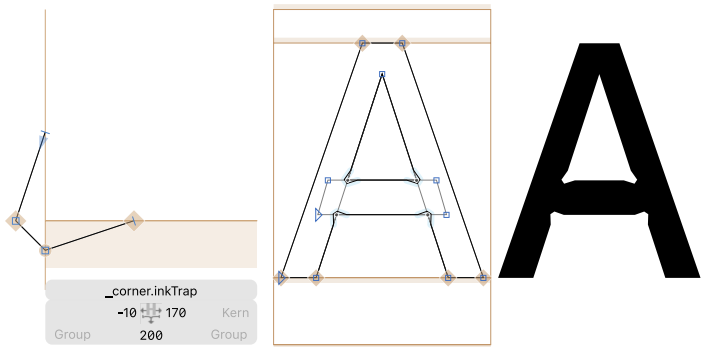
Left: Corner glyph with a closed path. Center: Corner component applied to the bottom-left node of a rectangular path. Right: Preview of the resulting glyph.



The open path must be the first shape on the corner glyph layer. Reorder shapes with *Filter > Shape Order*. See section 12.2.1, ‘Shape Order’ (p. 179) for details.

10.3.7 Extra Nodes

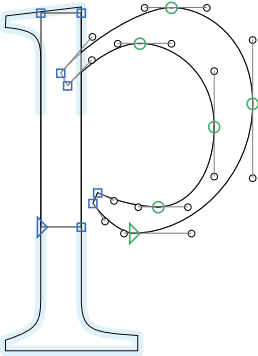
Left: Corner glyph for ink traps. Right: The A glyph with corner components applied to the four extra nodes.



Choose *View > Show Nodes > Extra Nodes* to show the nodes located on the intersections of overlapping paths. These extra nodes can also receive corner components. Attach a corner component to an extra node from the context menu with *Add Corner Component*.

Use corner components on extra nodes for smooth transitions at path intersections, or to add ink traps to stems and crossbars.

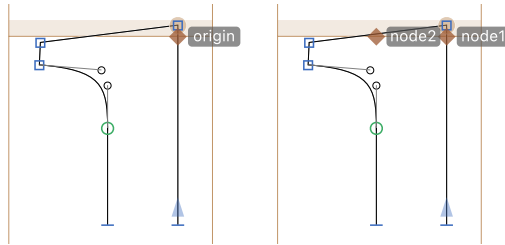
10.4 CAP COMPONENTS



Cap components attach to two nodes. This is in contrast to corner components (see p. 145), which attach to a single node. The two nodes need to be next to each other on the same path. Use cap components for spurs, terminals, flag serifs, and other shapes that appear at the end of strokes.

10.4.1 Creating Cap Glyphs

The name of a cap glyph starts with ‘_cap.’ followed by an arbitrary cap name, for example, ‘_cap.flag’. There are two possible anchor setups:



Tip: Place the anchors on a well-known position, such as the x-height or the baseline. This allows the stem in the referencing glyph to be independent of the overshoot of the cap.

In the first setup, place an **origin** anchor where the first of the two stem nodes should be relative to the cap. This is a simple setup for simple needs.

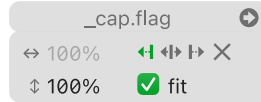
Add the two anchors **node1** and **node2** representing both of the attached stem nodes to control the positioning and scaling of the cap relative to both nodes. This second setup allows for more flexibility when fitting the cap onto stems of varying widths.

10.4.2 Using Cap Components

Attach a cap component to two adjacent nodes of the same path by selecting them and choosing *Add Cap Component* from the context menu. A glyph picker will open, showing only cap glyphs. Search for the desired cap and insert it by pressing Return.

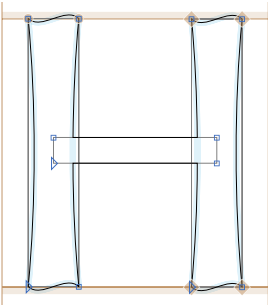
In Edit View, a cap component is highlighted with a subtle blue background. Click on a cap component to select it. Shift-click to select multiple caps. A selected cap can be copied with *Edit > Copy* (Cmd-C) and pasted onto other segments with *Edit > Paste* (Cmd-V). Press the Delete key to remove the selected cap components. Control-click or right-click a cap and choose *Decompose Cap* to replace the component by its path. Select a cap component to view its Info box. (*View > Show Info*,

Cmd-Shift-I, also needs to be selected.)



Check the *Fit* checkbox to automatically match the size of the cap to the size of the stroke to which it is attached. Cap components can be replaced, transformed, and aligned like corner components (see p. 146).

10.5 SEGMENT COMPONENTS



Segment components apply an open path to straight or curved segments. The open path can be a single bend segment or a more complex path containing many nodes and curves.

Glyphs will not add additional nodes when bending a segment component to fit a curve segment. This may reduce the accuracy with which the component follows the curvature of the segment. However, it ensures that the number of nodes stays the same across masters, allowing interpolation for multiple masters and variable fonts.

10.5.1 Creating Segment Glyphs



The name of a segment glyph starts with ‘_segment.’ followed by an arbitrary segment name, for example, ‘_segment.stem’. A segment component contains an open path along the baseline and two anchors, `start` and `end`. Choose *Glyph > Set Anchors* (Cmd-U) to place these anchors at the start and end nodes of the open path. If the `start` and `end` anchors are not placed on the start and end nodes of the path, then the segment component will not match the length of the segment it is applied to.

The start and end nodes of the path should be on the baseline or have both the same distance from the baseline.

10.5.2 Using Segment Components

Select a path segment by clicking it or by selecting both of its nodes, then choose *Add Segment Component* from the context

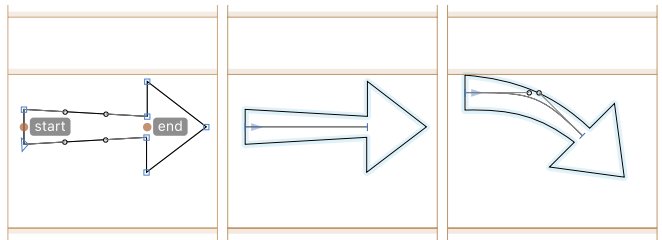
menu. A glyph picker will open, showing only segment glyphs. Search for the desired segment and insert it by pressing Return.

In Edit View, a segment component is highlighted with a subtle blue background. Click on a segment component to select it. Shift-click to select multiple segments. A selected segment can be copied with *Edit > Copy* (Cmd-C) and pasted onto other segments with *Edit > Paste* (Cmd-V). Press the Delete key to remove the selected segment components. Control-click or right-click a segment and choose *Decompose Segment Component* to replace the component by its path. Select a segment component to view its Info box. (*View > Show Info*, Cmd-Shift-I, also needs to be selected.)

Click the component name in the Info box to replace it with a different component. Click the arrow ➔ button to add and activate the segment glyph in Edit View.

10.6 BRUSHES

Brushes expand path segments using custom outlines. A brush can be applied to straight and curved path segments and is most useful when applied to open paths.

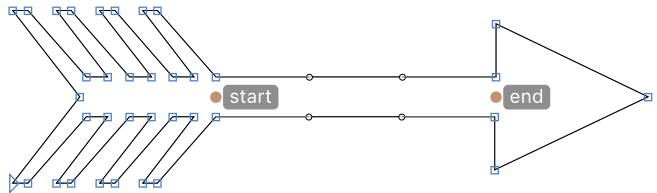


10.6.1 Creating Brush Glyphs

The name of a brush glyph starts with ‘_brush.’ followed by an arbitrary brush name, for example, ‘_brush.arrow’. A brush glyph contains a single closed path and two anchors, **start** and **end**. Choose *Glyph > Set Anchors* (Cmd-U) to place anchors on the left and right of the layer. Typically, the two anchors are placed on the baseline or have both the same distance to the baseline.

A brush glyph may contain only one single closed path. Use *Path > Remove Overlap* (Cmd-Shift-O) to merge multiple closed paths to a single path. For example, a _brush.arrow glyph might contain the shape of a rightward pointing arrow. Place the **start** and **end** anchors such that the part of the brush that stretches

and curves is in between the two anchors. Place non-stretching and non-curving parts of the brush before the **start** and after the **end** anchor, like this:




Parts of the brush that should bend when following a curve segment need handles, as shown in the image above. Add handles to a segment by Option-clicking it.

10.6.2 Using Brushes


Select a path segment by clicking it or selecting both of its nodes and choose *Add Brush* from the context menu. A glyph picker will open, showing brush glyphs only. Search for the desired brush and insert it by pressing Return.

Typically, brushes are applied to open paths consisting of a single segment. Brushes can be added to multiple segments by selecting more than one segment before choosing *Add Brush*. If nodes are selected, choosing *Add Brush* will add brushes such that the selected nodes are the end nodes of the new brushes.

A brush is highlighted with a subtle blue background. Click on the outline of a brush to select it. Shift-click to select multiple brushes. A selected brush can be copied with *Edit > Copy* (Cmd-C) and pasted onto other segments with *Edit > Paste* (Cmd-V). Press the Delete key to remove the selected brushes. Control-click or right-click a brush and choose *Decompose Brush* to replace the component by its path. Select a brush to view its Info box. (*View > Show Info*, Cmd-Shift-I, also needs to be selected.)

Click the component name in the Info box to replace it with a different component. Click the arrow  button to add and activate the brush glyph in Edit View.

10.7 PIXEL TOOL

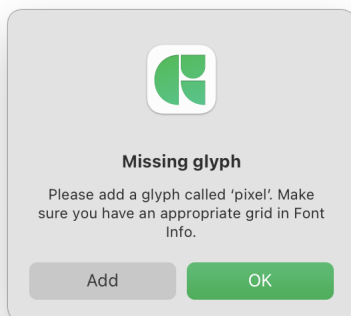
The Pixel tool  (shortcut X, Shift-B if the Pencil tool is selected) draws pixel components. Use the pixel tool to create pixel fonts and pixel symbols.

HELLO
WORLD

10.7.1 Setup

The Pixel tool requires a grid spacing of 2 or above. Change it in *File > Font Info... > Other > Grid Spacing*. See section 7.5.1, ‘Grid Spacing & Subdivision’ (p. 107) for details.

The Pixel tool uses the glyph named ‘pixel’ to draw pixels when clicking and dragging in Edit View. If the glyph named pixel is not already in the font, Glyphs will offer to add it upon a click with the Pixel tool:



Click *Add* to create the pixel glyph. By default, the pixel glyph is a square of the same size as the grid spacing.

10.7.2 Drawing Pixels

Select the Pixel tool, click once, and add a pixel component on the canvas. Click again to remove the pixel. Click and drag the mouse to add multiple pixels. Start to drag on a pixel already in place to remove pixels while dragging.

10.7.3 Pixel Shape

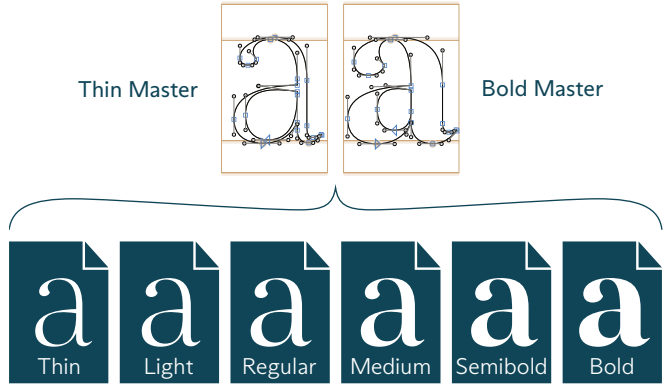
XXXX
XXXX
XXXX
XXXX
XXXX

The pixel glyph can be modified to contain any arbitrary shape. Reduce the grid spacing to a lower number (for example, the default of 1) to draw path segments smaller than a pixel. For instance, the pixel can be an x-mark **X** for a stitch effect. Increase the grid spacing again when using the Pixel tool.

11 Interpolation

The glyph outlines drawn in Edit View belong to a *font master*. Initially, a Glyphs file contains a single master named 'Regular'. If a Glyphs file contains additional masters (for example, a *Thin* and *Bold* master), this is referred to as a Multiple Masters setup. Working with Multiple Masters allows Glyphs to export font instances not just for each master, but also between masters.

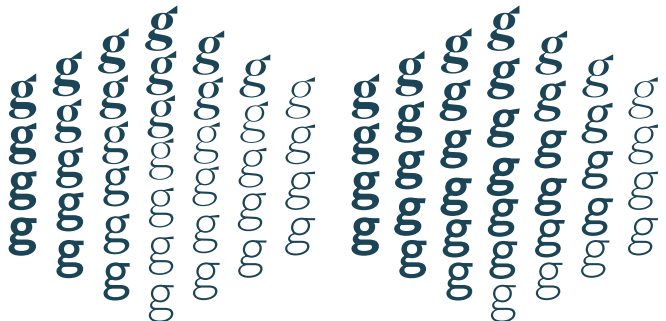
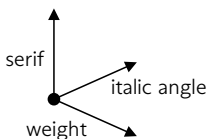
A small number of masters can result in a large number of interpolated instances. In this example, two masters *Thin* and *Bold* are used to produce a total of six instances along a *Weight* axis.



An *interpolation axis* describes the aspect of the glyph design that changes between masters. The most common axes are the *Weight* (from light to bold) and the *Width* (from condensed to extended), but there are many more possible interpolation axes.

A Multiple Master setup can have one or more axes. While a single axis can be thought of as a line along which the design varies, two axes create a two-dimensional space in which every point is a possible font instance. This space is referred to as the *designspace*, which may also have three or more dimensions.

Two perspectives on the three-axes designspace of ABC Arizona, the typeface in which this handbook is set.



11.1 INTERPOLATION APPLICATIONS

Interpolation axes have two use cases: static instances and variable fonts.

A **static instance** is a font file at a specific point in the designspace. For example, consider a Glyphs file with a *Weight* axis spanning from a *Thin* master to a *Bold* master. Then, a *Thin* instance is located at one end of the axis, a *Bold* instance is located at the other end, and a *Regular* instance is located somewhere in the middle of the axis.

If the Glyphs file has multiple axes, then multiple axis coordinates can be configured for each instance. Consider a Glyphs file with a *Weight* and *Width* axis. In that case, there may be *Thin*, *Thin Condensed*, *Thin Expanded*, *Regular Narrow*, *Semibold Condensed*, and *Bold Expanded* instances.

Glyphs can also *extrapolate* static instances. An extrapolated instance is located outside the coordinates defined by the masters, for example, an *Extra Bold* instance that is bolder than the *Bold* master. In practice, extrapolation is difficult to control, so most font projects only make use of interpolation.



Variable fonts are font files containing information about all masters and interpolation axes, allowing the font user to pick custom locations in the designspace.

For example, for a variable font containing a *Weight* axis (ranging 300–900) and a *Width* axis (50–140), a font user might pick ‘SomeFont Weight=600 Width=70’ instead of ‘SomeFont Semibold Narrow’. This method allows picking any configuration of weight and width without the font vendor preparing static instances for every possible combination. Variable fonts may also include static instances, offering a predefined set of axis configurations for convenient access.

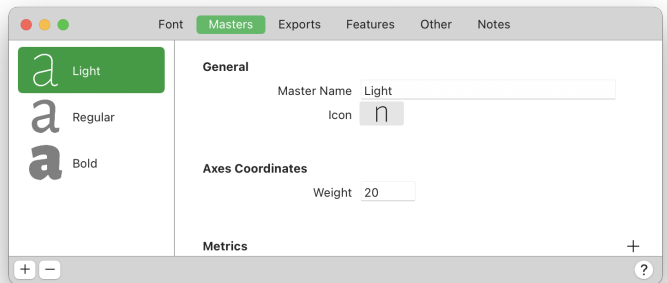
11.2 SETTING UP AXES

Define interpolation axes in *File > Font Info... > Font > Axes*. Click the plus \oplus button next to the *Axes* heading to add a new axis. An axis has a name, a four-character tag, and a hidden-checkbox.

Click the disclosure \checkmark button to pick one of the predefined axes. Choosing a predefined axis also sets its four-character tag field with the respective value. The predefined axes are a combination of the registered OpenType design-variation axes¹ and axes proposed for inclusion in the registry.² Otherwise, define a custom axis (also referred to as a *private axis*) by setting an arbitrary axis name and a custom four-letter tag. The tag of a private axis should use four capital letters (A–Z) as not to collide with future registered axes. For instance, a custom *Swash Length* axis might use the tag ‘SWLN’.

Check the *Hidden* checkbox to hide an axis in user interfaces. If set, this signals to applications using the font that they should not display controls (such as a slider) for this axis. Hide axes if they are applicable only in specialized software. Applications may or may not respect this option. In most cases, this checkbox should be left unchecked.

11.3 SETTING UP MASTERS



Interpolation requires at least two masters. Add masters in *File > Font Info... > Masters*. Click the plus \oplus button located in the bottom-left of the window to add a new master. See section 7.2.1, ‘Managing Masters’ (p. 94) for details.

Tip: Give each master a descriptive name such as *Light*, *Regular Condensed*, or *Bold Caption*, and pick a representative master icon (see p. 95).

- 1 docs.microsoft.com/typography/opentype/spec/dvaraxisreg
- 2 github.com/microsoft/OpenTypeDesignVariationAxisTags

11.3.1 Axes Coordinates

The *Axes Coordinates* of a master indicate its position in the designspace. Add masters and set their axis coordinates such that the designspace is covered by the masters. For a single *Weight* axis, two masters suffice:

Icon	Master Name	Weight
∩	Light	300
n	Bold	700

Adding a third master in the middle (for example, *Regular* at 400) allows for finer control over the interpolation.

The values for the *Axes Coordinates* fields can be chosen freely. For the *Weight* axis, the median vertical stem width can be used as axis coordinate values. For instance, a *Light* master with a stem width of 45 might use that as its *Weight* coordinate, while a *Bold* master uses its stem width of 160. A more abstract axis, such as *Serif*, might use a range from 0 to 100, with no serifs at 0 and long serifs at 100. Axes that do not smoothly transition, such as an *Italic* axis where the glyphs flip to different outlines, use a range from 0 to 1.

An *Italic* axis does not necessarily need to flip to different outlines. Instead, it may smoothly transition from an upright to an italic angle. Such an *Italic* axis may use a larger range like 0–100.

Add the ‘Axis Location’ custom parameter to use a different axis range for variable fonts. This is particularly relevant to registered axes such as *Weight* and *Optical Size*. See section 11.10.2, ‘Axis Location’ (p. 173) for details.

Axis coordinates are spread linearly in the designspace. For variable fonts, the ‘Axis Mappings’ custom parameter can be added for non-linear axes ranges. See section 11.10.3, ‘Axis Mappings’ (p. 173) for details.

11.3.2 Minimal Multiple Masters Setup

A minimal Multiple Masters setup requires one origin master and one master for each axis. The *Axes Coordinates* of the origin master differ from each other master on a single axis. For example, consider a font with a *Weight* and *Width* axis. Then, the following minimal master setup is possible:

Icon	Master Name	Weight	Width
∩	Light Condensed	300	50
∩	Light	300	80
n	Bold	700	80

The *Light* master is the origin master. Its coordinates differ from the *Light Condensed* only on the *Width* axis and from the *Bold*





only on the *Weight* axis. The *Bold*, for example, cannot be the origin master since its coordinates differ from the *Light Condensed* on both the *Weight* and *Width* axes.

See section 11.10.1, ‘Variable Font Origin’ (p. 172) for information on choosing the origin master. The choice of an origin master only matters to variable fonts; for static instances, Glyphs automatically determines the origin master.










11.3.3 Elaborate Multiple Masters Setups

While a minimal Multiple Master setup already covers the entire designspace, it offers little control over the design of instances that differ from the origin on multiple axes. For instance, the minimal setup example in the above subsection can produce a *Bold Condensed* instance, but its outlines might not be satisfactory.


A more elaborate Multiple Masters setup would include masters at all designspace corners:

Icon	Master Name	Weight	Width
	Light Condensed	300	50
	Light	300	80
	Bold Condensed	700	50
	Bold	700	80

Add intermediate masters as needed to fine-tune the design along the interpolation axes. A complex Multiple Masters setup can span across many masters:

Icon	Master Name	Weight	Width
	Light Condensed	300	50
	Light	300	80
	Light Extended	300	150
	Regular Condensed	400	50
	Regular	400	80
	Regular Extended	400	150
	Bold Condensed	700	50
	Bold	700	80
	Bold Extended	700	150

11.4 SETTING UP INSTANCES

In *File > Font Info... > Exports*, click the plus  button located in the bottom-left of the window to add a new instance. See section 7.3, ‘Exports’ (p. 98) for details on adding and configuring instances.

11.4.1 Static Instances

Static instances (⊕ > *Add Instance* and *Add Instance for each Master*) are exported as single font files. The *Axes Coordinates* of an instance need to be set up as described in section 7.3.4, ‘*Axes Coordinates*’ (p. 100). Note that the *Weight Class* and *Width Class* fields are independent of any axis coordinates; see section 7.3.3, ‘*Weight & Width*’ (p. 99).

Static instances are also included in variable fonts as a set of predefined axis configurations. This allows a font user to pick a predefined instance from a font style menu instead of configuring the variation axes manually. Instances that are outside the designspace (extrapolated) cannot be included in variable fonts.

Many custom parameters, including filters, are applied only to static instances when exported as single files, not when included in variable fonts. This is because, in a variable font, all instances need to be compatible, which means they have compatible outlines, the same glyphs, and the same features.

11.4.2 Variable Font Settings

A variable font setting (⊕ > *Add Variable Font Setting*) controls the export settings of a variable font. Adding multiple variable font settings will export multiple variable fonts with different configurations. Ensure that they have different names, so they do not conflict on export.

Since a variable font setting operates on the entire variable font, it can use the custom parameters that are not applied to its instances. These include filters, removing glyphs, and adding features. However, these custom parameters may still lead to incompatible outlines, which is reported as an error when exporting the variable fonts.

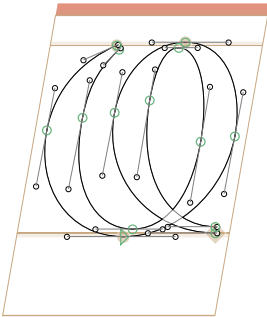
11.4.3 Subsetting the Designspace

Add a ‘*Disable Masters*’ custom parameter to a variable font setting to export a subset variable font. Write the names of font masters into the text field of the parameter, separated by a comma and a space (‘, ’). These masters will not contribute to the exported variable font. Axes which are not used by the remaining masters are removed from the variable font.


11.5 OUTLINE COMPATIBILITY

Glyph outlines need to be compatible for interpolation to work. Two glyph outlines are compatible when the following attributes are the same across all outlines:

- ▶ the number of paths and the order of their nodes;
- ▶ the number of anchors and their names;
- ▶ the number of components and their referenced glyphs;
- ▶ the order of paths and components (*Filter > Shape Order*).



11.5.1 Identifying Incompatible Outlines

If the outlines of a glyph are not compatible, a red bar is shown above its ascender line in Edit View and a red corner  in Font View. Filter for all incompatible glyphs using Smart Filters (see p. 86, Glyphs includes an *Incompatible masters* Smart Filter by default).

When using masters that are not compatible by design, such as color fonts, disable the red incompatibility indicators by adding the ‘Enforce Compatibility Check’ custom parameter in *File > Font Info... > Font* and unchecking it.

Exporting interpolated instances or variable fonts is not possible if there are glyphs with incompatible outlines.

11.5.2 Correcting Path Direction

Use *Path > Correct Path Direction* (Cmd-Shift-R) as a first measure to fix outline incompatibilities. This command does three things:

- ▶ it analyses the path structure and, if necessary, changes the path direction for each path;
- ▶ it normalizes the start node for each path, usually by choosing the node leftmost node at the bottom of the glyph layer;
- ▶ it reorders the shapes (paths and components), usually from bottom-left to top-right.

Hold down the Option key to apply the command on all master layers of the selected glyphs (*Path > Correct Path Direction for all Masters*, Cmd-Opt-Shift-R). See section 4.2.13, ‘Controlling Path Direction’ (p. 31) for more details.

11.5.3 Reordering Shapes

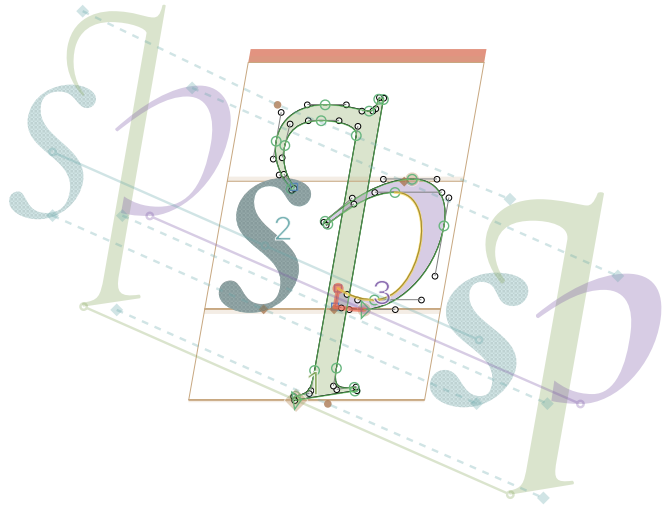
Choose *Filter > Shape Order* to show a window with the paths and components from all layers of the current glyph. Rearrange the shapes such that they are in the same order on all layers. See section 12.2.1, ‘Shape Order’ (p. 179) for details.

11.5.4 Master Compatibility

View > Show Master Compatibility (Cmd-Ctrl-Opt-N) shows the paths, components, and anchors across all masters.

Compatibility view for three incompatible masters: *Light*, *Regular*, and *Bold*.

In this case, a node is missing on the right of the counter of the p (shown in violet). This leaves a path segment with a larger curvature angle (highlighted in yellow) than on the other masters. Subsequent segments on the path are marked as incompatible (red).



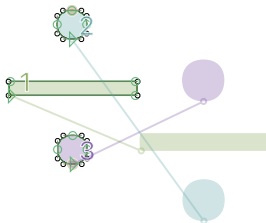
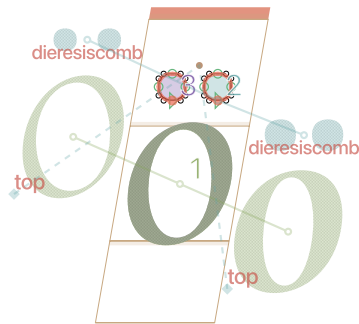
The following aspects are highlighted in this mode:

- ▶ Paths and components are colored based on their shape order. Components are additionally displayed with a checkered pattern.
- ▶ For the current layer, a shape order index number is written next to the starting node of each path and in the middle of each component.
- ▶ Diagonal lines connect starting nodes, shape centers, and anchors across the masters. Select points to show their connecting lines. Anchors are connected by dashed lines.
- ▶ Path segments are colored either green, yellow, or red. **Green** segments are compatible. **Yellow** segments are compatible, but their angle differs by more than 20° between masters, indicating that a node might be missing. **Red** segments are incompatible;

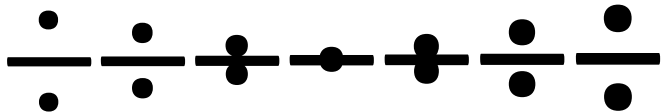
there is either a missing segment in other masters, or the segment types do not match (curved segment vs. straight segment).

Fix path direction and shape order related issues as described in section 11.5.2, 'Correcting Path Direction' (p. 162) and section 11.5.3, 'Reordering Shapes' (p. 163). The points on the connecting lines outside the current layer can be dragged to points of other connecting lines to swap to fix mismatched shapes. For path segment issues, pinpoint the incompatibility caused by selecting nodes and observing whether they fall on the expected spot in the other masters.

If a path, component, or anchor does not exist on a master, its connecting line will point to the origin at (0, 0) on that master. Glyphs will label components and anchors with red text in cases where these elements do not match:



The master layers of a glyph might be compatible but still produce undesirable interpolations. This can be the case when all shapes are in the same order but placed in different locations across the masters. The Master Compatibility view shows such shape-shifting glyphs with crossing diagonal lines. Interpolations between these masters might produce the following instances:



Fix shapershifters by reordering the shapes, either with *Path > Correct Path Direction*, *Filter > Shape Order*, or by dragging a point of a connecting line to the correct shape.

11.6 INTERMEDIATE LAYERS

Intermediate layers allow adjustments at a designspace location for a single glyph without adding another master. This helps fix interpolation issues that occur in a single glyph. For example, consider a two-master setup, *Thin-Black*, where the crossbar of the e appears too thin in the regular weights:

Interpolation between two masters: without Intermediate layer (above) and with Intermediate layer at the regular weight (below).



11.6.1 Intermediate Layer Setup

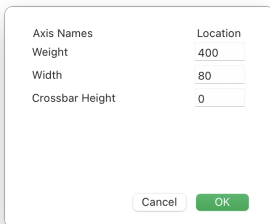
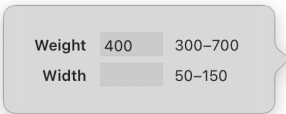
Firstly, select the master layer in the *Layers* palette most similar to the desired Intermediate layer. Then, click the plus $+$ button in the *Layers* palette to add a new layer. Control-click or right-click the new layer and choose *Intermediate* from the context menu. A number field for each axis will appear.

Enter the axis coordinates of the Intermediate, similar to how the *Axes Coordinates* work in *File > Font Info... > Masters*. Use the axis ranges displayed next to the fields for guidance. If a field is left empty, then the Intermediate layer uses the axis coordinate from the master layer to which it has been added. This is why adding an Intermediate to the most similar master layer is helpful. Confirm the entered values by pressing Return.

In the *Layers* palette, Intermediate layers are displayed as the comma-separated axis coordinates between curly braces (for example, **{90}** or **{80, 120}**). This is why Intermediate layers are also referred to as *brace layers*.

11.6.2 Virtual Masters

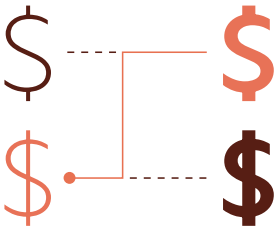
Intermediate layers allow for *virtual masters*. A virtual master works like a master in *File > Font Info... > Masters*, but it only affects a small number of glyphs. Other glyphs do not need to be redrawn, and kerning pairs do not need to be defined for virtual masters. For example, an axis might control the height of crossbars in glyphs such as A, E, F, and H. All other glyphs, including numbers, punctuation, and symbols, are unaffected by this axis, so it is a good candidate for a virtual master.



Add an axis as described in section 11.2, ‘Setting up Axes’ (p. 158). It can be one of the standard axes or a custom axis. The virtual master is defined using a ‘Virtual Master’ custom parameter in *File > Font Info... > Font*. Click the value of the custom parameter and set its axis coordinates like any other master. Confirm the dialog with *OK*. Note that coordinates for the new axis must be set for the existing masters in *File > Font Info... > Masters*.

With both the axis and the virtual master setup, add Intermediate layers to the glyphs that need to be adjusted for the virtual master. When exporting to variable fonts, only deltas for these glyphs are stored, which keeps the font file size small.

11.7 SWITCHING SHAPES



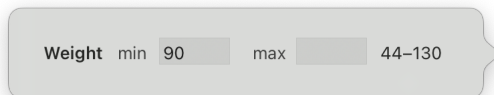
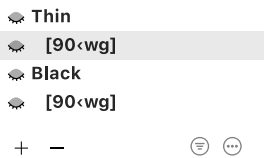
Some glyphs may not stay compatible as they interpolate. Examples include the dollar sign (\$) losing its stroke in the middle in bold weights or double-story forms switching to a single-story form at an italic angle (*a* → *a* and *g* → *g*). There are three methods by which glyph shapes can be switched:

- ▶ using Alternate layers to switch to different glyph outlines in both static instances and variable fonts;
- ▶ replacing glyphs with other glyphs at export;
- ▶ substituting glyphs by other glyphs for certain regions of the designspace in variable fonts.

11.7.1 Alternate Layers

An Alternate layer is a layer that contains an alternate glyph outline and information about where in the designspace the glyph should switch to the alternate outline.

Alternate layers are added to individual glyphs. For each master, select the master layer in the *Layers* palette, click the plus + button to add a new layer, Control-click or right-click the new layer and choose *Alternate* from the context menu.



After converting to an Alternate layer, number fields for the minimum and maximum coordinates will open. These describe the designspace region in which the alternate shape should be

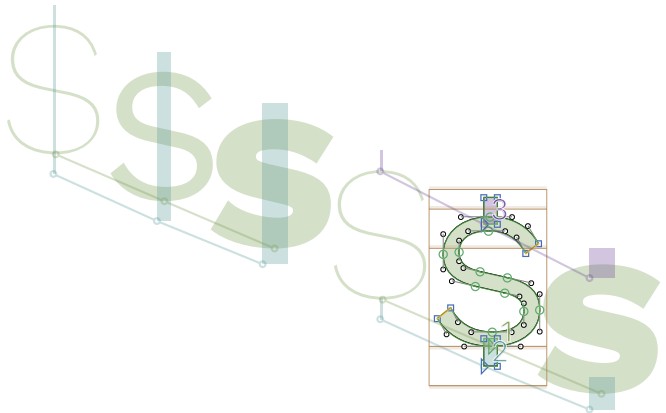
used. For example, consider a single *Weight* axis ranging from 44 to 130 where the Alternate layer should be used for a weight of 90 or greater. In that case, set the *min* value to 90 and leave the *max* field empty.

Confirm by pressing Return. The name of the Alternate layer is displayed in bold as comma-separated axis ranges between square brackets (for example, **[90<wg]** or **[80<wg, 25<wd<50]**). This is why Alternate layers are also referred to as *bracket layers*. Common axis names are abbreviated to two letters; the full axis tag is shown for all other axes.

Modify the outlines on the Alternate layers to fit the alternate design of the glyph. Alternate layers do not need to be compatible with the master layers, but they do need to be compatible with each other. The Master Compatibility view (see p. 163) also shows Alternate layers, but they are offset by a gap:

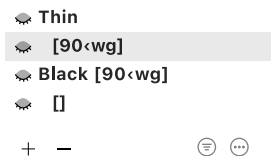
Axis Name	Tag	Abbr.
Italic	ital	it
Optical Size	opsz	oz
Slant	slnt	sl
Width	wdth	wd
Weight	wght	wg

Master Compatibility view for the dollar glyph with two masters (*Thin* and *Black*), two Intermediate layers, and three Alternate layers. The currently active glyph layer is both an Intermediate layer and an Alternate layer.



A layer can be both an Intermediate and Alternate layer. In that case, both the square bracket Alternate name (**[...]**) and the curly brace Intermediate name (**{...}**) are displayed in the *Layers* palette. Double-click either one to edit its settings.

A master layer can also be designated as an Alternate layer. For that, Control-click or right-click a master layer in the *Layers* palette and choose *Alternate*. Configure the axis range of the master layer like a normal Alternate layer. Then, add a new layer to the master, mark it as an Alternate layer, and leave both the *min* and *max* fields empty. This renames the new layer to '[']. Place the alternate outline on the master layer and the normal outline on the '[' layer. Whether the master or a backup layer is



converted to an Alternate layer is not relevant to the exported fonts; it is purely an organizational choice within Glyphs.

In variable fonts, Alternate layers are activated using a specific OpenType feature. Set the ‘Feature for Feature Variations’ custom parameter in *File > Font Info... > Font* to the four-letter feature tag. By default, Glyphs uses ‘rlig’, but ‘rvrn’ is also a common choice. Different features will be processed at different stages of the text shaping process; this depends on the operating system and application displaying the text.

11.7.2 Replacing Glyphs at Export



Glyphs can be replaced by other glyphs at export for both static instances and variable font settings. For this, two custom parameters must be added in *File > Font Info... > Exports*. Firstly, add the ‘Rename Glyphs’ custom parameter, click its value to edit it, and write one glyph swap per line. A glyph swap is written as `someglyph=otherglyph`. This switches the two glyphs in the exported font. For instance, switch the dollar glyph for a simplified dollar.alt by adding the line `dollar=dollar.alt`. List all glyphs with their replacement and confirm with *OK*.

Add the ‘Remove Glyphs’ custom parameter to not just swap, but replace glyphs. Click its value to edit it and list all alternative glyphs that now, after the swap, contain the normal glyph outlines.

11.7.3 Conditional Glyph Substitutions

Conditional substitutions allow glyphs to be replaced in variable fonts. See section 13.2, ‘Conditional Feature Code’ (p. 190) for details.



An example for the `dollar` → `dollar.alt` substitution from the examples above may look like this:

```
#ifdef VARIABLE
condition 80 < wght;
sub dollar by dollar.alt;
#endif
```





This replaces the `dollar` with the `dollar.alt` glyph for a weight of 80 or greater.

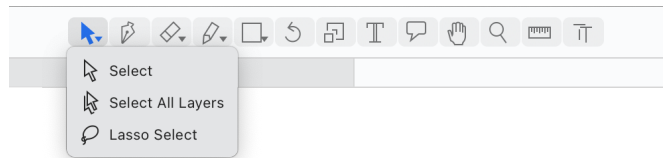
When exporting to both variable fonts and static instances, consider using the glyph replacement described in section 11.7.2, ‘Replacing Glyphs at Export’ (p. 168) to mimic the conditional substitutions in static instances.

11.8 EDITING MULTIPLE MASTERS

Font masters are listed in the *Layers* palette. Click the /  buttons to show a layer even if it is not currently active. See section 5.3, ‘Layers’ (p. 61) for details.

11.8.1 Select All Layers Tool

Use the Select All Layers tool  (shortcut Shift-V) to edit the paths of all visible layers at the same time. Click the /  buttons in the *Layers* palette to show or hide layers. Switch between the Select and Select All Layers tool by clicking and holding the Select  icon, then choose the desired tool from the menu:



11.8.2 Show All Masters

When in Edit View, choose *Edit > Show All Masters* to add all master layers of the current glyph to the Edit View tab. This command also inserts Intermediate and Alternate layers.

11.8.3 Keep Layer Selection in Sync

Select the *Edit > Keep Layer Selection in Sync* option to keep the current selection when changing masters. This option only works when working on a compatible glyph.

11.9 WORKING WITH MULTIPLE FONTS

11.9.1 Grouping Fonts into Families

A *font* typically refers to a single font file, like *Regular*, *Bold*, or *Semibold Italic*. In Glyphs, these font files are exported from the instances added in *File > Font Info... > Exports*. In most software applications, fonts are grouped by their family name, which is set in *File > Font Info... > Font > General > Family Name*.

Instances and variable font settings can overwrite the family name set in the *Font* tab with a ‘Localized Family Names’ custom parameter. Note that the *Default* language is used for grouping;

other localized names are only used to display the family name on screen.

Use style linking as described in section 7.3.5, ‘Style Linking’ (p. 100) to link the bold and italic styles. This enables the **Bold** and *Italic* buttons and the ⌘B and ⌘I keyboard shortcuts common in many applications.

A variable font contains a font family in a single file. While it is technically possible to put all family members into a single variable font, it may be desirable to split the family across multiple variable fonts. For example, an italic variable font may be sold separately from the regular variable font.

Add multiple variable font settings in *File > Font Info... > Exports* to export different variable fonts from a single Glyphs file (for example, a retail version and a trial version with a reduced glyph set). If the glyph set or OpenType features differ significantly, consider using multiple Glyphs files for the different variable fonts.

11.9.2 Glyphs Files, Masters, & Instances

Create a Glyphs file for each instance with *File > Generate Instances*. This command reads the instances of the currently open Glyphs file, converts them to font masters, and creates a new Glyphs file with each of those masters.

Convert a single instance to a master by selecting the instance in *File > Font Info... > Exports* and choosing *Instance as Master* from the plus (+) menu. The master will be added to the *Masters* tab. Copy a master from an open Glyphs file into the currently open file by choosing *Add Other Master* from the plus (+) menu in *File > Font Info... > Masters*.

11.9.3 Compare Fonts

Compare two font masters by choosing *Edit > Compare Fonts...* On the top of the *Compare Fonts* window are controls for picking two font files. Choose from all font files that are currently open in Glyphs. Below the file controls are pop-up buttons for choosing a master from the font file. The two selected masters are compared in the center of the window. Select the same font files twice to compare different masters of the same file.

The window lists the details of the two masters in a two-column layout, comparing font info, master metrics, glyph outlines, and kerning pairs. These categories can be collapsed or

expanded by clicking the disclosure ▼ triangle next to the gray headings. Click a row to select it. On the bottom of the window are two buttons: *Use Left* and *Use Right*. With a row selected, click one of these buttons to write the value from one side to the other. This can help fix inconsistencies between masters and Glyphs files.

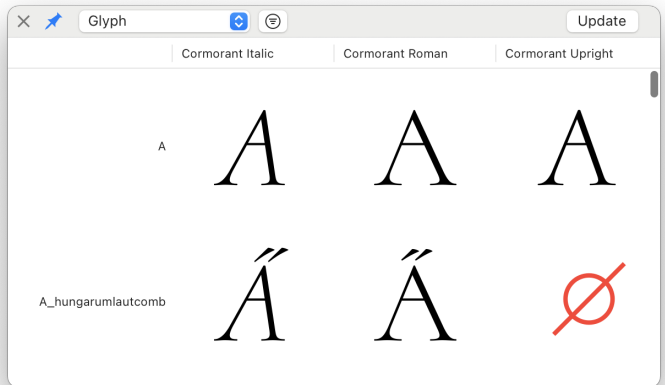
11.9.4 Compare Family


Open *Edit > Compare Family...* for a comparison of all open documents. Each document is represented by a column:


	Cormorant Italic	Cormorant Roman	Cormorant Upright
G_tildecomb	--	O	--
Germandbls	Germandbls	Germandbls	--
Ubar U		--	(missing glyph)
X X		X	--
Z --		Z	--
alpha-latin a		o	a
aogonek a		aaccent	a
d a		o	a
dcaron a		o	a
dcroat a		o	a

Pick from the pop-over menu located in the top-left of the window the attribute to compare. Choose from the following options: *Component Names*, *anchors*, *Left Kerning Group* and *Right Kerning Group*, and *Glyphs*.

Empty values are indicated with two dashes ('--'). If a glyph does not exist in one of the documents, '(missing glyph)' is shown. The *Glyphs* option shows a red ring with a stroke instead:



Double-click a cell to open the corresponding glyph in Edit View. Click the filter  button to filter the list of glyphs. *All* shows all glyphs with relevant information. *Missing in One* shows glyphs that are not present in all documents. *Ignore If Missing in One* shows only glyphs that are present in all documents.

Click the pin  button to keep the window above all Glyphs document windows. The *Update* button reloads all glyph information in case the documents were edited.

11.10 VARIABLE FONT OPTIONS

Additional adjustments can be made to variable fonts that affect their axes, fallback mode on legacy systems, and file size.

11.10.1 Variable Font Origin

Variable fonts require an *origin master*. The outlines of this master will be stored in the variable font file. All other masters are only stored as deltas from the origin master.

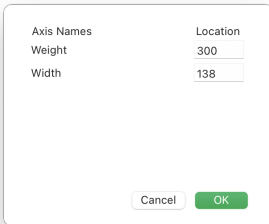
By default, Glyphs uses the first master in *File > Font Info... > Masters* as the origin master. Designate a different master as the origin by adding the ‘Variable Font Origin’ custom parameter in *File > Font Info... > Font*.

A minimal variable font setup requires at least an origin master and one master for each variation axis. This is described in detail in section 11.3.2, ‘Minimal Multiple Masters Setup’ (p. 159).

The origin master is used as a fallback on operating systems and applications that do not support variable fonts. Therefore, consider picking a regular master as the origin. Thereby, legacy

systems will show a regular font instead of a thin, bold, or italic font. Since the regular master tents to sit in the center of the designspace, it requires more deltas to describe the other masters. This increases the font file size compared to picking a corner master like *Light* or *Light Condensed*. For environments where file size is paramount (for example, webfonts), consider picking the origin master from a designspace corner.

11.10.2 Axis Location



When exporting a variable font, its design variation axes will use the same coordinates as the axis coordinates set in Glyphs. However, in some cases, the coordinates inside the Glyphs file might differ from the desired axis coordinates of the variable font. A common example is the *Weight* axis, which in Glyphs is often based on steam widths, but the OpenType specification recommends the *Weight* axis to use a range of 1–1000.³ Similar recommendations exist for the *Italic*,⁴ *Optical Size*,⁵ *Slant*,⁶ and *Width*⁷ axis.

Use different axis coordinates by adding an ‘Axis Location’ custom parameter to all masters in *File > Font Info... > Masters*. Click its value to edit it and assign each axis the coordinates that the master represents in variable fonts. Further configure the axis coordinates of variable fonts using axis mappings.

11.10.3 Axis Mappings



Three different axis mappings for the same weight axis.

For variable fonts, the axis coordinates that the user picks can be transformed to different axis coordinates by the font. Axis mappings perform these transformations.

By default, variable fonts use a linear axis mapping. *Linear* in this context means that each axis coordinate is mapped onto itself and is thus unchanged. For instance, consider a *Weight* axis ranging from 44 to 130. If a font user picks a weight of 95 on a slider, then that value is mapped to 95 (it is left unchanged) and used for interpolation:

3 docs.microsoft.com/typography/opentype/spec/dvaraxistag_wght

4 docs.microsoft.com/typography/opentype/spec/dvaraxistag_ital

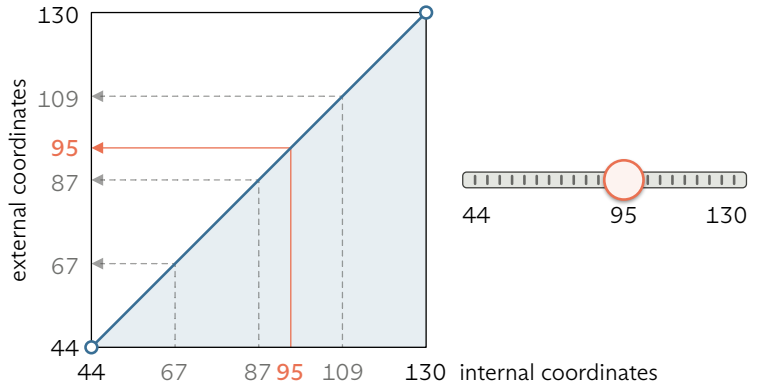
5 docs.microsoft.com/typography/opentype/spec/dvaraxistag_opsz

6 docs.microsoft.com/typography/opentype/spec/dvaraxistag_slnt

7 docs.microsoft.com/typography/opentype/spec/dvaraxistag_wdth

The linear axis mapping that is used by default.

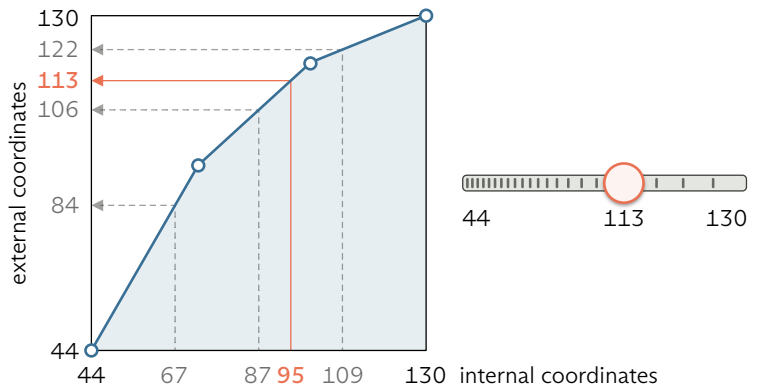
The slider is equally sensitive for the entire axis range. This means that changing the slider value by the same amount changes the interpolation value by the same amount everywhere along the axis.



Axis coordinates are mapped to other values by adding control points to the axis mapping that deviate from the linear diagonal:

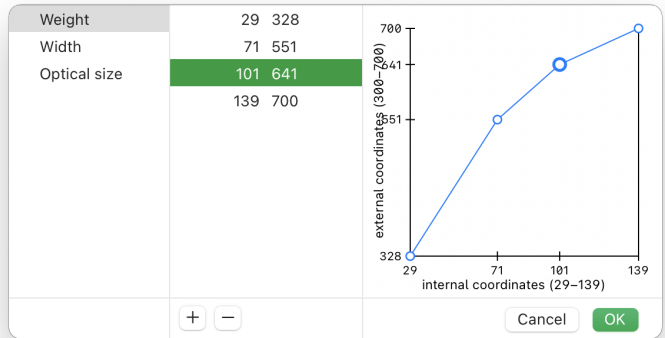
A non-linear axis mapping with two control points.

The slider is more sensitive on the lower end than on the higher end. This means that the same amount of change on the low end of the slider results in larger interpolation value changes than on the high end.



The *internal coordinates* are the values set in *File > Font Info... > Masters > Axes Coordinates*. The *external coordinates* are the result of axis mapping; they are used for interpolation.

Setup axis mapping by adding the 'Axis Mappings' custom parameter in *File > Font Info... > Font*. Click its value to edit the mappings:



The axis mappings dialog is split into three panes:

- ▶ the axis list;
- ▶ a list of the mapping control points of the selected axis;
- ▶ a visual editor for the control points of the selected axis.

The mappings list is divided into two columns: the internal coordinates on the left and the external coordinates on the right. Note that the external coordinates also have the ‘Axis Location’ custom parameter applied, if set, which is why the internal and external coordinates might differ even for a linear mapping.

Click the plus (+) button to add a new control point. The new point will show up in the mappings list and the visual editor. Alternatively, click on the blue line in the visual editor to add a new control point.

Click a value in the list to edit it or drag a control point up and down in the visual editor. Move the currently selected point with the arrow keys up/down and left/right. Hold down Shift for increments of 10 and Command for 100. The currently selected control point is highlighted in the list and enlarged in the visual editor. Delete the selected control point by clicking the minus (-) button.

Add an ‘Axis Mappings’ custom parameter to a variable font setting in *File > Font Info... > Exports* to customize the mapping for different variable font exports.

11.10.4 Style Attributes Table

The Style Attributes (STAT) table is information stored in a font file that gives names to different locations on the variation axes. This allows software using a variable font to display a style name

for every possible axis configuration.

For example, consider a variable font named ‘Example’ that has a *Weight* axis (200–700) and a *Width* axis (50–150). The STAT table allows to define names for certain axis locations:

<u>Weight</u>	<u>Name</u>	<u>Width</u>	<u>Name</u>
200	Thin	50	Compressed
300	Light	75	Condensed
400	Regular	100	Normal
500	Medium	125	Wide
600	Semibold	150	Expanded
700	Bold		

With this information, applications can construct a style name for arbitrary axis configurations. Setting *Weight* = 600, *Width* = 75 might yield ‘Example Semibold Condensed’. Names can be marked as *elidable*. An elidable name is removed when combined with other names. For example, the default ‘Normal’ width name may be marked as elidable so that the ‘Bold Normal’ style is simplified to ‘Bold’.

Glyphs automatically compiles all information required for the STAT table from the font instance names and their axis coordinates. If the STAT table is not as desired, customize it by adding the following two custom parameters in *File > Font Info... > Exports*.

- ▶ Add the ‘Style Name as STAT entry’ custom parameter to instances that differ from the variable font origin (see p. 172) on a single axis. Set the value to the four-letter tag of that axis. If, for example, the *Regular* master is the origin and the *Light* instance differs from it only on the *Weight* axis, add this parameter and set it to ‘wght’.
- ▶ Add the ‘Elidable STAT Axis Value Name’ custom parameter to instances that have an elidable name component on an axis. Set the value to the four-letter tag of the axis on which the instance name is elidable. For example, a *Regular* instance might need multiple parameters, one for each font axis, since its name is elidable in every case.

12 Filters

Filters process glyph layers. Their functionality ranges from simple width transformations to randomized distortions of glyph outlines. Glyphs includes a list of built-in filters.

Install additional filters from the Plugin Manager (see p. 235). These filters are also listed in the *Filter* menu and applicable as custom parameters.

12.1 APPLYING FILTERS

12.1.1 Filter Menu

Apply a filter by choosing *Filter > (name of the filter)*. The filter will be applied to all selected glyphs in Font View. In Edit View, the filter will be applied to a selection of glyphs made with the Text tool or to the current glyph with any other tool.

Filters usually affect only the currently visible layer. Applying a filter to all masters of a glyph may therefore require re-running the filter on all other masters. Quickly apply the last used filter by pressing Cmd-R.

12.1.2 Filters as Custom Parameters

Most filters can be applied to an instance on export using custom parameters. However, some filters cannot be used for variable fonts as they would produce incompatible outlines.

Add a custom parameter by navigating to *File > Font Info... > Exports > Custom Parameters*. Click the plus $+$ button, choose *Filter*, and click *Add*. A *Filter* parameter with a text field will be added to the list of custom parameters.

Write the name of the filter followed by its arguments into the text field. Arguments are separated by semicolons (;). Some argument values have a specific position in the arguments list, and some are named. Named arguments begin with their name and a colon (:) followed by their value.

```
FilterName; value1; value2; someName: value3
```

For example, the following line would apply the *RoundCorner* filter with a radius of 15 and visual corrections activated (1 for active, 0 for inactive):

```
RoundCorner; 15; 1
```

The order and meaning of arguments depend on the filter. See section 12.2, ‘Built-in Filters’ (p. 179) for details on the filters included in Glyphs.

Number value tokens (see p. 188) can be used to dynamically insert values by writing the name of a token prefixed by a dollar sign:

```
RoundCorner; $roundRadius; 1
```

When used in a variable font setting (see p. 161), number value tokens are used to interpolate the filter values.

Limit a filter to a subset of glyphs by adding an `include` argument that lists all glyph names for which the filter should be applied:

```
RoundCorner; 15; 1; include: a, b, c
```

Similarly, use the `exclude` argument to apply a filter to all glyphs except for the listed glyphs:

```
RoundCorner; 15; 1; exclude: a, b, c
```

Spaces in filter parameters are optional and may be added to improve legibility.

Multiple *Filter* parameters can be added to a single instance. They are applied during export in the order of the custom parameters. *Filter* parameters get applied after the components of a glyph are decomposed. Use a *PreFilter* custom parameter instead to run a filter before glyph decomposition. The filter rules in the custom parameter text field are the same for *PreFilter*. See section 10.1.10, ‘Decomposing’ (p. 140) for more on glyph decomposition.

Custom Parameters

<input checked="" type="checkbox"/>	PreFilter	Extrude; 100; -30
<input checked="" type="checkbox"/>	Filter	RoundCorner; 15; 1
<input checked="" type="checkbox"/>	Filter	Roughenizer; 15; 10; 10

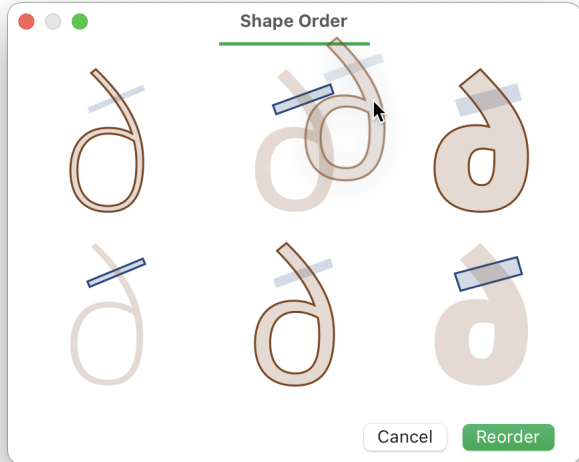
When applied from the *Filter* menu, many filters have an actions ☰ button in the lower left of their dialog window. Click the button and choose *Copy Custom Parameter*. The custom parameter for the filter can now be pasted into an instance. Open *File > Font Info... > Exports*, choose an instance from the sidebar, click the *Custom Parameters* heading, and paste the filter with *Edit > Paste* (Cmd-V).

12.2 BUILT-IN FILTERS

12.2.1 Shape Order

Shape Order dialog window:

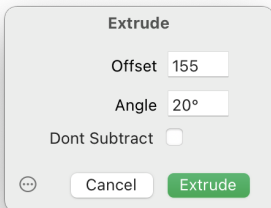
One column per master/
alternate/intermediate layer.
Drag to reorder shapes within
a column. The insertion
position is highlighted in blue.



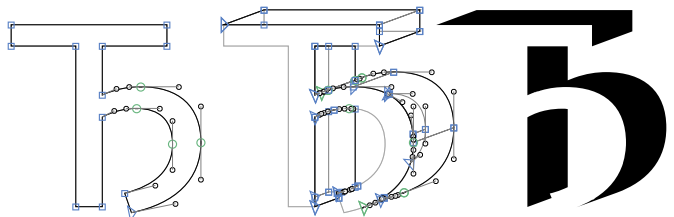
The *Shape Order* filter lists all shapes (paths and components) of a glyph across all layers. Paths are shown in navy blue; components in brown. Each column lists the shapes of a master/alternate/intermediate layer. Click and drag a shape to reorder it within its column. Confirm the chosen shape order with *OK*, or revert to the original shape order with *Cancel*.

Columns are separated by a gray gutter if there are multiple independent interpolations for the glyph. That might be the case when using intermediate layers (see p. 165) or when employing a complex master setup (for example, when condensed masters interpolate independently of extended masters).

The *Shape Order* filter is useful when *Path > Correct Path Directions for all Masters* (Cmd-Opt-Shift-R) does not yield the desired master compatibility.



12.2.2 Extrude



Filter > Extrude will create a solid shadow offset for the shape of the glyph. The following parameters are configurable:

Offset Controls the length of the shadow in font units.

Angle The direction of the extrusion in counterclockwise degrees from a right horizontal stretch.

Don't Subtract By default, the original shape will be subtracted from the extruded shape. Suppress such a subtraction by enabling this option.

From left to right:

- ① normal glyph
- ② standard extrude
- ③ composition of 1 and 2
- ④ *Don't subtract* enabled



The custom parameter rule is as follows:

```
Extrude; Offset; Angle; Don't Subtract
```

For example, to offset by 100 units at an angle of -30 degrees:

```
Extrude; 100; -30
```

Don't Subtract is disabled by default. Enable the option by setting its value to 1:

```
Extrude; 100; -30; 1
```

12.2.3 Hatch Outline



Filter > Hatch Outline creates hatched glyphs. The following parameters are configurable:

Origin Defines the origin point (X and Y coordinates) where the hatching pattern begins.

Step Width The distance between the strokes.

Angle The angle at which the strokes are drawn. 0° is horizontal, 90° is vertical.

Offset Path The thickness of the strokes. Disable this option to create open paths instead (which may be useful in combination with other filters or plug-ins).

The custom parameter rule is as follows:

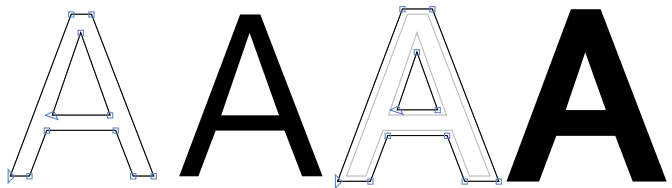
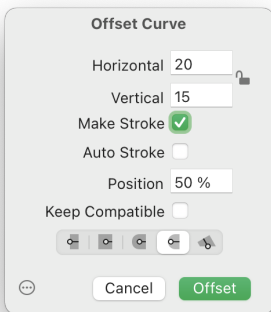
```
HatchOutlineFilter; OriginX:(X); OriginY:(Y);  
StepWidth:(Distance); Angle:(Angle); Offset:(Offset)
```

The arguments are all optional and can be written in any order. If left out, the argument of an option will assume its default value. For instance, for strokes, a thickness of 5 inclined at a 40° angle and placed at every 20 units:


```
HatchOutlineFilter; StepWidth:20; Angle:40; Offset:5
```

The hatch origin will default to $X = 0, Y = 0$.

12.2.4 Offset Curve



Filter > Offset Curve changes the thickness of stems horizontally and vertically. The following parameters are configurable:

Horizontal, Vertical Define the horizontal and vertical offset from the current outline. Positive values expand the outline; negative values contract the outline. Click the lock  button to use the horizontal field for both values.

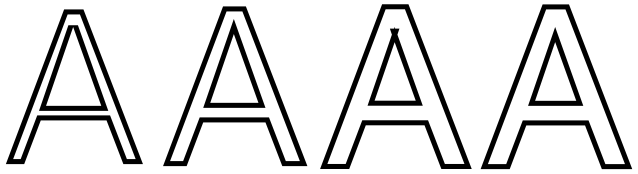
Make Stroke Offsets the outline in both directions to form a stroke along the outline. This is useful for creating outlines from closed paths. See section 4.3.1, ‘Creating Strokes’ (p. 34) for creating non-destructive strokes.

Auto Stroke Offsets the outline in both directions while maintaining the vertical dimensions intact when making a stroke. If enabled, it assumes a position of 50 %.






Position Controls the distribution of the stroke. 0 % places the stroke to the left of the path, 100 % places it to the right. At 50 %, the stroke is evenly distributed along the path.

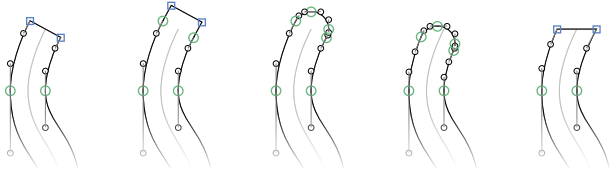
From left to right:

Make Stroke with *Position* of 100%, 50%, 0%, and 0% with *Keep Compatible* enabled.








Keep Compatible Keeps the offset curve compatible across masters by not introducing any new nodes or handles. Enabling this option might reduce the offset accuracy.

Cap Style Defines the stroke endings style when making a stroke from an open path. Choose from flat , square , round , round inset , and aligned to the vertical and horizontal axes .



The custom parameter rule is as follows:

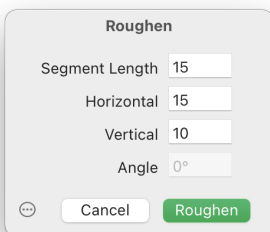
```
OffsetCurve; Horizontal; Vertical; Make Stroke;  
Position; Keep Compatible; cap: Cap Style
```

The **Horizontal** and **Vertical** offsets are in font units. **Make Stroke** is 0 for false or 1 for true. **Position** is either a percentage (for example, 0.5 for 50%) or 'auto' for the *Auto Stroke* option. Set **Keep Compatible** to 'keep' to enable the option or leave the argument off to disable it. **Cap Style** is 1 for , 2 for , 3 for , 4 for , and 0 or left off for .

```
OffsetCurve; 10; 10; 1; 0.5; cap:1
```

Curves can be offset non-destructively using stroke styles (see p. 34).

12.2.5 Roughen



Filter > Roughen segments an outline into straight line segments and randomly moves all resulting nodes within a given limit.

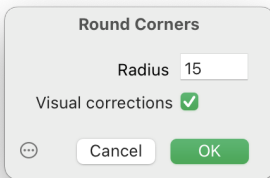
Control the size of the line segments with the *Segment Length* field. The *Horizontal* and *Vertical* values control the maximum offset for each node. The custom parameter rule is as follows:

```
Roughenizer; Length; Horizontal; Vertical
```

For example, the following rule would match the *Roughen* dialog window shown above:

```
Roughenizer; 15; 15; 10
```

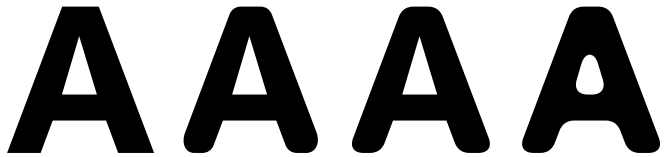
12.2.6 Round Corners



From left to right:

No filter, without visual correction, with visual correction, filter applied to outwards and inwards pointing corners.

Filter > Round Corners rounds all selected corners of a path. The filter is only applied to outwards pointing corners if nothing is selected. The *Radius* is defined in font units. Choose *Visual corrections* to balance the rounding of acute and obtuse corners using smaller and larger radii, respectively. Such visual corrections tend to create more natural-looking rounding.



The custom parameter rule is as follows:

```
RoundCorner; Radius; Visual Corrections
```

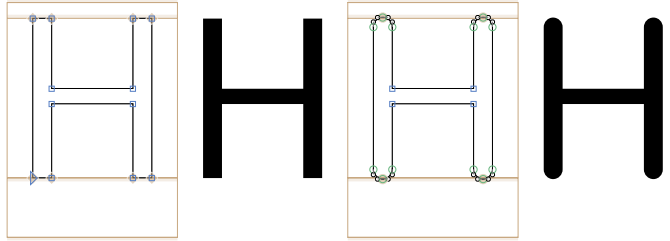
If *Radius* is positive, it is applied to outwards pointing corners; if it is negative, it is applied to inwards pointing corners. Add multiple *RoundCorner* custom parameter filters to control both. Set *Visual Corrections* to *1* to enable it or *0* to disable it. For example, the following rule settings apply a corner radius of 55 to outwards pointing corners with visual corrections enabled:

```
RoundCorner; 55; 1
```

When used on a variable font setting (see p. 161), add the keyword *compatible* at the end to improve the compatibility of the resulting rounded outlines, like so:

```
RoundCorner; 55; 1; compatible
```

12.2.7 Rounded Font



Filter > Rounded Font rounds the stem endings with appropriate overshoots. For this, it uses the vertical stem metrics as defined in *File > Font Info... > Masters > Stems*. The custom parameter rule is as follows:

```
RoundedFont; Vertical Stem Width
```

If a [Vertical Stem Width](#) argument is set, its value is used instead of the vertical stem width from the master metrics. Such an override can be helpful when combined with the `include` and `exclude` arguments to define a different stem width for a subset of glyphs. For example, consider three *Filter* custom parameters for an instance:

```
RoundedFont; exclude: f, k, t, dollar, percent  
RoundedFont; 74; include: f, k, t  
RoundedFont; 86; include: dollar, percent
```

Here, the glyphs f, k, t, dollar (\$), and percent (%) would be processed with special [Vertical Stem Width](#) values. See section 12.1.2, ‘Filters as Custom Parameters’, p. 177 for details on `include` and `exclude`.

12.2.8 Transformations

The Transformations filter is split into three dialogs: *Transform Metrics*, *Transformations*, and *Interpolate with Background*.

Transform Metrics *Glyph > Transform Metrics* assigns either a new width or new sidebearings to the selected glyph layers.

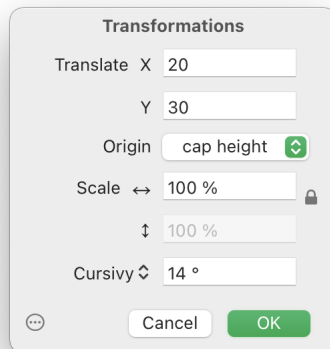


Enable *Width* to assign a new width to all selected glyphs. Check *on both sides* to modify both sides equally to match the new width. Otherwise, the points on the layer keep their original coordinates.

Uncheck *Width* to set new sidebearings. Control whether the left sidebearing (*LSB*), the right sidebearing (*RSB*), or both should be changed using the checkboxes next to their fields. If *Relative* is selected, the *LSB* and *RSB* values are added to the current sidebearings (or subtracted if the entered values are negative). Otherwise, the *LSB* and *RSB* values overwrite the existing sidebearings.

This type of transformation is also known as an *affine transformation*.

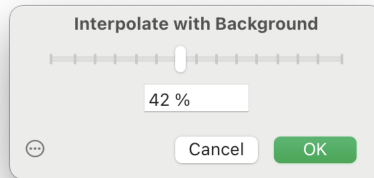
Transformations *Path > Transformations* applies a linear transformation to the selected points or selected layers.



Translate moves points by a given X and Y value. *Origin* defines the point from which the transformation originates. *Scale* scales the selection along the ↔ horizontal and ↕ vertical axis. If the lock button is locked 🔒, the horizontal value is used for both axes.

Slant skews the selection without optical correction. Click the *Slant* label and choose *Cursivy* for an optically corrected skew. Note that *Cursivy* requires horizontal and vertical metrics to be set in *File > Font Info... > Masters > Stems*.

Interpolate with Background *Path > Interpolate with Background* results in an interpolation of the foreground and background layers.



A value of 0 % leaves the foreground unchanged; 100 % replaces the foreground with the background, and values in-between result in an interpolated shape. Use a value below 0 % or above 100 % to extrapolate. Note that this filter does not work if the foreground and background are not compatible. For more details on interpolation, see chapter 11, 'Interpolation' (p. 156).

Custom Parameter The custom parameter rule is as follows:

```
Transformations; LSB: LSB, RSB: RSB; Width: Width;
ScaleX: Scale X; ScaleY: Scale Y; Slant: Slant;
SlantCorrection: Cursivy; OffsetX: X; OffsetY: Y;
Origin: Origin Metric
```

LSB and **RSB** set the left and right sidebearings. Prefix these values with a + (plus), - (minus), * (asterisk, multiply), or / (forward slash, divide) symbol to change the sidebearings relative to their current values. **Width** can be prefixed by + or -. For example, to multiply the LSB by 2 and add 15 to the RSB, use the following filter rule:

```
Transformations; LSB:*2; RSB:+15
```

Scale X and **Scale Y** are percentages where a value of 100 does not scale, < 100 scales down, and > 100 scales up. Set **Cursivy** to 0 to disable slant correction; it is 1 by default. Set **Origin Metric** to 0 for cap height, 1 for ½ cap height, 2 for x-height, 3 for ½ x-height (default), or 4 for baseline.

For example, the following rule sets the width of all glyphs to 700, scales the glyph outlines to 120 % (both horizontally and vertically), cursivies the outlines to a 14° angle (adding `SlantCorrection:0` would use normal slanting), and moves the outlines by 8 units to the right and 22 units down such that all these transformations originate from the x-height:

```
Transformations; Width:700; ScaleX:120; ScaleY:120;
  Slant:14; OffsetX:8; OffsetY:-22; Origin:2
```

12.2.9 Add Extremes

Path > Add Extremes adds missing nodes on extrema. See section 4.2.14, ‘Extremes & Inflections’ (p. 32) for more details. The custom parameter rule is the filter name `AddExtremes` without any arguments.

12.2.10 Remove Overlap

Path > Remove Overlap removes the overlap of the selected paths, or all paths if none are selected, or multiple glyphs are selected. It also clears the selected glyphs of all open paths and stray nodes. The filter expects all outline orientations to be set correctly (see section 4.2.13, ‘Controlling Path Direction’, p. 31).

Apply this filter as a custom parameter with the `RemoveOverlap` name and no arguments. Alternatively, apply *Remove Overlap* by checking its checkbox in the export dialog (see p. 111).

13 Feature Code

Glyphs extends the AFDKO feature syntax with a number of additional capabilities.

13.1 TOKENS

Tokens are pieces of code that help automate the feature code. They dynamically insert values and glyph names into feature code, and their syntax starts with a dollar sign (\$). There are three types of tokens: *number values*, *glyph properties*, and *glyph predicates*.

13.1.1 Number Value Tokens

A number value token inserts the values defined in *File > Font Info... > Masters > Number Values* (see p. 98). Number value tokens are written with a dollar sign followed by the number value name.

For example, the token `$padding` inserts the ‘padding’ number value on the master of the font. The value inserted by the token is interpolated if multiple masters exist.

Tokens also support basic arithmetic. Wrap the name of the number value in curly braces to use plus +, minus -, times *, and divide / operators: `${padding*2}`. The name of the number value must be placed at the start of the arithmetic expression. Such operations can be helpful in, for example, the capital spacing ‘csp’ feature:

```
pos @Uppercase <$padding 0 ${padding*2} 0>;
```

Hold down the Option key and click a number value token to show its computed value.

13.1.2 Glyph Property Tokens

A glyph property token is written as `glyphName:property` where `glyphName` is the name of a glyph and `property` is a dot-separated property path.

Properties can be metrics (`width`, `LSB`, `RSB`, `TSB`, `BSB`) and anchor positions (such as `anchors.top.x`). Basic arithmetic is supported as in number value tokens (see p. 188). For example:

```
# glyph positioning such as in the 'csp' feature
pos a.alt <${period:LSB} 0 ${period:LSB*2} 0>;
# custom mark to base positioning
```

```
pos base b <anchor
  ${b:anchors.top_special.x}
  ${b:anchors.top_special.y}> mark @SpecialTopMark;
```

Option-click a glyph property token to show its value.

13.1.3 Glyph Predicate Tokens

Glyph predicate tokens insert a space-separated list of glyph names. The predicate determines the glyphs matched and inserted. For example, `$_script == "adlam"` expands to a list of all glyphs belonging to the Adlam script, and `$_name endswith ".star"` expands to a list of all glyphs ending with a ‘.star’ suffix.

When used in a feature class, predicate tokens can be written among other glyph names, like this:

```
period comma $_category == "Symbol"
```

would expand to:

```
period comma at ampersand plus minus ...
```

Within prefixes and features, the predicate needs to be wrapped in square brackets, like this:

```
@Signs = [period comma $_category == "Symbol"];
@SmallCaps = [$_case == smallCaps];
```

The predicate can check for different aspects of a glyph:

True/False Check whether a boolean true or false condition applies. For example, `$_isAligned == true`, `$_hasHints == false`, or `$_isColorGlyph == true`.

Numbers Check for numeric values of the glyph. For example, `$_countOfLayers == 3` or `$_countOfUnicode > 0`. Numbers are also used for glyph properties with a limited set of values. For example, `$_colorIndex == 5` (see section 6.3.5, ‘Color Label’, p. 72 for color numbers), `$_case == upper` (use ‘noCase’, ‘upper’, ‘lower’, ‘smallCaps’, ‘minor’, or ‘other’ for comparison), or `$_direction == 2` (0: LTR, 1: BiDi, 2: RTL).

Strings Match a text string with a glyph property. For example, `$_name endswith ".sc"`, `$_script == "balinese"`, or `$_category == "Letter"`.

Objects Access nested glyph data for comparison. For example, `[$[layer.anchors.top.y > 600], $[layer.maxX > 300], or $["ipa" in tags]`.

Predicates can also be combined. Write **AND** between two predicates if both need to be true. Write **OR** if only one of the two needs to be true for the glyph to be included. Write **NOT** before a predicate to invert it; that is, include all glyphs not matching the predicate. If needed, use parentheses to group predicates. For example:

```
$[script == "greek" AND case == upper]
$["todo" in tags OR NOT note contains "done"]
$["ipa" in tags OR ("lang" in tags AND case == lower)]
```

Glyph predicate tokens also support a special `class(...)` function that represents all glyphs contained in a feature class (`@SomeClass`). Use it to check whether a glyph is in a given class, like this:

```
# all lowercase glyphs of the class @SomeClass
$name in class(SomeClass) AND case == lower]
# the intersection of two classes
$name in class(Narrow) AND name in class(TopMark)]
```

Option-click a glyph predicate token to show a list of all matching glyphs.

13.2 CONDITIONAL FEATURE CODE

Feature code can be wrapped in `#ifdef` blocks to limit it to variable fonts. Such a code block ends with a `#endif` line:

```
#ifdef VARIABLE
sub a by a.var;
#endif
```

Rules inside a `#ifdef` (if defined) block are included only in variable fonts. Use `#ifndef` (if not defined) to include feature rules only in non-variable fonts:

```
#ifndef VARIABLE
sub a by a.static;
#endif
```

Variable-only blocks may contain OpenType feature variations. These restrict substitution and positioning rules to a specific

region of the designspace. Define a feature variation by writing the **condition** keyword followed by the axis ranges to which the code following the condition should be limited:

```
#ifdef VARIABLE
condition 600 < wght < 900;
sub dollar by dollar.alt;
#endif
```

Axis ranges are written as the axis tag bounded by a lower and upper limit. Write a less-than sign (<) between the limit value and the axis tag. Note that, although the less-than sign is written for both the lower and upper limit, the lower limit actually behaves as if a less-than-or-equal sign (\leq) was used. An axis range can specify both a lower and upper limit, or only one of the two.

A condition can be restricted to multiple axis ranges. Separate them with a comma. The code following the condition is used only if all specified axis ranges match.

```
#ifdef VARIABLE
condition 600 < wght < 900, 70 < width < 90;
sub won by won.boldcondensed;
#endif
```

13.3 VARIABLE GPOS

Position (**pos**) rules can be contextualized to specific regions of the designspace. Write axis tags and their internal coordinates in parentheses, followed by the value to be used at that location of the designspace. For example, in the ‘csp’ feature:

```
pos @Uppercase 10; # static
pos @Uppercase 10 (width:80) 20; # variable
```

There can be multiple alternative values and axes:

```
pos @Uppercase 10 (width:80) 20 (width:40 opsz:28) 30;
```

The same works for the four-value syntax:

```
pos @Digit colon' <10 50 20 0
      (width:80) 30 40 60 0
      (width:40 opsz:28) 5 10 10 0> @Digit;
```

In a variable font, the position value interpolates smoothly along the specified axes.

13.4 DELETING GLYPHS

Glyphs may be removed with a feature using the **deLeTe** keyword (or its abbreviation: **deL**). Remove all occurrences of a glyph by writing its name after the **deLeTe** keyword:

```
deLeTe tonos;
```

Glyphs may also be deleted contextually. Mark the glyph to be deleted by adding a single quote (') after its name. Define the context by writing glyph names and classes before and after the marked glyph. In the following example, tonos is only deleted when followed by Alpha:

```
deLeTe tonos' Alpha;
```

The **deLeTe** keyword is not supported by standard AFDKO-based tools. Alternatively, substitute a glyph by the special keyword **NULL** to delete it. This method also works when used with AFDKO tools outside of Glyphs.

```
sub tonos by NULL;  
sub tonos' Alpha by NULL;
```

13.5 STANDALONE LOOKUPS

Lookups written inside a **feature** block are applied when the feature is enabled by the typesetting system. Commonly, this is the intended behavior.

Some lookups, however, should not be applied directly by the feature. Instead, they are used as part of a different lookup, for example via a chaining substitution. In such cases, the definition lookup needs to be moved outside of the **feature** block. In Glyphs, this is done by writing the lookup definition in a *Prefix* section. However, since the *Prefix* code is placed before the *Features* code, and lookups are applied in the order in which they are written, such prefix lookups might get applied too early.

Resolve this issue by writing the **standalone** keyword before the **lookup** keyword when defining a lookup inside a feature. This will apply the lookup only when used from other lookups, while keeping the order of the lookup unchanged.

For example, the following is a stand-alone lookup being used as part of a chaining substitution inside the 'ccmp' feature:

```
standalone lookup demo {  
  sub gravecomb by gravecomb.case;
```



```
    sub acutecomb by acutecomb.case;
} demo;
```

```
sub @Uppercase @CombiningTopAccents' Lookup demo;
```

For lookups defined outside of a feature, the **standalone** keyword is redundant and will be ignored.

13.6 OPTIONAL CLOSING LOOKUP NAME

The standard feature syntax requires the name of a lookup to be repeated at the end of the lookup definition:

```
Lookup some_descriptive_lookup_name {
    # lookup code
} some_descriptive_lookup_name;
```

In Glyphs, the lookup name at the end may be omitted:

```
Lookup some_descriptive_lookup_name {
    # lookup code
};
```

13.7 MULTIPLE LANGUAGES SYNTAX

In Glyphs, the **language** keyword may be followed by more than one language tag:

```
language ENG DEU FRA;
```

Without this syntax extension, rules that do not apply to the default language need to be repeated for every language for which they apply. For example:

```
script latn;
language AZE;
    sub i by idotaccent;
language CRT;
    sub i by idotaccent;
language KAZ;
    sub i by idotaccent;
language TAT;
    sub i by idotaccent;
language TRK;
    sub i by idotaccent;
```

The above code can be shortened to the following:

```
script latn;  
language AZE CRT KAZ TAT TRK;  
sub i by idotaccent;
```

13.8 MULTIPLE SUBSTITUTION WITH CLASSES

Glyphs allows multiple substitution (one-to-many) to use classes instead of just single glyphs. This works for both simple and contextual rules:

```
sub @Symbol by @SymbolLeft @SymbolRight;  
sub @Left @Symbol' @Right by @SymbolLeft @SymbolRight;
```

The class that gets substituted (`@Symbol` in the example above) needs to have the same number of glyphs as the classes that it gets substituted by (`@SymbolLeft`, `@SymbolRight`).

If there is a single glyph or a class containing a single glyph on either side of the substitution rule, that glyph is repeated to match the count of the other classes in the substitution.

13.9 LOOKUP FLAGS

The `lookupflag` keyword resets any previous lookup flags and sets a new set of flags. For example, to reset the previous flags and ignore marks in the current lookup, the following code line can be placed at the start of a lookup definition:

```
lookupflag IgnoreMarks;
```

When writing lookups, it is common to ignore any prior flags without setting new ones. In the standard feature syntax this is done by using the special flag `0`:

```
lookupflag 0;
```

As an alternative, Glyphs allows resetting lookup flags without setting new flags by simply defining an empty `lookupflag` statement:

```
lookupflag;
```

14 PostScript Hinting

PostScript hinting is a method to improve display at low resolutions for fonts with PostScript/CFF outlines. TrueType flavor OpenType fonts use a different method for hinting; see chapter 15, ‘TrueType Hinting’ (p. 204) for details.

The eventual picture on the screen is created by a software called the rasterizer. Hints help the rasterizer to create a more even glyph image. Especially stems are harmonized to look similar across a line of text. PostScript hints are simpler but also less flexible than TrueType hints.

Most hinting information revolves around determining which part of a letter is a necessary stroke element and should not be omitted at small sizes. There are two kinds of hints.

Font-level hints or *font-wide hints* store general information that applies to the entire font and encompasses standard stems and alignment zones.

Glyph-level hints are little pieces of information placed inside a glyph that help the rasterizer stretch the outline across the pixel grid. They can either be *stem hints* or *ghost hints*.

The best practice is choosing good font-level hints and letting an algorithm called the *autohinter* find the glyph-level hints.

Hinting only makes sense if the font has repeated regular features. If the font is very irregular, like many handwritten fonts are, or like ornamental and grunge fonts, then hinting cannot help to improve the rendering. Also, suppose a font is intended for exclusive use in environments where hinting information is ignored, like displays with a very high resolution, or on Apple hardware running macOS or iOS. In that case, the hinting information is not used and will only make the font file larger. Consider not hinting or disabling any existing hinting for such projects.

Note that PostScript hinting intends to create a sharper, more consistent pixel image at low resolutions. That means that the outline will be distorted to achieve a better fitting on the pixel grid. In other words, hinting *does not preserve shapes*; on the contrary. Hinting does not make sense for fonts where the preservation of the shape is more important than a crisp pixel image, such as in connecting script typefaces and icon fonts.

14.1 FONT-WIDE HINTS

Before adding glyph-level hinting, define a set of parameters that apply to all hinting throughout the font. These font-level hints are stored in the so-called *PostScript Private Dictionary* inside the exported font. For an in-depth discussion, see

- ▶ the Adobe Type 1 Font Format specification¹;
- ▶ Robothon 2012: Postscript hints,² a video presentation about PostScript hinting by Miguel Sousa from Adobe.

14.1.1 Standard Stems

Stem widths are the thicknesses of letter strokes. A *vertical* stem is the width of a vertical stroke of a letter, for example, the thickness of the I, or the thicknesses of left and right curves of an O. A *horizontal* stem is the thickness of a horizontal stroke movement, for example, the serifs or crossbars of A and H, or t and f, or the upper and lower curves of an O.

Standard stems are average values, as representative as possible, for as many stem widths in the font as possible. The autohinter needs good standard stem values to recognize the stems and insert glyph-level hints automatically. And the screen rasterizer can make use of these values to optimize the pixel rendering, especially synchronizing stem thicknesses across the whole font at low resolutions.

Try to find as few as possible and as representative as possible values for horizontal and vertical stem widths and enter them in the *Masters* tab of Font Info (*File > Font Info... > Masters*, Cmd-I). See section 7.2.5, ‘Stems’ (p. 97) for details on editing stem values in Font Info.

If two values are close to each other, consider merging them into one average value. Quickly measure the thickness of stems by selecting two nodes and looking at the Info box (Cmd-Shift-I) or by switching to the Measurement tool. See section 4.10, ‘Measuring’ (p. 48).

For instance, if the measured values are 68, 71, 72, 74, 75, 82, 83, and 85 for the vertical stems, pick 75 or 80 for the standard vertical stem because either would be a good median value for most of the stem measures. By using a single stem value, the stems will scale more uniformly across low PPMs.

PPM stands for pixel per em and is a measurement for pixel density.

- 1 adobe-type-tools.github.io/font-tech-notes/pdfs/T1_SPEC.pdf, specifically pages 35–45
- 2 vimeo.com/38364880, approximately 35 min

Theoretically, up to twelve stem width values can be considered for each orientation. But the best practice of trying to find as few as possible will typically either result in a single representative value for all stems or in two values: one for lowercase and one for uppercase letters, or (in the case of horizontal stems) one for an average horizontal stroke, and one for the serifs. Use a second or third value only if it is acceptable that the associated stems will have different thicknesses at the same pixel size. For instance, for a vertical standard stem set at 70 units and another at 80 units, the first stem may be displayed two pixels wide, while the other stem may get three pixels at the same pixel size.

The first horizontal and vertical stem values are the most important ones. Use a value that represents the most-used glyphs, typically the lowercase letters. Other functions in Glyphs also use these values, such as the Cursivy algorithm or the *Rounded Font* filter. Any stem values that follow are exclusively used for hinting. The horizontal stems also play a role in TrueType hinting (see p. 204).

When interpolating between masters, stems with the same name in both masters are used to interpolate the in-between stem value.

14.1.2 Alignment Zones

When a font is rendered with very few pixels on a computer screen, all the x-heights should use the same amount of pixels vertically. The same applies to ascenders of letters like f, h, or k, and to descenders of g, p or y, and to the heights of all capital letters. For many designs, all letters should share the same baseline when rasterized at a low resolution.

But all these letters usually do not align precisely. For instance, the bottom of a lowercase o will extend slightly below the baseline, while the serifs of an n may sit precisely on it. Or the apex of an uppercase A may extend a little bit beyond the height of an uppercase H. This difference, usually some ten to fifteen units, is commonly referred to as *overshoot*.

Alignment zones are a way to tell the rasterizer about the overshoots. Overshoots cannot help to provide an optically balanced text at small pixel sizes, so their display should be suppressed. More precisely, at low resolutions, any path constellation with a horizontal stem or ghost hint attached to it

that reaches into an alignment zone will be vertically aligned to the base of the zone.

Alignment zones take two values: a position and a size. The position is the vertical height of the zone, usually the vertical metrics, like x-height or ascender. The position is sometimes also referred to as the *flat edge* of a zone. The size is the thickness of the maximum overshoot that may appear at that position. If the overshoot extends above the position (x-height, small caps height, cap height, ascender), the size value must be positive. Such zones are referred to as *top zones*. If, however, the overshoots extend below the position (typically for the baseline or descender), the size must be negative and is referred to as a *bottom zone*. See section 7.2.4, ‘Metrics & Alignment Zones’ (p. 95) for details on editing the alignment zones of a font master.

A typical alignment zone setup: top zones with positive sizes at ascender, cap height and x-height; bottom zones with negative sizes at baseline and descender.



Alignment zones should be as small as possible, so do not try to make them larger ‘to be on the safe side’. More precisely, the maximum size of an alignment zone is constrained by the `blueScale` value (see below), which implies that no zone must be larger than $240 \div (240 \times \text{blueScale} - 0.98)$. In any event, a zone must not be larger than 25 units. There may be a maximum of 6 top zones, 5 bottom zones, and the baseline zone. Zones must not overlap. There must be a minimum distance of one unit between them; the larger, the better. The baseline zone must have a position value of zero.

If the font uses an alternative grid (see section 7.5.1, ‘Grid Spacing & Subdivision’, p. 107), extend the scope of the zones by one unit in both directions to catch potential small rounding errors for vertical node positions. That is, the position must be shifted by one unit and the size by two units. Only the baseline zone must be kept at position zero while its size is increased by one unit.

14.1.3 Custom Parameters

Apart from the alignment zones and standard stems, there are more optional parameters in the Private Dictionary: ‘blueFuzz’, ‘blueScale’, ‘blueShift’, and ‘Family Alignment Zones’. In Glyphs, set these values as custom parameters. See the description in Glyphs when adding the custom parameters for more details.

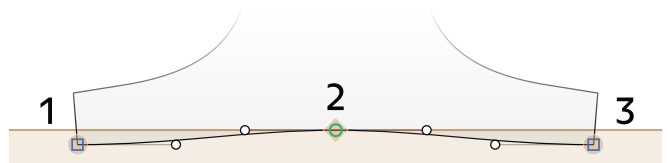
14.2 AUTOHINTING

If the font-wide parameters (alignment zones and standard stems) are correctly set, activate the autohinter by checking the *Autohint* option in the *File > Export* dialog. Enforce this setting with the *Autohint* custom parameter.

Test the hinting in an Adobe application (see section 4.13.6, ‘Previewing in Adobe Applications’, p. 58). Write a test text and zoom out far enough to display the letters with a few pixels only. Then zoom in using the operating system’s *Zoom* function (configurable in the *Accessibility* settings of the System Settings). If necessary, tweak the font settings or manually hint a problematic glyph and re-export. For details on manual hinting, see section 14.3, ‘Manual hinting’ (p. 200).

14.2.1 Flex Hints

Flex hints: Nodes 1 and 3 are on the same level and inside the alignment zone, node 2 should be exactly on the flat edge of the zone. The handles must stay inside the space defined by nodes 1 through 3.



If the font has cupped serifs or slightly tapered stems, the autohinter can automatically apply so-called flex hints. Flex hints suppress the display of such shallow curves at low resolutions. They cannot be set manually but are automatically applied when the font is exported. For flex hinting to kick in, a few conditions must be met.

First, the ‘blueShift’ value must at least be set to the depth of the cups plus one. For example, if the serifs are cupped 5 units deep, ‘blueShift’ should be set to 6 or more. Set ‘blueShift’ as a custom parameter in *File > Font Info... > Font* (Cmd-I).

Secondly, there are a few outline requirements. The cup or tapering must be built from exactly two consecutive outline segments between three nodes. The segments do not need to be

symmetrical. The first and third nodes must share the same X coordinate (for tapered stems) or the same Y coordinate (for cupped serifs). The four handles need not be entirely horizontal (serifs) or vertical (stems), but the three nodes must be placed on the extremes of the two segments. The overall depth must not exceed 20 units.

Thirdly, in the case of cupped serifs, it is recommended that the three points are completely submerged in the respective alignment zone. For best results, the second node (in the middle) should be precisely on the flat edge of the zone. And the other two nodes must reach into the zone. This means that cupped bottom serifs reach a little bit below the baseline and into its bottom zone, which may seem counter-intuitive at first.

14.3 MANUAL HINTING

The implementation of PostScript hinting in Glyphs allows manual and automatic hints inside the same font. Before resorting to manually inserting hints, try to get as far as possible with autohinting. Only glyphs that do not display correctly at low resolutions will need manual intervention.

Manual and automatic hinting cannot complement each other *inside the same glyph*. Any manually hinted glyph is excluded from the autohinting process. Thus, when adding hints manually, the glyph must be hinted *fully* by hand.

There are two types of glyph-level hints, *stem hints* and *ghost hints*. Stem hints describe a vertical or a horizontal stem or stem-like feature of a glyph, like a serif or a crossbar. Ghost hints mark the top and bottom edges when a horizontal stem hint cannot be applied.

In combination with alignment zones, horizontal ghost and stem hints are important for the vertical alignment at the vertical font metrics, like the x-height or the ascender. At low resolutions, the rasterizer will try to vertically align the edges of all hinted horizontal stems that reach into an alignment zone. The horizontal hints must have their Y coordinates in common with the nodes that are supposed to align. A single hint will do for all nodes it touches at its height.

Stem hints can overlap each other, for example, the vertical stem hints in the figure eight. PostScript hinting does not allow overlapping of hints. So, in cases like this, Glyphs will automatically insert pieces of information called *hint*

replacement, which turns hints on or off for different parts of the glyph outline. This handles issues related to overlapping hints.

In a Multiple Master setup, only hints in the main master will be considered. In this case, make sure all manually set hints are linked to nodes on the outline (see section 14.3.1, ‘Stem Hints’, p. 201). See section 14.3.3, ‘Hinting Multiple Masters’ (p. 203) for details.

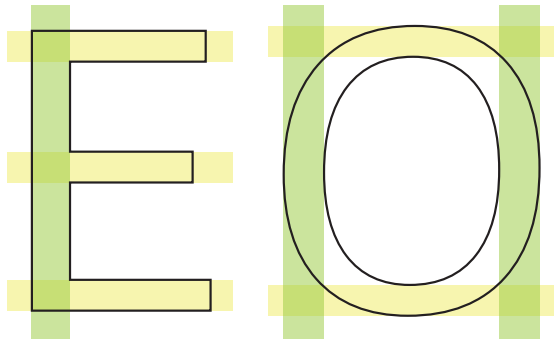
In Edit View, choose *Autohint* from the context menu to get a good start. This way, glyph-level hints will be inserted like the autohinter would have done it when the font is exported. Edit these hints as described in the following sections. Remember that a glyph is not automatically hinted at export if it contains manual hints so that no additional hints will be added at export.

14.3.1 Stem Hints

Add a stem hint to a glyph by choosing *Add Horizontal Hint* or *Add Vertical Hint* from the context menu. A gray bar with a number badge `412, 46` will appear. The two numbers indicate the origin `412, 46` and size `412, 46` of the hint.

Adding a hint while two nodes are selected will *link* the hint to those nodes. Adding linked hints this way even works on multiple node pairs at once, as long as each pair is on a separate outline. For best results, always link hints to extremum nodes (see section 4.2.14, ‘Extremes & Inflections’, p. 32).

Positioning of vertical stem hints (green) and horizontal stem hints (yellow).

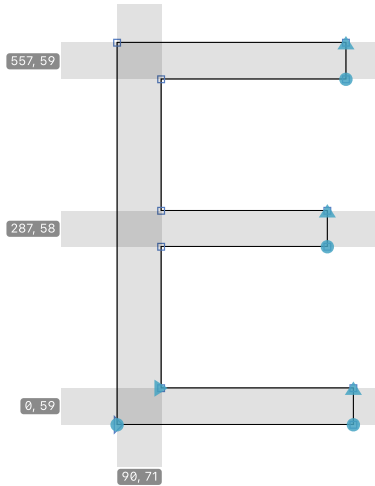


Select a hint by clicking its gray number badge. Shift-click to select multiple hints. Edit the value of the selected hint in the Info box (*View > Show Info*, Cmd-Shift-I). Press Tab to select the next hint, or Shift-Tab to go to the previous one.

412, 46



Edit a hint graphically by dragging the blue marks at the edges of the hint. The blue circle ● indicates the hint origin, while the triangle ▲ shows the size and orientation of the hint. When one of the markers is dragged onto a node, Glyphs will link the hint to the position of the node. Moving the node will also adapt the hint. Delete the selected hints by pressing the Delete key.

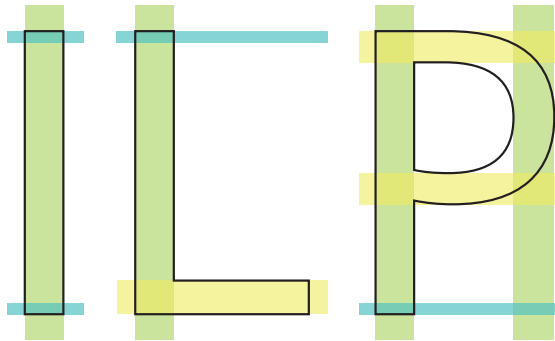


In the exported OTF file, all stem hints must have a width greater than zero. Glyphs will automatically correct hint directions at export to turn all stem hints positive.

14.3.2 Ghost Hints

Use ghost hints to vertically align the top or bottom of a glyph when horizontal stem hints are not applicable. For instance, consider a sans-serif uppercase I. The top needs to align with the cap height zone, the bottom with the baseline zone. In a serif I, horizontal hints would apply to the serifs, but the sans-serif letter lacks the horizontal features necessary for a horizontal hint. In this case, put a top ghost hint on the top of the I and a bottom ghost hint at the bottom of the I. Similar situations occur on the top of a sans-serif L and at the bottom of a sans-serif P:

Positioning of ghost hints (blue) alongside regular stem hints (green). The ghost hints work where no horizontal hints can be applied.



Create a ghost hint by Control-clicking or right-clicking a single node and choosing *Add Horizontal Hint* from the context menu. Turn an existing hint into a ghost hint by Control-clicking or right-clicking the coordinate badge of a hint and choosing *Make Ghost Hint*.



The badge of a ghost hint only displays the position and its orientation. An upward arrow ↑ a top ghost hint; a downward arrow ↓ indicates a bottom ghost hint. Attach it to a point by dragging the blue circle onto a node. Set the vertical orientation of a ghost hint by selecting it and clicking the upward ↑ or downward ↓ icon in the Info box (*View > Show Info*, Cmd-Shift-I).

14.3.3 Hinting Multiple Masters

PostScript hints, like TrueType hints, need to be defined for only the main master. By default, that is the first master in the masters list. Set the 'Get Hints From Master' custom parameter in *File > Font Info... > Font* (Cmd-I) to mark a different master as the main master. Provided the hints are linked to nodes on the outline and the paths are compatible, they will be transferred to the corresponding nodes in compatible masters at interpolation time.

Manual hints in other masters will be ignored unless there are no hints in the main master. When using Alternate (see p. 166) or Intermediate layers (see p. 165), insert hints in the layer that replaces the master layer carrying manual hints.

15 TrueType Hinting

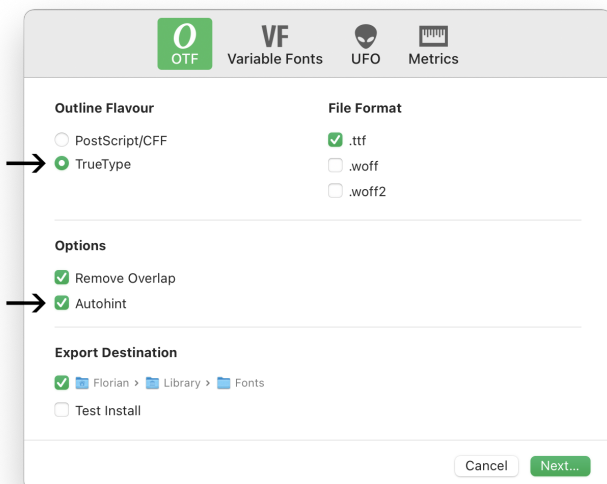
TrueType (TT) hinting optimizes the display of TrueType fonts at low screen resolutions. PostScript/CFF flavor OpenType fonts use a different hinting method; see chapter 14, ‘PostScript Hinting’ (p. 195) for details.

TrueType fonts employ quadratic splines. These are different from the PostScript-style cubic splines that Glyphs uses. When exporting to TrueType, all paths are converted to TrueType-style outlines on the fly, including all manually set hinting instructions.

Glyph-level TT hints, like PostScript hints, need to be defined for only the main master. By default, that is the first master in the masters list. Set the ‘Get Hints From Master’ custom parameter in *File > Font Info... > Font* (Cmd-I) to mark a different master as the main master.

15.1 AUTOHINTING

The technical details of TT hinting are too complex to mix manual and automatic hints. Therefore, checking the *Autohint* option when exporting a TrueType flavor font will ignore any manual TT hints in the Glyphs file:



For information on TTF Autohint, refer to the official website¹.

Glyphs uses TTF Autohint for autohinting TrueType fonts. TTF Autohint can be configured in *File > Font Info... > Export* (Cmd-I) with the ‘TTFAutohint options’ and ‘TTFAutohint control instructions’ custom parameters. See section 15.3.5, ‘Show Point Indexes’ (p. 212) for details on how to get the point indexes needed for the control instructions.

If the *Autohint* export option is not checked, Glyphs includes the manual TT hinting instructions in the exported font. Automatic hinting can also be used as a starting point for manual hinting; see section 15.4, ‘Instructions’ (p. 213) for details.

15.2 FONT-LEVEL HINTS

TrueType hinting uses standard stem values and TT zones on each master. Glyphs uses these font-level hints to replicate the glyph-level hints (see p. 209) from the main master to all other masters.

15.2.1 TrueType Zones

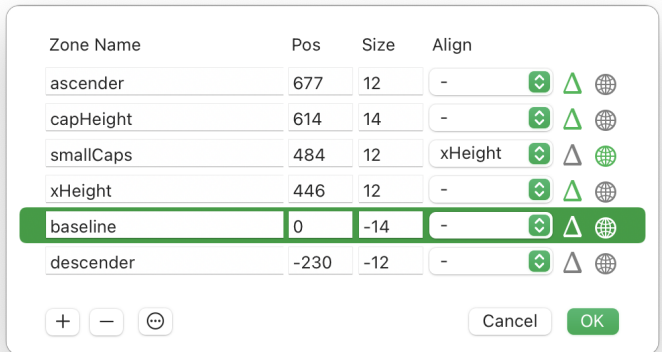
Zones help unify the vertical alignment of shapes throughout the font. When outlines are grid-fitted onto the screen pixels, vertical shape extrema that live in the same zone will be rounded to the same height, typically a pixel edge. Place the zones at vertical metrics such as the baseline, descender, ascender, shoulder height, figure height, nabira height, cap height, rekha height, small caps height, or whatever else makes sense for the design.

A typical TrueType zones setup: top zones with positive sizes at ascender, cap height and x-height; bottom zones with negative sizes at baseline and descender.



By default, TT hinting uses the main alignment zones (see p. 95) defined in *File > Font Info... > Masters > Metrics*. Set up zones specific to TT hinting by adding the ‘TTFZones’ custom parameter in *File > Font Info... > Masters*:

¹ freetype.org/ttfautohint



If the font already contains PostScript hinting zones, choose *Get PS zones* from the actions ☺ menu at the bottom-left of the window to import them as TT zones. Add additional zones with the plus (+) button and remove any selected zone with the minus (-) button. A font can have any number of TT zones. Zones have the following properties:

Name The name identifies a zone in the list. It can be arbitrary text, and each name must be unique. The ‘xHeight’ name is special: it is used for the x-height zone, which is treated differently in grid-fitting. A zone with that name has a higher probability than other zones of rounding up at small sizes. Names are shown in the Info box when using the TT Instructor tool and a Snap hint is selected.




Position The position determines the emplacement of the flat edge of the zone. The flat edge is the offset from the baseline from which the zone is extending. Negative positions are placed below the baseline.

Size The size defines the value of the flattening zone during the overshoot suppression. A top zone has a positive size; a bottom zone has a negative size. Top zones are meant for the top vertical extremes of shapes, like the x-height, shoulder height, cap height, or nabira height. Bottom zones are meant for catching the bottom ends of shapes, like the baseline, the descender, or the bottoms of small figures such as numerators, inferiors, and superiors.



Ensure that the area defined by the position and the size

encompasses all overshoots that are supposed to be flattened to the zone position by the rasterizer. For instance, at the x-height, put the *Position* to the height of the lowercase x, and make sure the *Size* is large enough to catch all overshoots, like in the lowercase o. This works similarly to PostScript alignment zones (see p. 197).

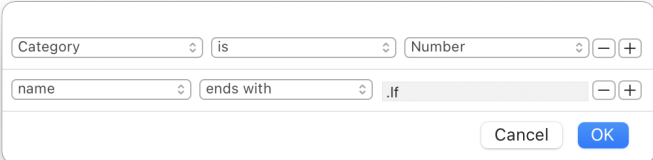
Alignment Link a zone to another zone with the *Align* option. If a zone is aligned to another, the distance between the zone positions is rounded and applied to the zone. This will result in more consistent transitions from one PPM size to the next. Use this for zones that are very close to each other, perhaps even overlapping, and where it may be problematic if the zones diverge too far at low-resolution pixel renderings. Aligned zones are displayed at the same height if their distance is less than half a pixel, one pixel apart if their distance is half a pixel, and so on.

Deltas (This property only applies to the main master.) Click the delta Δ button to fine-tune a zone at specific PPM sizes. A table of buttons will be shown. The columns of the table are the masters of the font, and the rows are PPM sizes. Click a button to switch between rounding up , down , or no rounding . The delta Δ icon in the zones list is filled with the accent color if any button is set to rounding up or down.

Rounding is helpful if a zone is one pixel too low or too high at a given PPM size. If it is too low, round it up; if it is too high, round it down. Use the Hinting Preview (see p. 212) to check for any zones that need to be rounded up or down.

Filter (This property only applies to the main master.) Click the globe  button to limit the zone to a subset of glyphs. The subset is defined like a Smart Filter. See section 6.6.4, ‘Smart Filters’ (p. 86) for details. The globe  icon in the stems list is filled with the accent color if any filters are set.

PPM (*pixels per em*) is a font size measurement. See section 15.3.2, ‘Pixel Size’ (p. 211) for details.

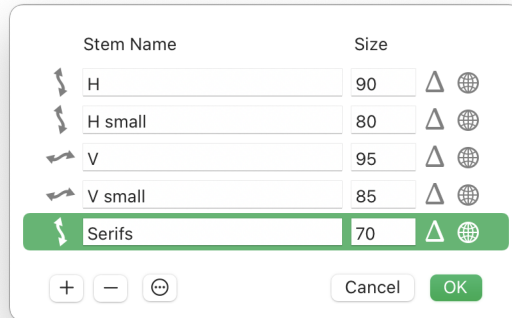


Category	is	Number	-	+
name	ends with	.lf	-	+

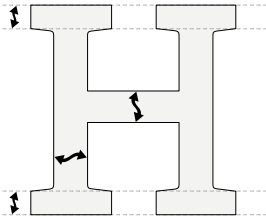
Cancel OK

15.2.2 TrueType Stems

Stems help unify the displayed size of stroke thicknesses at low resolutions. By default, TrueType hinting uses the standard stems of a master (see p. 97). Define stems specifically for TT hinting by adding the 'TTFStems' custom parameter in *File > Font Info... > Masters*. Click the value field of the parameter to edit the stems:



If the font already contains PostScript stems, choose *Get PS stems* from the actions menu to import them as TT stems. Add additional stems with the plus button and remove a selected stem with the minus button. The following stem properties can be edited:



Orientation A hint is oriented in the X-direction or Y-direction . Click the double arrow icon to switch between the two orientations.


The orientation of a hint describes the direction in which its points are moved. An X-direction hint spans the thickness of a vertical stem. A Y-direction hint spans the thickness of a horizontal stem.

Name The name labels the stem. It can be arbitrary text, but it is a good idea to use descriptive names.

Size The average stem thickness. In screen rendering, similarly-sized stems will be unified and displayed with the same number of pixels.

Deltas (This property only applies to the main master.) Click the delta button to fine-tune a stem at specific PPM sizes. This property works like the *Deltas* of TT zones (see p. 207). Note that

stem deltas are only applied for ClearType-style rendering modes (also referred to as Windows GDI).

Filter (This property only applies to the main master.) Click the globe  button to limit the zone to a subset of glyphs. This property works like the *Filter* of TT zones (see p. 207).



15.2.3 TrueType BlueFuzz

Use the ‘TTFBlueFuzz’ custom parameter to extend all TT zones by a certain amount. The specified amount will be added both above and below to each zone. It defaults to 1 unit.

The parameter is helpful for testing stem values or for fixing imprecisions in interpolation: Even if the drawings end up nicely in the zones in all masters, they may still drop out of them in interpolated instances. Such imprecisions can occur due to rounding errors. Test the font using the Hinting Preview (see p. 212) to see if the zones work in all instances, and if not, increase the ‘TTFBlueFuzz’.

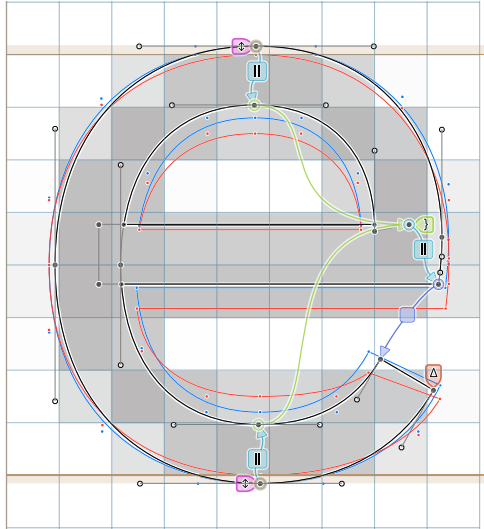
Add the parameter in *File > Font Info... > Font, Masters, or Exports*. Adding it to the *Font* tab applies it to all instances equally. If the parameter is instead added to the masters, it will be interpolated at export. Add it to a single instance in *Exports* to overwrite the masters and font settings.

15.3 GLYPH-LEVEL HINTS

Add glyph-level hints with the TrueType Instructor tool  (shortcut I). Note that the TrueType Instructor tool uses shortcuts without modifier keys, which are usually reserved for tool switching. Therefore, pressing A will not switch to the Annotation tool , but instead, it adds a Snap instruction. Use the toolbar at the top of the window to switch tools.

15.3.1 Hinting Outlines

A typical TT hinting setup. Blue and red outlines are shown in the background. In color, the hinting instructions overlapping the outlines are described in section 15.4, ‘Instructions’ (p. 213).



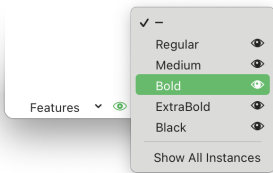
In the TrueType Instructor tool, three outlines are shown. The outline of the current master is **black**. The nodes and handles of this outline can be selected. Nodes are shown as gray ● discs instead of green ○ circles and blue □ squares. Extra nodes (see p. 150) are also shown, regardless of the *View > Show Nodes > Extra Nodes* setting.

The **blue** outline shows the outline of the current instance *before* TT hinting is applied. Change the current instance by choosing from the menu next to the eye icon located at the bottom-left of Edit View. Click the eye icon if the instances menu does not appear to the right.

If the selected instance corresponds to the current master, no blue outline is shown. For example, when both the *Regular* master and the *Regular* instance are currently selected, the black and blue outlines overlap and only the black outline is shown.

The **red** outline shows the current instance *after* TT hinting and grid-fitting are applied according to the current rendering intent (Grayscale, ClearType, or DirectWrite) and the currently selected pixel size. The outline is drawn with code from Microsoft and is the exact outline distortion used on Windows. It is not shown if the glyph has no TT hints.

Note that the pixel preview in the background is only an approximate example. Additional filtering happens between the



red outline and the pixels appearing on-screen. This filtering is subject to so many customizations (*gammas, color modes, transparencies, ...*) that an exact pixel image cannot be predicted.

Toggle the display of the blue and red outlines with *Show Pixel Preview* from the context menu of Edit View.



15.3.2 Pixel Size

With the TrueType Instructor tool active, the Info box has a pixel size field. This field contains the pixels per em value (PPM) of the pixel image preview. The PPM is the true pixel size of the font. For example, consider a font with 1000 UPM and a pixel size of 12 PPM. Then, one pixel is equivalent to $1000 \div 12 \approx 83$ font units.



Change the pixel size by entering a new value, or use the stepper buttons to increase or decrease the size. With the pixel size field active, press the arrow keys Up and Down to change the value. When the field is not active, press the Period (.) or Plus (+) key to increase the size and the Comma (,) or Minus (-) keys to decrease it.

15.3.3 Hint Direction

The current hinting direction is controlled by the wavy arrow button in the Info box. Click the button to toggle between vertical  and horizontal  hinting. Pressing the X key also toggles between the two modes. Vertical hints are used with horizontal stems and vice versa (see section 15.2.2, ‘Orientation’, p. 208).

The hints of the other direction are shown dimmed and grayed-out in the background for reference. They cannot be edited.

15.3.4 Hint Order

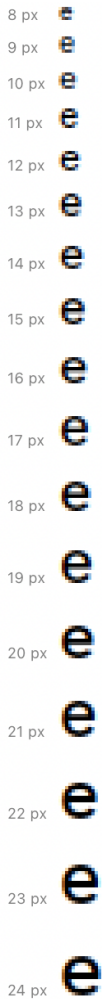
Hints are applied in chronological order. The order is significant because hints build on top of each other. When a hint is attached to a point on the outline, the point executes the movement implied by the hint and becomes a *touched* point.

After all hints have been applied, the positions of all remaining *untouched* points are interpolated between the touched points


nearest in the point index order. This is sometimes referred to as IUP, or *Interpolate Untouched Points* instruction.

15.3.5 Show Point Indexes

Choose *Show Point Indexes* from the Edit View context menu to label all on-curve nodes with their point index (or ‘point ID’). These numbers are needed for the ‘TTFAutohint control instructions’ custom parameter (see section 15.1, ‘Autohinting’, p. 204).



15.3.6 Hinting Preview

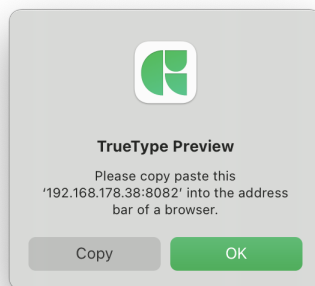
With the TrueType Instructor tool  active, the Preview Area (see p. 56) moves to the right side of Edit View and shows a list (or ‘waterfall’) of the glyphs in Edit View set in the currently selected instance.

The list displays a preview of the hinted glyphs at each size from 8 PPM to 80 PPM. View different preview sizes by dragging the list with the mouse cursor up and down. This preview is also available in the Preview Panel (*Window > Preview Panel*).

The preview can be displayed at three scales: no scaling (1x), 2x, and 4x. Control-click or right-click anywhere in Edit View and choose one of these scaling modes from the context menu.

15.3.7 Web Preview

Choose *Show Preview Address...* from the Edit View context menu to start a web server previewing the font similarly to the built-in Hinting Preview. Glyphs will present a dialog window with a Web address. Click *Copy* to copy the address.



Open the address in a web browser to see the preview page. The Mac does not use TT hinting, so open the page on a virtualized

Windows running on the Mac or another device connected to the same local network.

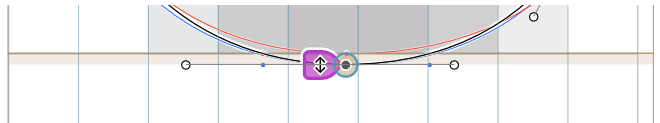
15.4 INSTRUCTIONS


Add TrueType instructions by selecting nodes and choosing one of the available instruction types from the context menu. Click and drag to reposition instructions.

Choose *Autohint* from the context menu to add some instructions automatically. This action will add Snap and Stem hints, but not Shift, Interpolate, or Delta hints. The result can be useful as a starting point for manual TT hinting. Some glyphs cannot be autohinted. In that case, the error message ‘There was a problem compiling TrueType instructions’ appears. Autohint the current glyph by pressing `Cmd-Ctrl-Opt-Shift-I` (configure the shortcut in the *Commands* section in the app settings, see p. 20). This shortcut also works when multiple glyphs are selected in Edit View.


Remove all glyph-level hints from the current glyph by choosing *Remove Hints* from the context menu.

15.4.1 Snap (A)



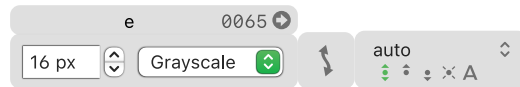
Snap instructions  (shortcut A) round the position of previously untouched points to the nearest pixel edge. Apply a Snap instruction by selecting one or more untouched points and pressing A or selecting *Snap Point* from the context menu.










Select a Snap instruction by clicking its purple badge. The Info box (*View > Show Info*, `Cmd-Shift-I`) shows configuration options for the selected instructions. From the pop-up menu located at the top of the Info box, choose one of the following options:

- ▶ *Auto*: In vertical  mode, the point will snap onto the pixel edge the zone is rounding to if it is inside a zone. Configure the zone rounding with the ‘TTFZones’ custom parameter. *Auto* is typically the best option.
- ▶ *No Zone*: Tells the instruction to ignore zones altogether. In that case, it will only look for the nearest pixel edge.

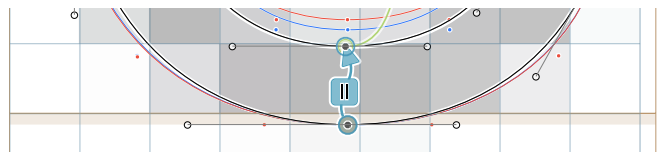
- ▶ The other options are the named zones (see section 15.2.1, ‘TrueType Zones’, p. 205). Choosing a zone will keep the node at a constant distance to the selected zone and ensure that the snapped node will not fall on the other side of the zone. Note that these zone options are not intended for snapping a node to a zone. Instead, it works similarly to the alignment of one zone with another zone in the ‘TTFZones’ custom parameter (see p. 205).


The icon buttons on the bottom of the Info box indicate the rounding that the instruction triggers:



- ▶ *Round*  (default) rounds the point to the nearest pixel edge.
- ▶ *Round up*  rounds the point to the nearest pixel edge above (in vertical  mode) or right (in horizontal  mode).
- ▶ *Round down*  rounds the point to the nearest pixel edge below (in vertical  mode) or left (in horizontal  mode).
- ▶ *No rounding*  keeps the point at its original position. Useful for suppressing the effects of IUP.
- ▶ *Round only in GDI ClearType*  works like *Round*, but only in the ClearType rendering intent. Useful for antialiasing in the Grayscale and DirectWrite intents, while ClearType has to round to the full pixel. This mode is only effective when hinting horizontal stems because GDI ClearType has no vertical oversampling.


15.4.2 Stem (S)



Stem instructions  (shortcut S) round the positions of either two previously untouched points or one touched and one untouched point to a distance determined by the TT stem. A Stem hint has an originating point and a target point (indicated by the arrowhead). The target point is moved to follow the distortion of the originating point. Stem hints can be added to

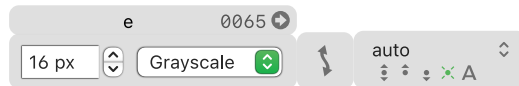
multiple point pairs at the same time, provided an even number of points is selected. Flip the origin and target points by choosing *Reverse* from the context menu on the Stem hint.






If a Stem hint starts in a zone, it will automatically snap the originating point in the zone. A Snap instruction is therefore not needed for the originating node.

Select a Stem hint by clicking its sky-blue  badge. Configure the selected Stem hint in its Info box. From the pop-up menu located at the top of the Info box, choose one of the following options:

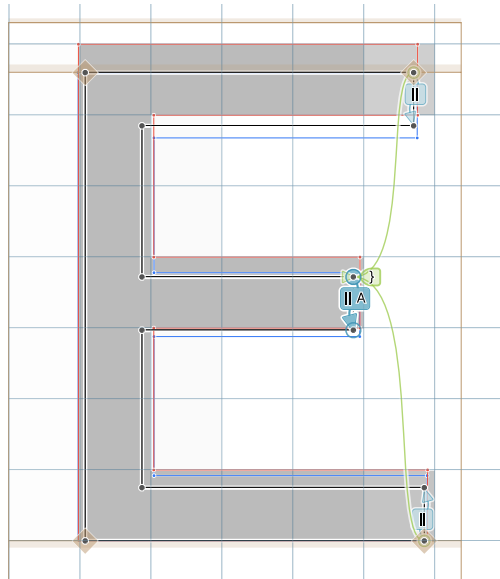
- ▶ *Auto*: The hint will use the closest stem as defined in the ‘TTFStems’ parameter. This is the best mode, provided the stems are clearly distinct.
- ▶ *No Stem*: The target points are not snapped to the grid; only their distance to the originating node is kept. This means that the target point is moved only if the originating point is moved as well. This is the best mode for higher shape fidelity in medium-range pixel sizes (not too small) or if the distortions in the resulting rendering environment are otherwise too important.
- ▶ The other options are the named stems (see p. 208). Select one of these options only if the auto mode would give the wrong results.


The icon buttons on the bottom of the Info box indicate the rounding that the instruction triggers:




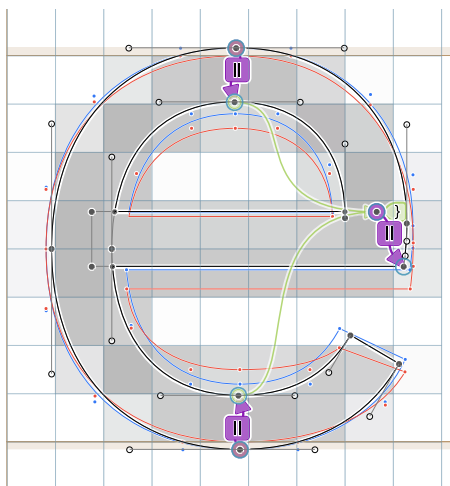
- ▶ *Round*  rounds the target point to the nearest pixel edge.
- ▶ *Round up*  rounds the stem size up. This will snap the target point to the pixel edge that is further away from the originating point. If the Stem hint is pointing downwards, that will be below.
- ▶ *Round down*  rounds the stem size down. This will snap the target point to the pixel edge that is closer to the originating point. If the Stem hint is pointing downwards, that will be above.
- ▶ *No rounding*  (default) does not round the stem to any pixel edge, but will still round its size to the oversampling edges provided by the current rendering intent.
- ▶ *Round only in GDI ClearType*  works like the *Round* mode, but only for the GDI ClearType rendering intent. If a stem is not

adjacent to a zone, this mode can help keep at least one edge of the stem aligned with the closest pixel boundary. This is useful for the middle bars in E and e, and best if combined with an Interpolate instruction (see p. 219):

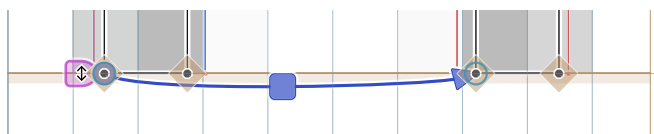





The stem widths taken from the 'TTFStems' parameter are rounded according to the oversampling of the respective rendering intent. Stems should not be rounded (with the default *No rounding*  mode) not to overwrite that. Having stems that are not rounded to full pixel edges means that at least one side of the stem will get a gray border. The resulting rendering is a little less sharp but much better preserves the details of the design. It also produces fewer problems with distorted outlines, for example, fewer collapsed counters.

If there are three horizontal stems, select all three Stem hints by consecutively Shift-clicking them, then choose *Make Triple Hint* from the context menu. The Stem hints will turn purple , indicating they are connected and will try to preserve at least one-pixel distance even in the worst of circumstances:




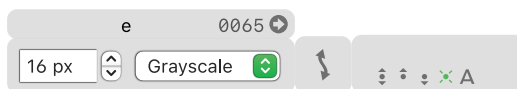
15.4.3 Shift (F)




Shift instructions    (shortcut F) transfer the movement of a touched (originating) point to an untouched (target) point. This shifts the target point the same way the originating point is shifted. Use Shift instructions to make sure that two parts of a glyph implement the same distortion.

Apply a Shift instruction by selecting a touched and an untouched point, and press F or choose *Shift Points* from the context menu. Multiple Shift instructions with the same originating point can be added simultaneously, provided only one of the selected points is touched (originating point), and all others are untouched (target points).

Select a Shift hint by clicking its dark blue  badge. The icon buttons in the Info box indicate the rounding that the instruction triggers:

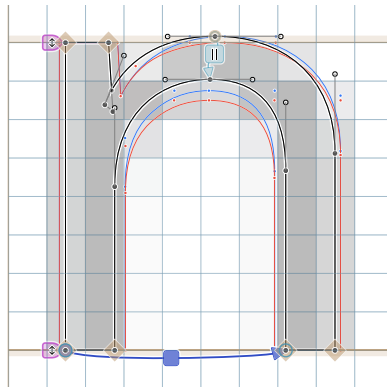


- ▶ **Round**  rounds the target point to the nearest pixel edge.

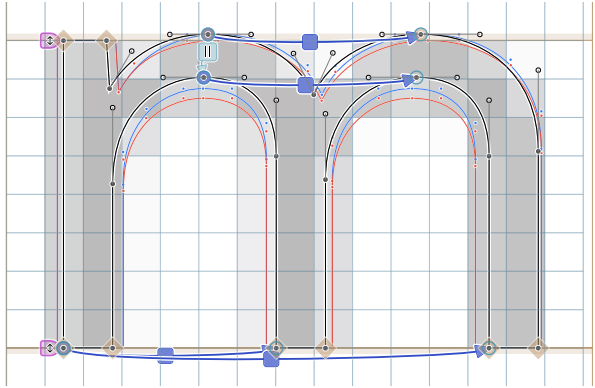
- ▶ *Round up* ⬆️ rounds the target point to the nearest pixel edge above.
- ▶ *Round down* ⬇️ rounds the target point to the nearest pixel edge below.
- ▶ *No rounding* ✖️ (default) keeps the transferred movement exactly as it is without any additional rounding. This is typically the best mode for Shift instructions.
- ▶ *Round only in GDI ClearType* ⚠️ works like *Round*, but only for GDI ClearType.

For example, consider the two legs of a Latin lowercase sans-serif n. The left leg should get snapped in the baseline zone. But, after adding a Stem hint for the shoulder between the two legs, the right leg is out of sync with the left leg.


This happens because, along the line of the path, the effect of the Snap instruction in the lower left is interrupted by the Stem instruction on the top. Therefore, for the right leg, the ensuing IUP (interpolation of untouched points) can only extrapolate the distortion caused by the Stem instruction, not the Snap instruction. Effectively, the Snap instruction is confined to the left leg. Mitigate this by adding a Shift instruction from the touched (snapped) point on the left leg to one of the points on the right leg:



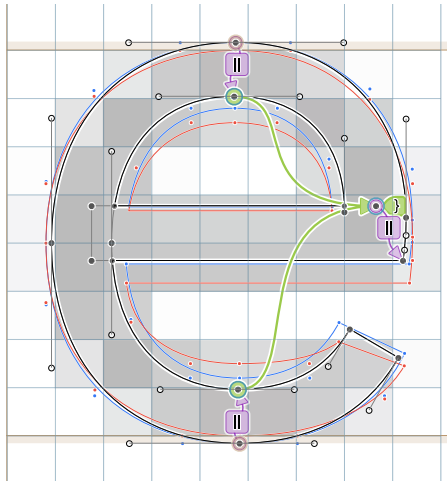
The Shift instruction duplicates the movement of the left leg to the right leg, making the target point a touched point. This way, the right leg will always perform the same movement as the left leg. The lowercase Latin m may use multiple Shift instructions:




15.4.4 Interpolate (G)






Interpolate instructions  (shortcut G) keep a previously untouched point at the same relative position to two touched points as in the original uninstructed outline. Apply an Interpolate instruction by selecting two touched points and a third, untouched point. Then press the G key or choose *Interpolate* from the context menu.

The main intention of Interpolate hints is to remedy unwanted side effects of IUP. For example, use an Interpolate instruction to keep a middle stem at the same relative distance from the (already touched) outer stems, as in this lowercase e:

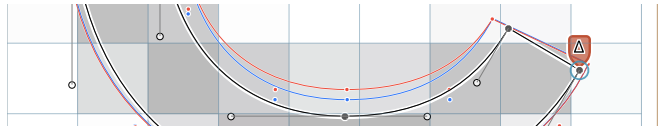




Select a Shift hint by clicking its green  badge. The icon buttons in the Info box indicate the rounding that the instruction triggers:




- ▶ **Round**  snaps the target point to the nearest pixel edge. Choose this mode when adding a Stem hint on top of the target point.
- ▶ **Round up**  snaps the target point to the next pixel edge above.
- ▶ **Round down**  snaps the target point to the next pixel edge below.
- ▶ **No rounding**  (default) keeps the interpolated position and does not round it. This is typically the best mode for Interpolate instructions.
- ▶ **Round only in GDI ClearType**  will snap to the nearest pixel edge for the GDI ClearType rendering intent; the others should remain unaffected.

15.4.5 Delta (E)



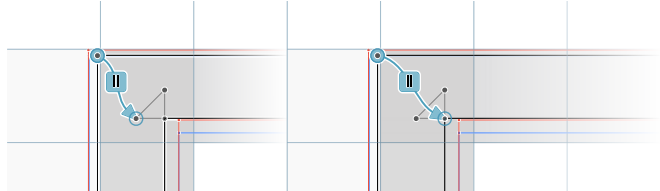
Delta instructions  (shortcut E) move a point up or down by exactly one pixel, but only in a specific static instance and a specific PPM size. Delta hints are intended as final pixel corrections after all other hints have been applied. Add a Delta hint by selecting any number of points and pressing E or choosing *Delta* from the context menu. Select one or more Delta hints, click the Delta  icon in the Info box, and configure them like the deltas of a TT zone. See section 15.2.1, ‘Deltas’ (p. 207) for details.

Delta hints should be used sparsely, if at all, and should be added last, after all other hints. Delta hints are only available for vertical  hinting and are ignored in variable fonts.

15.4.6 Points in Overlapping Intersections

Instructions may be placed on any node shown in the TrueType Instructor tool. However, path intersections are handled

separately. When a font is exported with the *Remove Overlaps* option selected, overlapping intersections are reduced to a single point. Glyphs will move any hints on points that are removed as part of the overlap removal to the nearest resulting intersection point. Thus, the following two hints are equivalent in *non-variable fonts*:



In *variable fonts*, overlaps are not removed. Hints added to extra nodes, such as the right hint in the image above, are ignored. When exporting variable fonts, only add hints to normal nodes, like the left hint in the image above.

15.5 ADVANCED TRUETYPE HINTING

Use the following custom parameters to further configure the TT hinting of the exported font files.

TTFOvershootSuppressionBelowPPM Add this parameter in *File > Font Info... > Font* (Cmd-I). Set it to the pixel size, below which overshoots are collapsed to the flat edge of their zone. The flat edge is the position of a zone as configured in the ‘TTFZones’ parameter (see p. 206).

TTZoneRoundingThreshold Add this parameter in either the *Font* or *Exports* tab in the Font Info window (Cmd-I). The *Font* value is used for all instances where the parameter is not set. This value controls the likelihood of a positive zone being pushed up a pixel. It takes a small decimal number, typically something around 0.1 or 0.2. The value is added to any positive zone position before rounding, and added twice to the x-height zone (the one named ‘xHeight’ in the TrueType zones, see p. 206). Its default value is 0.09375.

For example: At a certain font size, the small caps zone ends up at 6.45 px, and the x-height at 5.25 px. Without any change, the small caps zone rounds and snaps to a height of 6 pixels, while the x-height ends up with 5 pixels. But when setting the

rounding threshold to 0.2, the small caps height ends up at $6.45 + 0.2 = 6.65 \approx 7$ pixels, and the x-height at $5.25 + 2 \times 0.2 = 5.65 \approx 6$ pixels.

TTMinimumDistance Add this parameter in *File > Font Info... > Exports* (Cmd-I). The default value is 0.25, meaning that any hinted stem will be drawn with a minimum length of a quarter pixel, regardless of which PPM size it has a stem hint applied. If the default value does not fit the design, add this parameter with a custom minimum distance (in pixels).

See the TrueType Reference Manual² for details.

Control Value Program Three custom parameters are added to the masters when opening a hinted TrueType font: ‘CVT Table’, ‘prep Table Assembly’, and ‘fpgm Table Assembly’. These parameters contain the assembly code for the existing TT hinting and are rarely edited manually. Instead, they are used not to lose the existing hinting instructions on export. They correspond to the cvt, prep, and fpgm tables in the font.

gasp Table Add this parameter in either the *Font* or *Exports* tab in the Font Info window (Cmd-I). The *Font* value is used for all instances where the parameter is not set. This parameter configures the grid-fitting and scan-conversion procedure for TrueType fonts. It controls the two PPM thresholds at which the recommended on-screen rendering behavior changes. The gasp table contains rendering recommendations for both a traditional Grayscale and a ClearType subpixel renderer. However, keep in mind that a renderer may ignore the data stored therein. The default threshold sizes are 8 and 20 PPM. Because there are two thresholds, three ranges can be differentiated:

Source for quoted parts: *Now read this: The Microsoft ClearType Font Collection*, Microsoft 2004, p. 14.

- ▶ *No Hinting & Symmetric*: Until the first threshold size, no grid-fitting is applied, and text is rendered with antialiasing wherever possible. ‘At very small sizes, the best appearance on grayscale devices can usually be achieved by rendering the glyphs in grayscale without using hints.’
- ▶ *Hinting & Asymmetric*: Between the two threshold sizes, the renderer is recommended to apply grid-fitting and suppress grayscale. ‘At intermediate sizes, hinting and monochrome rendering will usually produce the best appearance.’ In ClearType, the matter is handled asymmetrically: vertical grid-fitting is applied, while horizontally, subpixel rendering is used.

2 developer.apple.com/fonts/TrueType-Reference-Manual/RM03/Chap3.html

- ▶ *Hinting & Symmetric*: Beyond the thresholds, the rasterizer is instructed to apply grid-fitting and render the shapes in grayscale. ‘At large sizes, the combination of hinting and grayscale rendering will typically produce the best appearance.’ The ClearType rasterizer is instructed to apply symmetric smoothing. This uses antialiasing in the Y direction in addition to horizontal subpixel rendering. ‘At display sizes on screen, [...] this new improvement of the Windows font renderer produces smoother and cleaner-looking type.’

16 Color Fonts

Glyphs offers a streamlined workflow for creating four types of color fonts: classic layer fonts, Microsoft-style CPAL and COLR OpenType tables, the Apple-style sbix OpenType table, and the Mozilla/Adobe-style SVG table.


16.1 WORKING WITH COLOR FONTS

Color fonts make use of multiple layers per glyph. See section 11.8, ‘Editing Multiple Masters’ (p. 169) for general information on working with multiple layers.

16.1.1 Keeping the Metrics in Sync

When working on a color font project, masters must share the same metrics and kerning pairs so that all color layers align. Use the ‘Link Metrics With First Master’ custom parameter in *File > Font Info... > Masters* to sync the metrics and kerning of the first master with all masters. Add this custom parameter to every master except the first one. Alternatively, add the ‘Link Metrics With Master’ custom parameter, which can be linked to any other master, not just the first one.

16.1.2 Previewing Color Fonts

Edit View shows colors for other glyphs when a color layer is selected in the current glyph. The Preview area at the bottom of Edit View (activated with the eye  button) and the Preview Panel (*Window > Preview Panel*) display a glyph in color when one of its color layers is selected. The Text Preview (*Window > Text Preview*) displays the font as it would appear in Mac applications using Core Text (such as Text Edit or Pages).

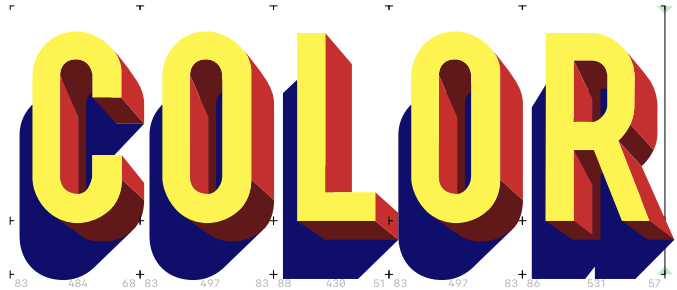
16.1.3 Exporting Color Fonts

Color fonts are regular OpenType fonts. They are compatible with all OpenType flavors supported by Glyphs: PostScript/CFF (.otf), TrueType (.ttf), and WOFF/WOFF2. However, color fonts only work in environments that support their display. For instance, layered color fonts only work in applications that can put pieces of text precisely on top of each other, like most DTP software.

16.2 LAYERED COLOR FONTS



The four fonts which result in the stacked text.



Layered color fonts are separate fonts that are stacked on top of each other. For such a font to work properly, the software environment in which it is used must support the stacking of text layers.

16.2.1 Initial Setup

Layered color fonts have a color axis. Add it in *File > Font Info... > Font* by clicking the plus $+$ button next to the *Axes* heading. Name the axis ‘Color’ and use a four-letter tag such as ‘COLR’.



Each color gets its own master. Contrary to most other fonts, the masters of a layered color font are not interpolated. This means that the masters of a layered color font are not required to be compatible for interpolation. Signal this to Glyphs by adding the ‘Enforce Compatibility Check’ custom parameter in the *Font* tab and unchecking it:



With this parameter disabled, add masters for each color layer. Give each master a descriptive name like ‘Front’, ‘Inside Light’, ‘Inside Shadow’, or ‘Outer Shadow’. In the *Axes Coordinates* section, set a different numeric *Color* value for each master; for example, 1 for the first master and 2 for the second.




Add a ‘Master Color’ custom parameter to each master to define its preview color. Note that custom parameters can also be added when multiple masters are selected. This color is only

Pro Tip: Additionally, add a ‘Master Color Dark’ custom parameter to use a different color for the dark system appearance.

used inside Glyphs; the font user can set the exported font files in any color. Drag the *Opacity* slider in the color picker to use a semitransparent color for the preview.

Switch to the *Exports* tab, click the plus (+) button located in the bottom-left and choose *Add Instance for each Master*. This will add an exporting instance for each axis coordinate of the *Color* axis.

16.2.2 Editing Color Layers

Edit color layers like normal master layers in Edit View (see p. 55). By default, the currently selected master is shown. Click the eye /  buttons in the *Layers* palette (see p. 61) to view multiple layers at the same time. When working with layered color fonts, it is common to show  all master layers.

The order of the masters in *File > Font Info... > Masters* is reflected in Edit View: the first master is shown on top, the second master one layer below, and so on.

16.2.3 Exporting

Layered color fonts are exported as separate font files, one for each color layer. These color layers use the instances as described in section 16.2.1, ‘Initial Setup’ (p. 225).

Glyphs can also convert a layered color font to an SVG color font that includes all color layers in a single font file. For that, add a new instance in *File > Font Info... > Exports* and add the ‘Color Layers to SVG’ custom parameter. Then, add the ‘Export SVG Table’ custom parameter to the same instance for the SVG data to be included in the export. Ensure that the checkboxes of both custom parameters are checked.

16.3 COLR/CPAL FONTS

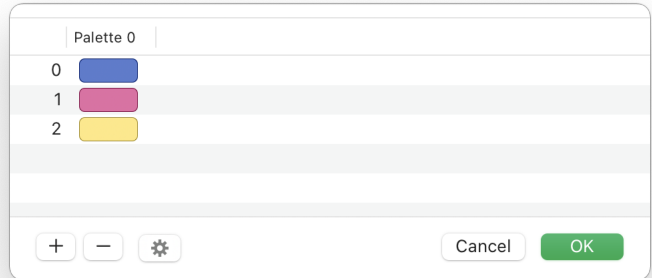


Microsoft-style color fonts are fonts that contain two additional tables: CPAL (Color Palette) and COLR (Color). These color fonts work like layered color fonts (see p. 225), but they export all color

layers to a single font file instead of exporting one font file per layer.

16.3.1 Defining the Color Palette

In *File > Font Info... > Font*, add the ‘Color Palettes’ custom parameter. Click its value to edit the color palette.



Click the plus (+) button to add additional colors to the palette. Each color is identified by a color index counting up from 0. Change colors by clicking a color swatch and choosing a different value with the color picker. Select a row and click the minus (-) button to delete a color.

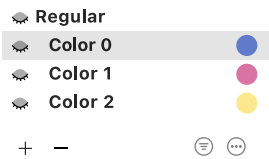
Multiple color palettes can be defined. Add additional color palettes by choosing *Add Palette* from the gear ⚙️ menu. Click the column heading of a palette to remove it with *Remove Palette*. See section 16.3.4, ‘Exporting’ (p. 228) for details on choosing a palette for each instance.

Confirm the edits to the color palettes with *OK*, or click *Cancel* to discard all edits and keep the palettes as they were.

16.3.2 Master Layer as Fallback

CPAL/COLR color fonts are not supported by all applications. In case an app cannot handle the color information, it displays the master layer. The master is not displayed in apps that support CPAL/COLR color fonts.

16.3.3 Color Palette Layers



Add color layers by clicking the plus + button in the *Layers* palette (*Window > Palette*, Cmd-Opt-P) and choose *Color Palette* from the context menu on the layer. Pick a color from the color palette by clicking the color swatch to the right of the layer name. Choose either a predefined color or the asterisk (*) option. The

asterisk option uses the color that the font user defines. Glyphs previews this color in black (or in white, for the dark system appearance).

Color Palette layers are named ‘Color’ and the color index from the ‘Color Palettes’ custom parameter. Multiple layers of a glyph can have the same color, and some colors of the palette might not be used at all in a glyph. Not all glyphs need to have the same number of color layers: some might only have a single color layer, while others have multiple layers for each color.

16.3.4 Exporting

Exported instances of CPAL/COLR fonts use the first color palette by default. Add the ‘Color Palette for CPAL’ custom parameter to an instance in *File > Font Info... > Exports* to choose a different color palette for that instance. Note that the color palette IDs start at 0 for the first palette. Add additional palettes as described in section 16.3.1, ‘Defining the Color Palette’ (p. 227).

Remove all colors from an instance by adding an ‘Export COLR Table’ custom parameter and unchecking it. This will use just the fallback layer and discard the CPAL and COLR tables.

Glyphs can also convert a CPAL/COLR font to an SVG color font by adding the following custom parameters in *File > Font Info... > Exports*:

- ▶ Check the ‘Color Layers to SVG’ custom parameter.
- ▶ Set ‘Color Palette for SVG’ to the number of the palette that should be used for the conversion. Palettes are numbered starting at ‘0’. The palette number is written in the column header in the ‘Color Palettes’ custom parameter (see p. 227).
- ▶ Check the ‘Export SVG Table’ custom parameter for the SVG data to be included in the export.

Custom Parameters			+
<input checked="" type="checkbox"/>	Color Layers to SVG	<input checked="" type="checkbox"/>	—
<input checked="" type="checkbox"/>	Color Palette for SVG	<input type="text" value="0"/>	—
<input checked="" type="checkbox"/>	Export SVG Table	<input checked="" type="checkbox"/>	—

16.4 SBIX FONTS



See the TrueType Reference Manual¹ for details on sbix color fonts.

Apple-style color fonts include an sbix table (Standard Bitmap Graphics) containing bitmap data of various resolutions. Multiple images of various sizes may be assigned to each glyph. Thus, the device displaying it can pick the most appropriate resolution.

16.4.1 Standard Bitmap Graphics

In contrast to the other color font formats covered in this chapter, sbix fonts do not use vector but bitmap graphics. These graphics may be prepared as PNG, JPEG, or TIFF images.

Each glyph contains a single image file. However, images can be provided for different pixel sizes. A glyph might contain a 512 × 512 pixel image, but also lower resolution versions at, for example, 128, 32, and 16 pixels. While a single large image per glyph could be scaled down to fit every size below, including smaller sizes of images presents two advantages:

- ▶ Providing images for smaller sizes allows fine-tuning the graphics for low-resolution output. For example, the design might be adapted to read more clearly at smaller pixel sizes.
- ▶ If smaller images are not included in the font, the text renderer needs to scale the large images down to the desired font size. This can be slow, especially on low-end devices, and use more energy, which is relevant to mobile devices.

Images are scaled up by the text renderer when using a font size above the largest image dimensions, resulting in blurry glyph images.

16.4.2 Preparing Images

Create the graphics in an image editing application and export them at the desired sizes. This size is measured in pixel units, not in font size units. For example, export images at heights of 512, 256, 32, and 16 pixels.

1. developer.apple.com/fonts/TrueType-Reference-Manual/RM06/Chap6sbix.html

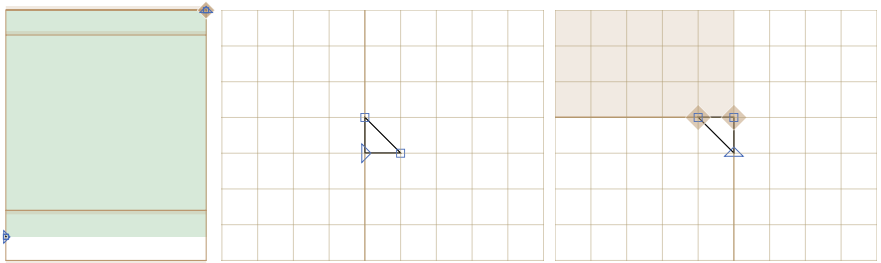
The differently sized images for a glyph must all share the same aspect ratio. This aspect ratio is width ÷ height. So, if the image version with a height of 512 pixels is 512 pixels wide (512 ÷ 512 = 1, a square), all other images must also be square. If one image is twice as tall as it is wide (256 ÷ 512 = 0.5), all other images for that glyph must share the aspect ratio of 0.5.


Pro Tip: For clarity, name image files after their glyph name and pixel size, such as 'rainbow 32.png' for 🌈.


When adding these images to a Glyphs file, they are not copied into the Glyphs file, but merely referenced. Moving, renaming, or deleting the image files will break these references. Therefore, it is best practice to place the images in a folder next to the Glyphs file.

16.4.3 Adding Images to Glyphs

The master layer does not contain glyph outlines. However, some software—such as Google Chrome and Adobe Photoshop—use the bounding box of paths on the master layer as a mask for the bitmap images. Therefore, it is a best practice to place tiny paths in the bottom-left and top-right of the master layer, like here:



These tiny paths can also have their nodes on the same single point, thus hiding the path. The red node background  that Glyphs shows for overlapping nodes can be ignored in this case. This bounding box, highlighted in green in the image above, is used by Glyphs to place the images on their color layers.

Color layers for the sbix format are named 'iColor' layers. Add iColor layers by clicking the plus  button in the *Layers* palette (*Window > Palette*, *Cmd-Opt-P*) and choosing *iColor* from the context menu on the layer. The name of an iColor layer includes its size, for example, 'iColor 512' or 'iColor 24'.

Double-click a layer entry in the *Layers* palette to change its size. Scale images by changing this iColor pixel size. Images appear smaller if the entered pixel size is larger than the image, and they appear larger if the entered size is smaller.

than the image.

Add iColor layers for each image resolution. Note that changing the width of the master layer also changes the width of all of its iColor layers.

16.4.4 Exporting

Glyphs includes the sbix table in exported font files if iColor layers are present in the font. Discard the sbix table by adding an ‘Export sbix Table’ custom parameter to an instance and unchecking it.

Glyphs can convert sbix color fonts to SVG color fonts (see p. 231). This allows the color font to be used in software that supports SVG color fonts, but not sbix fonts. Add the ‘SBIX to SVG’ custom parameter to an instance and set its value to the desired pixel size at which Glyphs will convert the sbix image data to SVG data. For example, setting this parameter to 128 uses the iColor layers sized at 128 pixels, if any, otherwise the smallest iColor layers above 128, and if there are none at or above that size, the next largest available iColor layers.

Uncheck ‘Export sbix Table’ and set the ‘SBIX to SVG’ parameter to export an SVG-only color font.

16.5 SVG COLOR FONTS

Scalable

There used to be an SVG format for fonts, not images. That SVG format is now obsolete. The SVG color fonts in this chapter contain SVG images for their glyphs but are still OpenType fonts.

The Scalable Vector Graphics (SVG) image format can contain both vector and bitmap data. An SVG color font uses SVG files to display glyphs. SVG color fonts are also referred to as *OpenType-SVG* or *SVG-in-OpenType*. Glyphs provides three methods for creating SVG color fonts:

- ▶ convert a layered color font, CPAL/COLR font, or sbix font to SVG;
- ▶ use SVG files created in a graphic design app;
- ▶ create the SVG images inside Glyphs.

16.5.1 Converting to SVG

See the respective sections for details on how to convert other color font formats to SVG:

- ▶ Layered color fonts: section 16.2.3, ‘Exporting’ (p. 226)
- ▶ CPAL/COLR color fonts: section 16.3.4, ‘Exporting’ (p. 228)
- ▶ sbix color fonts: section 16.4.4, ‘Exporting’ (p. 231)

16.5.2 Importing Existing SVG Files

In Edit View, add an SVG layer by clicking the plus \oplus button in the *Layers* palette (*Window > Palette*, Cmd-Opt-P) and choose *svg* from the context menu on the layer. An SVG image can also contain bitmap data, which may be useful for fonts imitating handwriting (like the sbix format, see p. 229). Drag an SVG file onto the layer to use it for that glyph.

Bitmap.svg


Tip: For clarity, name image files after their glyph name, such as ‘A.svg’ and place them in a folder named ‘Images’.

When adding SVG images to a Glyphs file, they are not copied into it, but merely referenced. Moving, renaming, or deleting the SVG files will break these references. Therefore, it is best practice to place the files in a folder next to the Glyphs file.

Resize and reposition the SVG image on the glyph layer like any other image. See section 4.12.2, ‘Manipulating Images’ (p. 54).

Changing the width of the master layer also changes the width of its SVG layer. In case an app cannot handle the SVG data, it shows the master layer. The master layer is not shown in apps that support SVG color fonts.

16.5.3 Creating SVG Glyphs

SVG glyphs can be created using color layers. Add a color layer by clicking the plus \oplus button in the *Layers* palette (*Window > Palette*, Cmd-Opt-P) and choose *Color* from the context menu on the layer. Color layers are named ‘Color’ and display a color spectrum  disc.

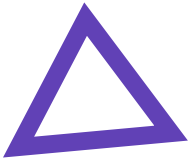
Draw paths on a color layer like on any other layer. By default, the drawn shapes are filled with a gray color. Edit the style of the selected paths in the *Attributes* inspector located at the bottom of the *Palette* (*Window > Palette*, Cmd-Opt-P). Toggle the display of the *Attributes* inspector with *View > Show Info* (Cmd-Shift-I).

By default, the *Attributes* inspector contains controls for the stroke and fill of the selected paths. Click the plus \oplus button next to the *Attributes* heading to show additional controls. The *Shadow*, *Inner Shadow*, and *Pattern Image* options cannot be

exported to SVG; they are used when exporting to PNG or PDF with *Filter > Glyph as Image*.

Change the color of an attribute by clicking the color swatch button. A red stroke through the swatch indicates a fully transparent color.

Stroke



Enter a stroke width in the text field next to the *Stroke* heading. The stroke width is measured in font units. For example, entering ‘12’ will add a 12 unit wide stroke along the path. An empty stroke width field assumes a value of 0. By default, the stroke is centered on the path. Choose the left or right stroke alignment to move the stroke inside or outside the path.

Change the stroke color by clicking the color swatch button. Click the minus — button to remove the stroke by setting the stroke width to 0 and the color to transparent.

Fill



Change the fill color by clicking the color swatch button. Click the minus — button to remove the fill by setting the color to transparent. The fill control is not shown when a gradient is added to a path.



Gradient

Click the plus + button next to the *Attributes* heading to add a *Gradient* effect to the selected paths. A gradient replaces the *Fill* attribute. There are two types of gradients: linear and radial gradients. Click an icon to change the type.



A gradient has at least two color stops. These are indicated as small knobs on the bottom of the gradient bar. Click a knob to edit the stop color. Add additional color stops by clicking anywhere on the gradient bar. Remove a color stop by dragging its knob away from the gradient bar.

In Edit View, knobs control the position of the gradient. For a linear gradient, two knobs control the start and end position of the gradient. For a radial gradient, a single knob controls the origin of the gradient.

Exporting

Export color layers with path attributes to SVG by adding the ‘Color Layers to SVG’ custom parameter in *File > Font Info... > Exports*. If this parameter is checked, Glyphs uses the paths and attributes from the Color layers to produce an SVG table and adds it to the exported font file.

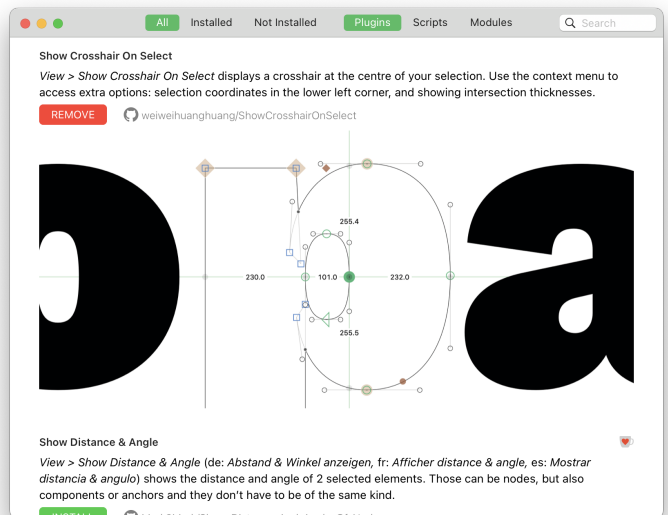
17 Extensions

Extend Glyphs with scripts and plug-ins. There are many existing scripts and plug-ins that can be freely downloaded from the Plugin Manager. Commercial extensions are also available online. Write custom extensions to satisfy specific needs.

All scripts and most plug-ins are written in the Python programming language. For these to run inside Glyphs, ensure that Python is set up in the settings (see p. 18).

17.1 PLUGIN MANAGER



Browse a selection of freely available scripts, plug-ins, and modules in *Window > Plugin Manager*.



At the top of the window are two groups of filters. Filter the extensions by whether they are installed with the *All*, *Installed*, *Not Installed* buttons. Filter by extension type with the *Plugins*, *Scripts*, *Modules* buttons. Enter a keyword in the search field located in the top-right of the window to search by extension name, description, or the name of the extension vendor.

Install an extension by clicking the green **Install** button beside it. Uninstall an extension by clicking its **Remove** button. Glyphs needs to be relaunched for newly installed plug-ins and

modules to be available. Newly installed scripts can be loaded without relaunching Glyphs by holding down the Option key and choosing *Script > Reload Scripts* (Cmd-Opt-Shift-Y). Updated versions of extensions are installed automatically.

Enlarge preview images to their full size by clicking on them. Click the gray link below the extension description to view its website. Some extension vendors accept donations via  PayPal and  Ko-fi. Click the icon to open the donation page.

New extensions are added regularly to the Plugin Manager. Open a pull request at the Glyphs plug-in repository¹ to add a new entry or contact the Glyphs team on the Glyphs forum² or directly per email.³ See section 17.2.3, ‘Creating Scripts’ (p. 237) and section 17.3.2, ‘Creating Plug-ins’ (p. 238) for details on creating new extensions.

Plugin Manager lists extensions from the main repository, accessible to all users. Add alternative repositories as described in section 3.6.3, ‘Alternate Plugin Repositories’ (p. 18). These will also show up in the Plugin Manager, but only for users that have registered the same alternative repository in their Glyphs setup.

17.2 SCRIPTS

Scripts are written in the Python programming language and can automate every part of Glyphs. Some scripts require specific modules to be installed; install these from the *Modules* tab in the Plugin Manager (see p. 235).

17.2.1 Run Scripts

Access all installed scripts from the *Script* menu. Many scripts show a short description when the mouse pointer rests on its menu item for a few seconds. Click a menu item to run the script.

Assign a keyboard shortcut to a script to run it with a key press. See section 3.7, ‘Shortcuts’ (p. 20) for details. Re-run the last script with Cmd-Opt-R or from the bottom of the *Script* menu.

17.2.2 The Scripts Folder

The *Script* menu reflects the files of the Scripts folder. Open the Scripts folder with *Script > Open Scripts Folder* (Cmd-Shift-Y). Reload the *Script* menu by holding down the Option key and

1 github.com/schriftgestalt/glyphs-packages

2 forum.glyphsapp.com

3 glyphsapp.com/contact

choosing *Script > Reload Scripts* (Cmd-Opt-Shift-Y).

17.2.3 Creating Scripts

Add scripts by creating a file ending in `.py` in the Scripts folder. The part of the filename before the `.py` is arbitrary, but for clarity, it is recommended to name the file as it should appear in the *Script* menu, for example, `Some Script.py`. Create subfolders to group scripts. Inside the Python file, add a comment at the top with the following format:

```
# MenuItem: Script Name
```

Replace `Script Name` with the name that should be displayed in the *Script* menu. Add a description that is shown when the mouse pointer rests on the menu item by writing the following code below the `# MenuItem` line:

```
__doc__="Description"
```

Replace `Description` with a short description of the script. The description can span multiple lines:

```
__doc__="""  
This description spans multiple lines.  
Here is the second line.  
"""
```

For writing the actual script, consult the Glyphs Python API documentation.⁴ On the Glyphs website, there is a tutorial series⁵ from the first steps of programming in Python up to writing advanced scripts. Viewing the code of existing scripts is also a great way to learn. If any questions arise, feel free to ask on the Glyphs forum⁶ or contact the Glyphs team directly.⁷

17.3 PLUG-INS

Plug-ins add a variety of capabilities and functionalities to Glyphs. They can be grouped into the following types:

- ▶ **Reporter** plug-ins (file name suffix `.glyphsReporter`) draw additional information in Edit View and are controllable with a *Show ...* menu item at the bottom of the *View* menu.

4 [docu.glyphsapp.com](https://docs.glyphsapp.com)

5 glyphsapp.com/learn/recommendation:learn-python

6 forum.glyphsapp.com

7 glyphsapp.com/contact

- ▶ **Filter** plug-ins (‘.glyphsFilter’) process glyph layers. They are accessible from the *Filter* menu. Some can be run using the ‘Filter’ custom parameter (see section 12.1.2, ‘Filters as Custom Parameters’, p. 177).
- ▶ **Palette** plug-ins (‘.glyphsPalette’) add entries to the Palette (*Window > Palette*, Cmd-Shift-P). See chapter 5, ‘Palette’ (p. 60) for details.
- ▶ **File Format** plug-ins (‘.glyphsFileFormat’) add support for opening and exporting additional file formats.
- ▶ **Tool** plug-ins (‘.glyphsTool’) add additional tools to the toolbar. These tools extend Edit View with new features.
- ▶ **General** plug-ins (‘.glyphsPlugin’) add functionality not covered by one of the above types.

17.3.1 Installing Plug-ins

Install plug-ins from the Plugin Manager (see p. 235) with a single click. Plug-ins are loaded when Glyphs launches, so Glyphs needs to be relaunched for newly installed plug-ins to be loaded. Some plug-ins require specific modules to be installed; install these from the *Modules* tab in the Plugin Manager.

Manually install a plug-in by dragging and dropping it onto the Glyphs app icon in the Dock. Installed plugins are moved to the Plugins folder. The Plugins folder is located next to the Scripts folder (see p. 236). *Do not* manually move plug-ins to the Plugins folder since that interferes with the security system of the Mac. Uninstall a plug-in by deleting it from the Plugins folder.

Plug-ins installed from the Plugin Manager can be uninstalled there, too.

17.3.2 Creating Plug-ins

Plug-ins are written in Python or Objective-C using the Glyphs SDK.⁸ The Glyphs SDK (Software Development Kit) is an open-source set of templates for creating Glyphs plug-ins. The templates are available for both Python and Objective-C.

The *Writing Plug-ins* tutorial⁹ introduces the basic concepts of plug-in development. See the Glyphs Python API documentation¹⁰ and Glyphs Objective-C API documentation¹¹ for a complete reference. If any questions arise, feel free to ask

8 github.com/schriftgestalt/GlyphsSDK

9 glyphsapp.com/learn/plugins

10 docu.glyphsapp.com

11 docu.glyphsapp.com/Core/

on the Glyphs forum¹² or contact the Glyphs team directly.¹³

The Glyphs SDK also offers methods for interoperating with Glyphs from an external application. In addition to the SDK, Glyphs supports the Open Scripting Architecture, allowing for automation via AppleScript or JavaScript.

¹² forum.glyphsapp.com

¹³ glyphsapp.com/contact

18 Appendix

18.1 REGULAR EXPRESSIONS

Regular expressions (often abbreviated as *regex*) can be used in some search fields to search for text patterns. For example, `[LlDd]caron` matches any of the following: ‘Lcaron’, ‘lcaron’, ‘Dcaron’, and ‘dcaron’. Characters that have a special meaning in patterns (such as `.` or `?`) can be found by prefixing them with a backslash `\`, for example, `\.` or `\?`.

`.` Matches any character (a, ., -, a space, ...).

`[abc]` Matches any character within the square brackets. For example, `[ae]acute` matches both ‘aacute’ and ‘eacute’.

`[^abc]` Matches any character that is not within the square brackets. For example, `[^au]-cy` matches ‘e-cy’ and ‘o-cy’ but not ‘a-cy’ or ‘u-cy’.

`\d` Matches any digit: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. `\D` matches any character except for digits. For example, `A.\00\d` matches ‘A.000’ and ‘A.003’ but not ‘A.00F’ or ‘A.123’.

`\w` Matches *word* characters: letters, digits, and the underscore (`_`). This is the same as writing `[a-zA-Z0-9_]`. For example, `\w+` finds ‘a’ and ‘alpha’, but not ‘a-cy’. `\W` matches any character except for word characters.

`?` Matches the preceding pattern zero or one time. For example, `oe?` finds both ‘o’ and ‘oe’; `[ae]?breve` finds ‘abreve’, ‘ebreve’, and ‘breve’; and `grave(comb)?` finds both ‘grave’ and ‘gravecomb’.

`*` Matches the preceding pattern zero or multiple times. For example, `A.*` finds ‘A’, ‘AE’, ‘Atilde’, and ‘Alpha’.

`+` Matches the preceding pattern one or multiple times. For example, `A.+` finds ‘AE’, ‘Atilde’, and ‘Alpha’, but not ‘A’.

`{n}` Matches the preceding pattern *n* times. For example, `a{5}` only finds ‘aaaaa’ and `\d{3}` finds ‘000’ and ‘123’ but not ‘12’ or ‘1234’.

`{n,m}` Like `{n}`, but finds the preceding pattern between *n* and *m* times. *n* may be 0 and *m* may be left off not to set an upper limit. `a{0,1}` is the same as `a?`; `a{0,}` is the same as `a*`; and `a{1,}` is the same as `a+`. For example, `\d{3,5}` finds

'123', '1234', and '12345', but not '12' or '123456'.

Useful patterns include:

a.* Matches any glyph that starts with 'a'.

.*a Matches any glyph that ends with 'a'.

.*-.* Matches any glyph that contains a hyphen.

.*\.* Matches any glyph that contains a dot.

.*\.\d+ Matches any glyph that ends in a dot followed by digits.


\D*\.\d{3} Matches any glyph that ends in a dot and three digits and does not contain any digits before that.

When finding and replacing with regular expressions, use '\1', '\2', ... to insert parts of the found pattern in the replacement.

The parts must be enclosed in round parenthesis in the find pattern. In the replacement pattern, use a backslash followed by the number referencing the parenthesized part.

For example, search for '(\d)(\d)' and replace it with '\2\1' to swap two digits ('15' becomes '51' or '03' becomes '30'). See also section 6.6.1, 'Search Field' (p. 85).

18.2 CUSTOM FEATURE CODE SNIPPETS

When editing OpenType layout feature code (see p. 103), click the snippets  button located in the bottom-right of the window. Choose *Show Snippet Folder* to reveal the folder in which additional snippets can be placed. Snippets are defined in property list files (.plist) named 'FeatureSnippetsNAME.plist' where 'NAME' is the name of the snippets group. For example, create a file named 'FeatureSnippetsDecomposition.plist':

```
(
  {
    title = "Soft_Dotted";
    code = "lookupflag_UseMarkFilteringSet_U@TopMark;
sub_U@SoftDotted'_U@TopMark_by_U@Dotless;";
  },
  {
    title = "Decompose_Precomposed";
    code = "sub_U@Precomp'_Ulookup_decomp_U@Mark;";
  }
)
```

18.3 AUTOMATIC FEATURE GENERATION

Glyphs can automatically generate many OpenType features if it finds glyphs with certain names in the font.

aalt	<i>All Alternatives</i>	Glyphs automatically builds the ‘aalt’ feature based on all features that substitute glyphs.
liga	<i>Standard Ligatures</i>	Join the glyph names of the components with an underscore (‘_’). Some common ligatures (f_f_i, f_f_l, f_f, fi, fl, lu_lakkhangyao-thai, ru_lakkhangyao-thai) are automatically placed in the ‘liga’ feature; all others go into ‘dlig’ by default. Force a ligature into the ‘liga’ feature by adding the ‘.liga’ suffix to its name, for example, f_b.liga or yi_yi-cy.liga.
dlig	<i>Discretionary Ligatures</i>	Join the glyph names of the components with an underscore (‘_’), for example, f_odieresis.
hlig	<i>Historical Ligatures</i>	Ligatures with longs, or with a ‘.hlig’ or ‘.hist’ suffix.
rlig	<i>Required Ligatures</i>	Add ‘.rlig’ to the ligature name. Also triggered by lam_alef-ar, lam_alefHamzaabove-ar, lam_alefHamzabelow-ar, lam_alefMadda-ar, lam_alef-ar.fina, lam_alefHamzaabove-ar.fina, lam_alefHamzabelow-ar.fina, lam_alefMadda-ar.fina, lam_alefWasla-ar, lam_alefWasla-ar.fina.
c2sc	<i>Small Capitals from Capitals</i>	Add ‘.sc’, ‘.c2sc’, or ‘.smcp’ to the glyph name. For separate sets for ‘c2sc’ and ‘smcp’, use ‘.c2sc’ for uppercase glyph names and ‘.smcp’ for lowercase glyph names.
smcp	<i>Small Capitals</i>	Add ‘.sc’ or ‘.smcp’ to the glyph name.
c2pc	<i>Petite Capitals from Capitals</i>	Add ‘.pc’, ‘.c2pc’, or ‘.pcap’ to the glyph name.
pcap	<i>Petite Capitals</i>	Add ‘.pc’ or ‘.pcap’ to the glyph name.
sup	<i>Superscript</i>	Add ‘.sup’ to the glyph name. Or extend figure names with superior, without the period, for example, onesuperior.
sub	<i>Subscript</i>	Add inferior, ‘.sinf’ or ‘.subs’ to the glyph name. If the font does not differentiate between subscript and scientific inferior, use one set of ‘.subs’ glyphs, and Glyphs will create both features with it.
sinf	<i>Scientific Inferiors</i>	Add inferior, ‘.sinf’ or ‘.subs’ to the glyph name.

afrc	<i>Alternative Fractions</i>	Add figures, slash, and any of these nut fraction glyphs to the font: oneovertwo, zerooverthree, oneoverthree, twooverthree, oneoverfour, threeoverfour, oneoverfive, twooverfive, threeoverfive, fouroverfive, oneoversix, fiveoversix, oneoverseven, twooverseven, threeoverseven, fouroverseven, fiveoverseven, sixoverseven, oneovertime, threeovertime, fiveovertime, sevenovertime, oneovernine, twoovernine, fourovernine, fiveovernine, sevenovernine, eightovernine, oneoverten, threeoverten, sevenoverten, nineoverten, oneovereleven, twoovereleven, threeovereleven, fourovereleven, fiveovereleven, sixovereleven, sevenovereleven, eightovereleven, nineovereleven, tenovereleven, oneovertwelve, fiveovertwelve, sevenovertwelve, elevenovertwelve, oneoveronehundred.
frac	<i>Fractions</i>	The frac feature is generated from the ‘.numr’, ‘.dnom’, and fraction glyphs. If they are not present in the font, then the feature will be composed of any pre-built fractions mentioned in the ‘afrc’ above.
dnom	<i>Denominators</i>	Add ‘.dnom’ to the glyph name.
numr	<i>Numerators</i>	Add ‘.numr’ to the glyph name.
onum	<i>Oldstyle Figures</i>	Add ‘.osf’ (for proportional oldstyle figures) or ‘.tosf’ (for tabular oldstyle figures) to the glyph name. Can be applied to other characters as well, for example, currency signs.
tnum	<i>Tabular Figures</i>	Add ‘.tf’ (for tabular figures) or ‘.tosf’ (for tabular oldstyle figures) to the glyph name. Can be applied to other characters as well, for example, currency signs. Do not use the figure suffix for the default figures; for example, if the default figures are proportional oldstyle figures, do not use figures with a ‘.osf’ suffix.
pnum	<i>Proportional Figures</i>	Add ‘.osf’ (for proportional oldstyle figures) or ‘.lf’ (for proportional lining figures) to the glyph name. Can be applied to other characters as well, for example, currency signs.
lnum	<i>Lining Figures</i>	Add ‘.lf’ (for proportional lining figures) or ‘.tf’ (for tabular figures) to the glyph name. Can be applied to other characters as well, for example, currency signs.
ordn	<i>Ordinals</i>	Automatically built if default figures, numero, ordfeminine, and ordmasculine are found in the font.
ornm	<i>Ornaments</i>	Add ‘.ornm’ to the glyph name of letters A-Z or a-z. Also, make sure the bullet glyph is in the font.

hist	<i>Historical Forms</i>	Add ‘.hist’ to the glyph name.
case	<i>Uppercase Forms</i>	Add ‘.case’ to the glyph name or ‘.lf’ to the name of a figure.
cpSP	<i>Capital Spacing</i>	Add uppercase letters to the font, and choose <i>Capital Spacing</i> from the plus (+) menu in the bottom-left of <i>File > Font Info... > Features</i> .
locl	<i>Localized Forms</i>	<p>Add ‘.loclXXX’ to the glyph name, where ‘XXX’ represents the three letter language tag,¹ for example, ‘.loclENG’ for English, or ‘.loclSVE’ for Swedish. There is also a built-in list of glyphs that trigger localizations:</p> <ul style="list-style-type: none"> ▶ idotaccent, i.TRK or i.loclTRK: trigger i substitutions for TRK, AZE, CRT, KAZ, and TAT if i is present. ▶ Scommaccent, Tcommaaccent, scommaccent, and tcommaaccent: trigger substitutions for ROM and MOL if Scedilla, Tcedilla, scedilla, and tcedilla are present. ▶ periodcentered.loclCAT (add a ‘.case’ suffix for uppercase; L_periodcentered_L.loclCAT, L_periodcentered_L.loclCAT; or the legacy Ldot, ldot: trigger ella geminada (punt volat) substitutions for CAT if L, l, and periodcentered are also present in the font. ▶ Iacute_J.loclNLD and iacute_j.loclNLD, or Jacute and jacute: trigger accented ij substitutions for NLD if Iacute, iacute, J, and j are present. ▶ six-ar and numbers with ‘.urdu’ suffix (for example, four-persian.urdu) will trigger URD localization for Persian.
cv01– cv99	<i>Character Variants</i>	<p>Add ‘.cv01’ through ‘.cv99’ to the glyph name. Note that the OpenType spec² recommends grouping related alternative glyphs into the same feature. Use manual feature code if there are multiple alternatives for a glyph; for example, place the following code in a single ‘cv##’ feature:</p> <pre>sub a from [a.alt1 a.alt2 a.alt3].</pre>
ss01– ss20	<i>Stylistic Set</i>	Add ‘.ss01’ through ‘.ss20’ to the glyph name. Stylistic sets can be named (see p. 105).
salt	<i>Stylistic Alternates</i>	By default, Glyphs will duplicate the ‘ss01’ feature in ‘salt’. Adobe Illustrator and Adobe Photoshop make use of this feature in their OpenType palettes.

1 docs.microsoft.com/typography/opentype/spec/language-tags

2 docs.microsoft.com/typography/opentype/spec/features-ae

swsh	<i>Swashes</i>	Add ‘.swsh’ to the glyph name.
titl	<i>Titling</i>	Add ‘.titl’ to the glyph name.
init	<i>Initial Forms</i>	Add ‘.init’ to the glyph name.
medi	<i>Medial Forms</i>	Add ‘.medi’ to the glyph name.
med2	<i>Medial Forms</i>	Add ‘.med2’ to the glyph name. Used only with the Syriac script.
fin	<i>Terminal Forms</i>	Add ‘.fin’ to the glyph name.
fin2	<i>Terminal Forms</i>	Add ‘.fin2’ to the glyph name. Used only with the Syriac script.
fin3	<i>Terminal Forms</i>	Add ‘.fin3’ to the glyph name. Used only with the Syriac script.
fwid	<i>Full Widths</i>	Add ‘.full’ or ‘.fullwidth’ to the glyph name, and emspace will substitute space.
hkna	<i>Horizontal Kana Alternates</i>	Add ‘.hori’ to a Katakana or Hiragana glyph name.
hojo	<i>Hoyo Kanji Forms</i>	Add ‘.hojo’ to the glyph name. Use it for JIS X 0212–1990 Kanji Forms. The respective unsuffixed default glyphs should be JIS X 0213: 2004 shapes.
hwid	<i>Half Widths</i>	Add ‘.half’ to the glyph name, end emspace will substitute space.
ital	<i>Italics</i>	Add ‘.italic’ to the glyph name.
pkna	<i>Proportional Kana</i>	Add ‘.proportional’ to a Katakana or Hiragana glyph name.
pwid	<i>Proportional Widths</i>	Add ‘.proportional’ to the glyph name.
qwid	<i>Quarter Widths</i>	Add ‘.qartwidth’ to the glyph name.
ruby	<i>Ruby Notation Forms</i>	Add ‘.ruby’ to the glyph name.
twid	<i>Third Widths</i>	Add ‘.thirdwidth’ to the glyph name.
vkna	<i>Vertical Kana Alternates</i>	Add ‘.vert’ to a Katakana or Hiragana glyph name.
vrt2	<i>Vertical Alternates and Rotation</i>	Add ‘.vert’ to the glyph name.
abvs	<i>Above-base Substitutions</i>	Triggered by certain base and mark combinations in South Asian scripts.
akhn	<i>Akhands</i>	Triggered by a number of letter and ligature glyphs in South Asian scripts.
blwf	<i>Below Base Forms</i>	Triggered by halants and ‘.below’ glyphs in combination with a range of other glyphs in South Asian scripts.

blws	<i>Below-base Substitutions</i>	Triggered by certain letter combinations in South Asian scripts, such as Gujarati letters with a ‘.straight’ suffix or specific halant combinations in Oriya.
cjct	<i>Conjunct Forms</i>	Triggered by conjunct clusters in South Asian scripts.
half	<i>Half Forms</i>	Triggered by half-form glyphs ending in Halfform in conjunction with halant in South Asian scripts.
nukt	<i>Nukta Forms</i>	Triggered by nukta ligatures ending in Nukta and the script abbreviation, in conjunction with the same glyphs without nukta, in South Asian scripts.
pref	<i>Pre-base Forms</i>	Triggered by a halant in certain letter constellations in South and Southeast Asian scripts.
pres	<i>Pre-base Substitutions</i>	Triggered by certain matra and conjunct constellations in South Asian scripts. Define width variants with number suffixes, for example, iMatra-deva.01 through ‘iMatra-deva.20’, and the feature generator will pick the appropriate width variant for each conjunct.
psts	<i>Post-base Substitutions</i>	Triggered by a halant glyph in combination with script-specific base and matra constellations, such as eMatra-kannada with ‘.base’ and ‘.base.e’ glyphs in Kannada. Applies to South Asian scripts only.
rkrf	<i>Rakar Forms</i>	Triggered by rakar ligatures in conjunction with the isolated glyphs and halant in Devanagari, Gujarati, and Malayalam.
rphf	<i>Reph Forms</i>	Triggered by ra-deva, halant-deva and reph-deva, or an analogous glyph structure in other South Asian scripts.
vatv	<i>Vattu Variants</i>	Triggered by the presence of specific lookups in the ‘rkrf’ feature.
ccmp	<i>Glyph Composition and Decomposition</i>	A wide range of glyph constellations in various scripts will trigger the automatic creation of ‘ccmp’. For instance, idotless and jdotless next to i, j and combining top marks will trigger a ‘ccmp’ lookup for Latin, which replaces the dotted with the dotless glyphs before top marks.
mark	<i>Mark to Base Positioning</i>	Add anchors without underscores to base letters, like <code>top</code> or <code>bottom</code> , and add combining marks (for example, acutecomb) with underscore anchors (for example, <code>_top</code> or <code>_bottom</code>) to glyphs in the font. Their width is automatically set to zero at export. This GPOS feature is not added to the <i>Features</i> tab but calculated at export (see p. 105).

Mark to Mark Positioning Add combining marks (U+0300 and above) with both underscore (for example, `_top`) and non-underscore anchors (for example, `top`) to glyphs in the font. This GPOS feature is not added to the Features tab but calculated at export (see p. 105).

18.4 CUSTOM PARAMETERS

Custom parameters provide additional settings and are identified by their property and have a value. In this appendix, the properties are printed in bold. The short description next to it explains the respective values and the function of the parameter. Parameters can be added to the *Custom Parameters* field of the Font, Masters, and Exports tab of *File > Font Info* (Cmd-I).

Enabled parameters are displayed in black; disabled parameters are displayed in gray. Quickly disable a parameter by unchecking the checkbox to the left of its property name.

Custom parameters in camelcase ('exampleName') are defined in the UFO specification and change the font information. In contrast, capitalized ones ('Example Name') are specific to Glyphs and usually change something in the font (for instance, run a filter on the outlines). UFO parameters follow the naming convention for Font Info properties outlined in the UFO 3 specification published in March 2012. Glyphs also makes use of a simplified naming convention. Wherever possible, leave out the prefix of the keyword. For example, instead of 'openTypeNameDescription', use 'description', or 'blueScale' instead of 'postscriptBlueScale'. Both long and short versions work side by side.

Add Class *string* Add an OpenType class to the font. The first word must be the class name (without the at sign), followed by a semicolon, and the new class code.

Add Feature *string* Add an OpenType feature to the font. The first word must be the feature tag, followed by a semicolon, and the new feature code. It will be put at the end of the list, so it may not be in the right order.

Add Prefix *string* Add an OpenType prefix to the font. The first word must be the prefix name, followed by a semicolon, and the new prefix code.

ascender *string* Overrides the Ascender value for a certain script. The values will be preferred for glyphs associated with the script in the Edit view. As value, enter the lowercase name of the script,

followed by a colon and the value, e.g., `cyrillic: 720`. Use multiple parameters for multiple scripts.

Autohint *boolean* Forces autohinting for the given instance, regardless of the setting in the Export dialog

Axis Location *list* Remaps a master or instance from the internal design space coordinates to values that are exposed in a variable font user interface (‘external coordinates’). This can be useful if your internal design space coordinates do not already adhere to the suggestions of the OpenType specification.

Axis Mappings Creates an `avar` table by mapping internal design coordinates to external coordinates exposed to the user. Put the internal coordinate into the left column, and the external coordinate in the right column. Or, add and edit values by clicking and dragging in the chart. The range for the internal coordinates is determined by the actual axis positions of the masters in *Font Info > Masters*, whereas the range for external coordinates is determined by the *Axis Location* parameters in *Font Info > Masters*.

Note that axis mappings work only *per axis*, and cannot be adjusted for different positions of *other* axes. This is a limitation of the `avar` implementation, so you may have to compromise on one of the mappings or pick an average value as the target mapping.

blueFuzz *integer* BlueFuzz value. This corresponds to the Type 1/CFF BlueFuzz field. From the Type 1 specification: The number specifies the ‘units to extend (in both directions) the effect of an alignment zone on a horizontal stem. If the top of a horizontal stem is within BlueFuzz units (in character space) outside of a top-zone, the interpreter will act as if the stem top were actually within the zone; the same holds for the bottoms of horizontal stems in bottom-zones. The default value of BlueFuzz is 1. Adobe themselves state that BlueFuzz was intended to compensate ‘for slightly inaccurate coordinate data.’ They therefore suggest adjusting the alignment zones themselves as well as explicitly setting BlueFuzz to zero. ‘Because a non-zero value for BlueFuzz extends the range of alignment zones, alignment zones must be declared at least $(2 \times \text{BlueFuzz} + 1)$ units apart from each other. Therefore, a default BlueFuzz value of 1 implies that alignment zones should be at least 3 units apart from each other.’

blueScale *float* PostScript BlueScale value. This corresponds to the Type 1/CFF BlueScale field. Controls the font size until which overshoot display is suppressed. Calculated as $(\text{pointsize at 300 dpi} - 0.49) \div 240$, e.g., 0.039625 for 10 points at 300 dpi. If you do not set the value yourself, blueScale defaults to 0.037, which corresponds to 9.37 points at 300 dpi or 39 pixels per em. This means that, in this case, overshoots will be visible if at least 40 pixels are used to display an em. The maximum blueScale value depends on the sizes of your alignment zones. The maximum pointsize at 300 dpi is calculated as $0.49 + 240 \div \text{largest alignment zone size}$, which corresponds to a PPM (size in pixels per em) of $2.04 + 1000 \div \text{largest alignment zone size}$. The product of $(\text{maximum pointsize} - 0.49) \times (\text{largest alignment zone height})$ must be less than 240.

For example, your largest zone is 21 units deep, thus: $2.04 + 1000 \div 21 = 49.659$, so the maximum PPM at which overshoots can be suppressed is 49. The corresponding maximum pointsize is $0.49 + 240 \div 21 = 11.919$ points at 300 dpi, thus the blueScale value cannot exceed $(11.919 - 0.49) \div 240 = 0.04762$.

blueShift *integer* PostScript BlueShift value. This corresponds to the Type 1/CFF BlueShift field. Default value is 7. Extends for very small glyph features beyond the font size indicated by blueScale. Overshoots inside an alignment zone are displayed if: (a) they are equal to or larger than BlueShift and (b) if they are smaller than BlueShift but larger than half a pixel. E.g. blueScale is set to suppress overshoots until 32 PPM, blueShift is 6, overshoots are 12 units deep. The stroke endings are slightly slanted and extend 5 units below the baseline. Between 0 and 32 PPM, the baseline will be kept completely level. Starting at 33 PPM, the overshoots will kick in. But the stroke endings will stay flat, because 5 units do not cover half a pixel until 100 PPM.

capHeight *string* Overrides the Cap Height value for a certain script. The values will be preferred for glyphs associated with the script in the Edit view. As value, enter the lowercase name of the script, followed by a colon and the value, e.g., `adLam: 680`. Use multiple parameters for multiple scripts.

CJK Grid *integer* Number of rows and columns of a dotted-line grid displayed when editing CJK glyphs. You can set number of rows and columns separately with the *CJK Grid Horizontal* and *CJK Grid Vertical* parameters. No grid is displayed when none of these parameters are set. This parameter can be localized like the

CJK Guide parameter.

CJK Grid Horizontal *integer* Number of columns of a dotted-line grid displayed when editing CJK glyphs. This parameter can be localized like the *CJK Guide* parameter.

CJK Grid Vertical *integer* Number of rows of a dotted-line grid displayed when editing CJK glyphs. This parameter can be localized like the *CJK Guide* parameter.

CJK Guide *string* Percentage of inset for CJK guide squares, e.g., 10 for 10 percent. If set, Glyphs will display a second square guide for the virtual body in CJK glyphs. You can localize the parameter by preceding the value with the script name, e.g., *kana:5*. If you want to define virtual bodies for more than one script, add more *CJK Guide* parameters.

codePageRanges *list* Sets the appropriate bits of the `uLCodePageRange1` and `uLCodePageRange2` entries in the `OS/2` table. ‘This field is used to specify the code pages encompassed by the font file in the cmap subtable for platform 3, encoding ID 1 (Microsoft platform).’ Every activated ‘code page is considered functional. Each of the bits is treated as an independent flag and the bits can be set in any combination. The determination of “functional” is left up to the font designer, although character set selection should attempt to be functional by code pages if at all possible.’

Color Layers to COLR *boolean* If turned on, will produce `COLR` and `CPAL` tables with the color information stored in a color setup, from the masters that have a `Master Color` parameter.

Color Layers to SVG *boolean* If turned on, will produce an `SVG` table with the color information stored in `CPAL` Color layers, or, in a layered-font setup, from the masters that have a `Master Color` parameter.

Color Palette for CPAL *integer* Index of the color palette (as defined in the `Color Palettes` parameter) that is supposed to be used for the `CPAL` OpenType table.

Color Palette for SVG *integer* Index of the color palette (as defined in the `Color Palettes` parameter) that is supposed to be used for the `SVG` OpenType table. Useful only in conjunction with the `Color Layers to SVG` parameter.

Color Palettes *list* Define color palettes for export as a `CPAL` OpenType table in ‘Microsoft-style’ `CPAL`/`COLR` color fonts. This

parameter allows Glyphs to display a preview of color glyphs in Font and Edit view.

Compatible name Table *boolean* Exports a legacy *name* table as expected by some Mac apps (in particular Quark XPress and FontExplorer). It improves grouping of font styles in the font menu of those apps, but may break functionality elsewhere, especially Microsoft Office. Therefore, caution is advised in the use of this parameter.

compatibleFullNames Sets *name* table ID 18, ‘Compatible Full Name’, a Mac-only name which is intended to be preferred over ID 4 (‘Full Name’). If not set, the value for *name* table ID 18 is calculated from Family Name plus space plus Style Name of the respective instance. ‘On the Macintosh, the menu name is constructed using the FOND resource. This usually matches the Full Name. If you want the name of the font to appear differently than the Full Name, you can insert the Compatible Full Name in ID 18.’ **Caution:** since this may lead to a situation where Mac and Windows use different full names for the same font, it may break cross-platform interoperability of documents.

copyrights Copyright statement. Overrides the entry in the Copyright field in the Font tab of the Font Info. Corresponds to the OpenType *name* table ID 0, ‘Copyright notice’.

cvt Table *string* List of values for the *cvt* (control value table). When you open a TTF in Glyphs, the application will store the existing *cvt* values in this parameter. This is intended to preserve existing TT hinting for reexporting, and certainly not to be edited manually. Remove this parameter if you want to do your own TrueType instructioning.

Decompose Brace Layers *boolean* In variable font exports, decompose components that point to glyphs with intermediate layers a.k.a. ‘brace layers’. Default is on.

Decompose Components *boolean* Decompose all components on export. Useful for situations where the environment cannot handle components correctly, such as eraly implementations of Variable Fonts in legacy macOS versions. Will increase file size. Only makes sense for TrueType fonts; CFF-based fonts will always decompose.

Decompose Components in Variable Font *boolean* When exporting a (TT-based) variable font, all components will be decomposed to outlines. This option is intended to circumvent a CoreText

rendering bug in macOS versions prior to Mojave 10.14.5, which affected the spacing of composite glyphs with changing LSB values under certain circumstances. Since the resulting variable font contains no more components, its file size can be expected to increase dramatically.

Decompose Glyphs *list* At export, decomposes the composite glyphs listed. This can be useful if you want to avoid changing of composites when one of the components is being changed with the `Rename Glyphs` parameter.

Default Layer Width *integer* Newly created glyphs and layers will have the specified width rather than the default 600 units, or 1000 units for CJK glyphs, or 200 units for corner components. If you prefix the value with a (lowercase) script name and a colon, the specified layer width will only be applied to glyphs of that script, e.g., `cyrillic: 400` or `adlam: 450`. Add multiple parameters for multiple scripts.

descender *string* Overrides the Descender value for a certain script. The values will be preferred for glyphs associated with the script in the Edit view. As value, enter the lowercase name of the script, followed by a colon and the value, e.g., `arabic: -300`. Use multiple parameters for multiple scripts.

descriptions Description of the font. Corresponds to the OpenType `name` table ID 10: ‘description of the typeface. Can contain revision information, usage recommendations, history, features, etc.’

Design Grid *string* Number of rows and columns of a dotted-line grid. Mostly used for CJK design. It can take three values:
`[script:]horizontal,vertical`

designers Name of the designer. Overrides the `Font Info > Font > Designer` entry for the given instance. Corresponds to the OpenType `name` table ID 9, ‘name of the designer of the typeface.’

designerURL *string* The URL of the designer. Overrides the `Font Info > Font > Designer URL` entry for the given instance. Corresponds to the OpenType `name` table ID 12, ‘URL of typeface designer (with protocol, e.g., `http://`, `ftp://`).’

Disable autohinting for glyphs *list* Excludes listed glyphs from the PostScript autohinting at export time. (TT autohinting cannot be disabled on a per-glyph basis.) This can be useful if some glyphs do not lend themselves for hinting, e.g., ornaments.

Disable Masters *list* Disables all specified masters. Intended mainly for specific production workflows, or for testing purposes, e.g., to see if interpolation still behaves as expected if you leave out one of the intermediate masters.

Disable Subroutines *boolean* If set, CFF outline subroutinization is disabled when the font is exported. Use this (a) when the font has complex outlines with many nodes and does not export at all, or (b) for testing purposes when the font has many glyphs, e.g., a CJK font, and takes too long to compile every time you export.

DisableAllAutomaticBehaviour *boolean* Will disable most of the automatic stuff on export.

- ▶ reordering `.notdef` and `space`
- ▶ zero widths for nonspacing marks

Don't use Production Names *boolean* If checked, Glyphs will not automatically rename glyphs of the final font file according to the internal glyph database, but export the current glyph names. Some applications and systems, amongst which the text engine of OS X 10.4, expect the AGL naming scheme, though. This is equivalent to the *File > Font Info > Other Settings > Use Custom Naming* setting, and intended for users who want to employ their own custom naming scheme.

EditView Line Height *float* Sets the line height for text set in an Edit tab. Useful if you have unusual vertical metrics, and the default leading seems inappropriate. Has no effect on the exported font file.

Elidable STAT Axis Value Name *string* Declares the style name of the instance as elidable for the axis specified in the parameter's value. As value, use the four-letter tag of the respective axis. Typically, you will add this parameter in the regular style, and you will add one for each axis for which the name is an elidable display string. Example: In a two-axis setup, an instance called Regular has two Elidable STAT Axis Value Name parameters, one with `wght` and one with `wdth` as parameter values. A display string is deemed elidable if it is left out when combined with other display strings. Usually this is the case for default-style names like 'Regular', 'Normal', or the like. The semibold weight combined with regular width is usually just called 'Semibold', not 'Semibold Regular'; or the normal weight in combination with the italic style is simply called 'Italic', not 'Regular Italic'. Thus, the display name 'Regular' is considered elidable.

Enforce Compatibility Check *boolean* Forces compatibility checks between all masters. Compatibility is only checked for a pair of masters if they are both required for interpolating one of the instances defined in *File > Font Info > Instances*. Useful if you have no instance between certain masters, but still need to keep them compatible, e.g., for a variable font.

Export AAT *boolean* Toggles the export of Apple Advanced Typography (AAT) instructions as entered in *File > Font Info > Features > Prefix > morx*. If *morx* instructions are present in Font Info, they will be compiled into a *morx* table by default. This parameter is intended as an option to prevent its export, and thus create a pure OpenType font.

Export COLR Table *boolean* Toggles the export of *CPAL* and *COLR* tables. If all conditions are met, *CPAL* and *COLR* tables will be compiled and exported by default, though. This parameter is intended as an option to prevent their export.

Export Folder *string* Adds a subfolder to the default export destination. Please make sure to use only characters that are valid in folder names.

Export Glyphs *list* Exports all glyphs listed, regardless of whether the glyph was set to export or not.

Export kern Table *boolean* On export, will write an old-style *kern* table in addition to the kern feature in the *GPOS* table. Only affects in TrueType exports (i.e., *.ttf* or TT-flavored webfonts). All group kerning will be expanded into all possible singleton pairs. This means that you will have to subset before using this parameter, otherwise you risk a table overflow. **Use of this parameter is strongly discouraged**, and only makes sense in rare edge cases where support of defunct or legacy software is necessary. **Only use it if you know what you are doing.**

Export Mac Name Table Entries *boolean* Control if Macintosh name table entries are exported (they are by default). So this is meant to disable the Macintosh names.

Export morx Table *boolean* Inserts a *morx* (‘extended glyph metamorphosis’) AAT table into the exported font. Takes the contents of a prefix with the name *morx* in *File > Font Info > Features*, written in MIF code (metamorphosis input file). For this to work, the *ftxenhancer* command line tool of the Apple Font Tools must be installed. For more information, refer to the documentation included with the Apple Font Tools.

- Export OpenType** *boolean* Toggles the export of most OpenType tables. If disabled, and if `morx` data is entered in *File > Font Info > Features > Prefix*, will export a pure AAT font.
- Export PostScript Hints** *boolean* If set, will force export of PostScript hinting (default). Can be used for suppressing PostScript hints when its boolean value is turned off.
- Export sbix Table** *boolean* Toggles the export of an `sbix` table. If all conditions are met, an `sbix` table will be compiled and exported by default, though. This parameter is intended as an option to prevent its export.
- Export STAT Table** *boolean* Toggles the export of the `STAT` table. A `STAT` table will be compiled and exported by default, though. This parameter is intended as an option to prevent its export.
- Export SVG Table** *boolean* Toggles the export of an `SVG` table. An `SVG` table will be compiled and exported by default, though. This parameter is intended as an option to prevent its export.
- Export TrueType Hints** *boolean* If set, will force export of TrueType hinting (default). Can be used for suppressing TrueType hints when its boolean value is turned off.
- Export vmtx Table** *boolean* Forces or suppresses the export of a `vmtx` ('Vertical Metrics') table. Can be useful if you have top and bottom sidebearings set in your glyphs, but want to export a horizontal variant of your font only. OpenType spec: `vmtx`³
- Family Alignment Zones** *list* This parameter can help create a more consistent screen appearance at low resolutions, even if the overshoots differ in the individual weights. It is a good idea to reduplicate the alignment zones of the most important font in your family, usually of the Regular or Book instance. A rasterizer will then try to align all weights if the height difference between the individual weight and the family alignment is less than one pixel. Important: For this mechanism to work, family alignment zones must be compatible with the alignment zones set up in the masters.
- familyNames** Font family name. Overrides the entry in the *Family Name* field in the Font section of the Font Info. Corresponds to the OpenType `name` table IDs 1 and 4. Used to calculate IDs 3, 4 and 6.
- Feature for Feature Variations** *string* Define the OpenType feature

³ <https://learn.microsoft.com/typography/opentype/spec/vmtx>

into which glyph variant substitutions (a.k.a. ‘Bracket tricks’) are written in OpenType variable fonts. Default is `rvarn`.

fileName *string* Name for the font file, without the dot suffix, i.e., without `.otf`, etc. Gives you the chance to export two versions of the same font style name without the second file overwriting the first one.

Filter *string* Triggers Glyphs filters or app functions in an instance, after decomposition of composite glyphs. The values for the built-in filters are as follows:

- ▶ **AddExtremes**
- ▶ **Autohinting**
- ▶ **HatchOutlineFilter**; OriginX: *x*; OriginY: *y*; StepWidth: *distance*; Angle: *angle*; Offset: *thickness*
- ▶ **OffsetCurve**; *x*; *y*; *makestroke*; *position/auto*
- ▶ **RemoveOverlap**
- ▶ **Roughenizer**; *segmentlength*; *x*; *y*; *angle*
- ▶ **RoundCorner**; *radius*; *visualcorrection*
- ▶ **RoundedFont**; *stem*
- ▶ **Transformations**; LSB: \pm *shift**; RSB: \pm *shift**; Width: \pm *shift*; ScaleX: *percent*; ScaleY: *percent*; Slant: *degrees*; SlantCorrection: *bool*; OffsetX: *amount*; OffsetY: *amount*; Origin: *select*

Replace all words in italics with your values: The boolean values (*makestroke*, *visual correction*, *bool*) are 1 for yes and 0 for no. The value for *position/auto* must be a floating point number where 0.0 represents 0%, and 1.0 stands for 100%, or the string `auto` for *Auto Stroke*. The *stem* value in the *RoundedFont* syntax is optional. The *select* value in the *Transformations* syntax can be a number from 0 to 4, representing the five options displayed in *Filter > Transformations > Transform > Origin*: cap height (0), half cap height (1), x-height (2), half x-height (3), baseline (4).

If you want a filter to be applied only to some glyphs, add `include:` or `exclude:`, followed by space- or comma-separated glyph names, e.g., `RemoveOverlap`; `exclude:a,b,c`.

If you are using third-party filters, refer to their documentation for the parameter string. In particular, the include and exclude options may not be available.

If you want to apply filters before decomposition, use these values with a `PreFilter` parameter.

Fit Curve Panel Settings Use this parameter to store the minimum and maximum percentages for the *Fit Curve* panel in the right sidebar. As parameter value, use two numbers, separated by a comma, e.g., '57, 72'. Once set, the parameter value will pick up changes you make in the *Fit Curve* UI.

fpgm Table Assembly *string* Assembly code for the `fpgm` (font program) table. When you open a TTF in Glyphs, the application will store the existing `fpgm` code in this parameter. This is intended to preserve existing TT hinting for reexporting, and certainly not to be edited manually. Remove this parameter if you want to do your own TrueType instructioning.

fsType *list* A list of bit numbers indicating the embedding type. The bit numbers are listed in the OpenType *OS/2* specification. Corresponds to the OpenType *OS/2* table `fsType` field. 'Type flags. Indicates font embedding licensing rights for the font. Embeddable fonts may be stored in a document. When a document with embedded fonts is opened on a system that does not have the font installed (the remote system), the embedded font may be loaded for temporary (and in some cases, permanent) use on that system by an embedding-aware application. Embedding licensing rights are granted by the vendor of the font. The OpenType Font Embedding DLL Specification and DLL release notes describe the APIs used to implement support for OpenType font embedding and loading. Applications that implement support for font embedding, either through use of the Font Embedding DLL or through other means, must not embed fonts which are not licensed to permit embedding. Further, applications loading embedded fonts for temporary use (see Preview & Print and Editable embedding below) must delete the fonts when the document containing the embedded font is closed.' You can set `fsType` to one of these five states:

- ▶ *Installable*: 'Fonts with this setting indicate that they may be embedded and permanently installed on the remote system by an application. The user of the remote system acquires the identical rights, obligations and licenses for that font as the original purchaser of the font, and is subject to the same end-user license agreement, copyright, design patent, and/or trademark as was the original purchaser.'
- ▶ *Forbidden*: 'Restricted License embedding: Fonts that have only this bit set must not be modified, embedded or exchanged in any

manner without first obtaining permission of the legal owner. Caution: For Restricted License embedding to take effect, it must be the only level of embedding selected.’

- ▶ *Editable*: ‘When this bit is set, the font may be embedded but must only be installed temporarily on other systems. In contrast to Preview & Print fonts, documents containing Editable fonts may be opened for reading, editing is permitted, and changes may be saved.’
- ▶ *Preview & Print*: ‘When this bit is set, the font may be embedded, and temporarily loaded on the remote system. Documents containing Preview & Print fonts must be opened “read-only;” no edits can be applied to the document.’
- ▶ *Subsetting forbidden*: ‘When this bit is set, the font may not be subsetted prior to embedding. Other embedding restrictions also apply.’

gasp Table Sets the `gasp` table (‘grid-fitting and scan-conversion procedure’) for TrueType fonts. It controls the two PPM thresholds at which the recommended on-screen rendering behavior changes. The `gasp` table contains rendering recommendations for both a traditional grayscale and a ClearType subpixel renderer. However, keep in mind that a renderer may ignore the data stored herein. ‘This table contains information which describes the preferred rasterization techniques for the typeface when it is rendered on grayscale-capable devices. This table also has some use for monochrome devices, which may use the table to turn off hinting at very large or small sizes, to improve performance.’ The default threshold sizes are 8 and 20 PPM. Because there are two thresholds, three ranges can be differentiated:

- ▶ *no hinting & symmetric*: Until the first threshold size, no gridfitting is applied, and text is rendered with antialiasing wherever possible. ‘At very small sizes, the best appearance on grayscale devices can usually be achieved by rendering the glyphs in grayscale without using hints.’
- ▶ *hinting & asymmetric*: Between the two threshold sizes, the renderer is recommended to apply gridfitting and suppress grayscale. ‘At intermediate sizes, hinting and monochrome rendering will usually produce the best appearance.’ In ClearType, the matter is handled asymmetrically, i.e., vertical gridfitting is applied, while horizontally, subpixel rendering is used.

- ▶ *hinting & symmetric*: Beyond the thresholds, the rasterizer is instructed to apply gridfitting and render the shapes in grayscale. ‘At large sizes, the combination of hinting and grayscale rendering will typically produce the best appearance.’ The ClearType rasterizer is instructed to apply symmetric smoothing, i.e., to use anti-aliasing in y direction in addition to horizontal subpixel rendering. ‘At display sizes on screen, [...] this new improvement of the Windows font renderer produces smoother and cleaner-looking type’ (Now Read this: The Microsoft ClearType Font Collection, Microsoft 2004, p. 14).

Get Hints From Master *string* Defines which master is taken as source of manual PS and TT hints, sometimes also referred to as the ‘main master’. Hinting in other masters will be ignored. If not set, manually entered hints will be taken from the first master listed in *File > Font Info > Masters*. This also affects treatment of TTFStems and TTFZones parameters: only in the indicated master, the UI for those parameters will provide the delta and range buttons.

glyphOrder *list* Sets the order of glyphs in both the working file and the final font. Glyph names need to be separated by newlines. You can copy and paste the content of a List Filter. Glyphs not listed but still in the font will be appended after listed glyphs, in the default order that Glyphs employs.

Grid Spacing *integer* Set the coordinate precision for the resulting CFF font, in font units. The value corresponds to the quotient of the Grid Spacing value divided by the Subdivision value in *File > Font Info > Other Settings*. At export time though, any Grid Spacing value smaller than 1.0 will result in coordinate precision a hundred times finer than the default unit-grid rounding. In other words, all parameter values smaller than 1 are equivalent to 0.01. The purposes of this parameter is to avoid too coarse rounding of point coordinates in very thin interpolations. Any Grid Spacing value equal to or larger than 1.0 will result in grid-unit rounding. In other words, it any value greater than 1.0 is equivalent to 1.0. The parameter has no effect on TrueType outlines, which cannot have higher coordinate precision.

Hangul Composition Groups *list* Defines composition groups for Hangul type design. Pick a key Jamo glyph and similarly formatted variants for automated composition of complex Hangul composites.

Has WWS Names *boolean* Sets bit 8 of the `fsSelection` entry in the

OS/2 table: According to the OpenType specification, this bit indicates that ‘the font has `name` table strings consistent with a weight/width/slope family without requiring use of “name” IDs 21 and 22.’ This makes sense only if the naming of your font already adheres to the WWS scheme.

hheaAscender *integer* Height of the ascender as stored in the `hhea` (horizontal header) table. ‘Typographic ascent (distance from baseline of highest ascender).’

The `hhea` vertical metrics are mainly in use on Mac apps, including browsers on the Mac. Therefore, unless you have to maintain backwards compatibility with legacy software, it is recommended to keep the `hhea` values in sync with the `typo*` values, and make sure that `Use Typo Metrics` (`fsSelection` bit 7) is switched on.

For a detailed discussion of vertical metrics, see the Vertical Metrics⁴ tutorial.

hheaDescender *integer* A *negative* integer describing the depth of the descender as stored in the `hhea` table. ‘Typographic descent (distance from baseline of lowest descender).’ For a discussion of the `hhea` values, see `hheaAscender`..

hheaLineGap *integer* The recommended interlinear whitespace as stored in the `hhea` table. ‘Typographic line gap. Negative values are treated as zero in some legacy implementations.’ For a discussion of the `hhea` values, see `hheaAscender`.

Ignore custom GlyphData file *boolean* If set, will ignore the content of a `GlyphData.xml` next to the `.glyphs` file, or in a `Info` subfolder next to the `.glyphs` file.

Ignore out of bounds instances *boolean* Ignore all instances that are outside of the design space (e.g. when subsetting).

Ignore Vertical Hints *boolean* In a TrueType export, will ignore all stem hints that are linked to a vertical stem definition, as set forth in a `TTFStems` parameter. Makes sense in exports intended for subpixel rendering, or for webfont exports where you want to keep the file size as small as possible. Only affects manual TT hinting, does not apply to TTF Autohint or PS hinting.

Import Font Reference a different `.glyphs` file, and all glyphs of the referenced file that are not in the host font, will be hot-linked, in the order of their layers and masters, as if you had copied the

⁴ <https://glyphsapp.com/tutorials/vertical-metrics>

glyphs from one file to the other. The glyphs will be visible and typeable in Font and Edit view, but locked. Useful for splitting design work between multiple users, or sharing smart components between files.

Import Master Reference a master in different .glyphs file, and it will be hot-linked into the host file, as if you had added it into *File > Font Info > Masters*. Interpolation values must be set compatibly in both files. Add one parameter for each master you want to hot-link.

Instance Preview *list* Changes the preview string of an instance in *File > Font Info > Instances* from the default 'Aang126' to the glyph names listed.

InterpolationWeightY *float* Vertical interpolation value. In an instance, you can differentiate between interpolation along the x axis and interpolation along the y axis by introducing this custom parameter. For it to take effect, it must differ from the interpolation weight of the instance. Be careful, as this can lead to deformation in diagonals. We advice to keep the InterpolationWeightY close to the normal Weight interpolation value.

E.g., there are two masters at weight 20 and 100, and an instance with a weight interpolation value of 50. The vertical stems look correct, but the horizontals look too thin. They would look right at 60, but then the verticals would appear too thick. So, you keep your instance at 50, but add the custom parameter InterpolationWeightY with a value of 60. Now, the vertical stems (x coordinates) are still interpolated with 50, and the horizontals (y coordinates) with 60.

isFixedPitch *boolean* Sets the `isFixedPitch` flag in the `post` table. Indicates whether the font is monospaced. Software can use this information to make sure that all glyphs are rendered with the same amount of pixels horizontally at any given PPM size.

⚠️ Danger: Will sync the width of all glyphs with the width of the space glyph This can be overwritten by adding a `.monospace` number value in the master settings.

Italic Style Linking *boolean* In the STAT table of variable fonts, adds a Format 3 (style linking) AxisValue for the Italic (*ital*) axis, linking the Regular (*ital*=0) to the Italic (*ital*=1). Should **not** be added to (exclusively) italic fonts. Use this parameter for forcing or suppressing the style linking rather than relying on the built-in

automation.

italicAngle *string* Overrides the Italic Angle value in *File > Font Info > Masters* either for a certain script or for the whole font. The values will be preferred for glyphs associated with the script in the Edit view. As value, enter the lowercase name of the script, followed by a colon and the value, e.g., `latin: 7`. Use multiple parameters for multiple scripts. The number must be an angle in clockwise degrees from the vertical. Also useful for upright fonts with an angle other than 0°, because macOS may interpret non-zero angles as italic. Affects the CFF ItalicAngle, the post italicAngle, the x offsets of the OS/2 subscript and superscript values, as well as the hhea caretSlopeRise and caretSlopeRun entries.

Keep Glyphs *list* List of glyph names for all glyphs that will be kept in the exported font. All other glyphs will be discarded, and kerning and automatic feature code will be updated accordingly. Useful for webfont subsetting in order to achieve smaller file sizes. Can use the same wildcards and category searches as the *Remove Glyphs* parameter. Uses of *Remove Glyphs* and *Keep Glyphs* are mutually exclusive.

Keep Kerning in one Lookup *boolean* Will attempt to fit all ‘pair adjustment’ kerning into the first GPOS lookup. This is a workaround for a bug in Microsoft apps where kerning outside the first lookup is ignored. If turned off, Glyphs will default to separating kerning into lookups based on the glyph categories.

Keep Overlapping Components *boolean* If set, Glyphs does not decompose composite glyphs with overlapping components, such as in Ccedilla. Useful for post-production of TrueType fonts. Has no effect on CFF exports.

Keep Transformed Components *boolean* Does not decompose composite glyphs with transformed (horizontally or vertically scaled, or vertically shifted) components. Useful for post-production of TrueType fonts. For release-ready production, we do not recommend using this parameter, as it may interfere with TT hinting in affected glyphs. The parameter has no effect on CFF exports.

Keep UI-Font Bounding Box *boolean* Prevents conjuncts from stacking below in South Asian scripts, by altering the automated feature code for the `cjct` feature accordingly. Useful for

intended use in environments where vertical stacking is limited, e.g., in user interfaces, hence the name. Currently only implemented for Oriya.

licenses License description. Corresponds to the OpenType `name` table ID 13, the ‘description of how the font may be legally used, or different example scenarios for licensed use. This field should be written in plain language, not legalese.’

licenseURL *string* URL for the license. Corresponds to the OpenType `name` table ID 14. ‘URL where additional licensing information can be found.’ Make sure it starts with the protocol specification, typically `https://`.

Link Metrics With First Master *boolean* If checked, keeps the side-bearings and the kerning of the respective master (in which the parameter is entered) in sync with the first master. In effect, you only have to space and kern the first master. This is especially useful for color fonts or fonts that should not change their metrics throughout their weights.

Link Metrics With Master *string* Same as Link Metrics With First Master (see above), except that it specifies the name of the master to which kerning and spacing is linked to. If you use this parameter, it is advisable to make sure that all masters have unique names.

Local Interpolation *string* Apply different interpolation values for specified glyphs. The string must contain a semicolon-separated list of interpolation values for each axis, followed by another semicolon, an `include:` statement and a comma-separated list of glyph names. For a single-axis setup, a single interpolation value suffices. E.g., `120; include: a, g, s` uses interpolation value 120 just for the glyphs a, g and s, while all other glyphs are interpolated according to the interpolation value of the respective instance.

MakeOTF Compatibility Mode *boolean* If checked, the font created by Glyphs will have its OpenType Layout tables comparable to that created by Adobe’s `makeotf` program. This can be useful when the feature code for those tables was written with specific `makeotf` behavior in mind, or if the font is meant to be used in applications that rely on `makeotf`-compatible output.

manufacturers Manufacturer Name. Overwrites the *Manufacturer* as set in *Font Info > Font*. Corresponds to the OpenType `name` table ID 8, ‘Manufacturer Name’.

manufacturerURL *string* Manufacturer or Vendor URL. Overwrites the *Manufacturer URL* as set in *Font Info > Font*. Corresponds to the OpenType *name* table ID 11, 'URL of font vendor (with protocol, e.g., http://, ftp://)'. If a unique serial number is embedded in the URL, it can be used to register the font.'

Master Background Color *string* Sets the canvas color of a master. The canvas assumes the specified color when the respective master is active in Edit view.

Master Background Color Dark *string* Like **Master Background Color**, but for Dark Mode.

Master Color *string* Color used for the display of the filled outline of the master in question. Useful also to preview layer fonts (multiple fonts intended to be set on top of each other with different colors). Only effective inside Glyphs, does not export into the OpenType font file.

Master Color Dark *string* Like **Master Color**, but for Dark Mode.

Master Icon Glyph Name *string* Name of the glyph that is to be used for the respective master button displayed in the top left corner of the font window when two or more masters are present in *File > Font Info > Masters*.

Master Stroke Color *string* Color used for the display of outlines of the master in question, visible when the Select All Layers tool (Shift-V) is active. Only effective inside Glyphs, does not export into the OpenType font file.

Master Stroke Color Dark *string* Like **Master Stroke Color**, but for Dark Mode.

meta Table *list* Add a meta ('Metadata') table to your font export:

- ▶ **dLng** (Design languages): languages or scripts of the primary user audiences for which the font was designed. This value may be useful for selecting default font formatting based on content language, for presenting filtered font options based on user language preferences, or similar applications involving the language or script of content or user settings.
- ▶ **sLng** (Supported languages): languages or scripts that the font is capable of supported. This value may be useful for font fallback mechanisms or other applications involving the language or script of content or user settings. The declarations provided by the **sLng** value should be the same as or a superset of those provided by **dLng**. Implementations that use **sLng** values in a font

may choose to ignore Unicode-range bits set in the OS/2 table. Each of the entries takes a comma-separated list of `ScriptLangTag` entries. Each `ScriptLangTag` is a hyphen-concatenated string for language, script and region (ISO 3166-1, Alpha-2), compliant to IETF BCP 47, but only script is required. E.g., `Cyrl` for Cyrillic, `sr-Cyrl` for Serbian written in Cyrillic, and `sr-Cyrl-BA` for Serbian written with Cyrillic in Bosnia and Herzegovina. OpenType spec: meta⁵ Languages, scripts and regions⁶

Name Table Entry *string* A custom entry for the OpenType name table. The syntax is one of the following three:

- ▶ `nameID; nameString`
- ▶ `nameID platformID; nameString`
- ▶ `nameID platformID encID langID; nameString` If not specified, `platformID` will be assumed as 3, and successively, `encID` as 1 (Unicode), and `langID` as 0x0049 (Windows English). If only `platformID` is specified as 1, then both `encID` and `langID` will be assumed as 0 (Mac Roman, and Mac English).

The `nameID` can be anything except 1, 2, 3, 5, and 6, which cannot be set through this parameter. The `platformID` can either be 1 for Macintosh or 3 for Windows. The optional `encID` and `langID` represent either Windows or Macintosh encoding and language IDs, depending on the `platformID`. They must be numbers between 0 and 65536, and can be entered in decimal, octal or hexadecimal form. The AFDKO Syntax specification stipulates that ‘decimal numbers must begin with a non-0 digit, octal numbers with a 0 digit, and hexadecimal numbers with a 0x prefix to numbers and hexadecimal letters a-f or A-F’

note *string* Arbitrary note about the font. This is not exported in the final OpenType font, only stored in the `.glyphs` file. Setting the font note as a custom parameter is equivalent to setting it in *File > Font Info > Notes*.

Optical Size *string* Builds the Optical Size OpenType feature (feature tag ‘size’), with encoded size menu names for Mac and Windows. Requires a string with five semicolon-separated values:

- ▶ *design size*: size in decipoints (tenths of points) the font was

⁵ <https://learn.microsoft.com/typography/opentype/spec/meta>

⁶ <https://www.iana.org/assignments/language-subtag-registry/language-subtag-registry>

designed for;

- ▶ *subfamily identifier*: arbitrary integer; different fonts with the same number can be grouped together in an optical size submenu, if the software supports it;
- ▶ *range start*: decipoint size of the size above which the font is supposed to be used for;
- ▶ *range end*: decipoint size of the size until (and including) which the font is supposed to be used for;
- ▶ *size menu name*: submenu name for the optical size, e.g., ‘Display’, ‘Subhead’, ‘Small’, or ‘Caption’.

Example: ‘100; 1; 69; 120; Ten’ will create a size feature that specifies 10 points as the intended design size, the range in which it is supposed to be used is 7 to 12 points, and the optical size name is ‘Ten’. Other fonts that use 1 as subfamily identifier and ‘Ten’ as name, can be grouped together.

Optimize Variable Deltas *boolean* Will drop OpenType Variation deltas from a contour if none of its nodes moves more than half a unit. Default is on. Set this parameter to off in order to also keep low-significant deltas.

panose *list* Once you click in the Value field, a dialog will appear that allows you to determine the setting for each category in the PANOSE specification. This corresponds to the ten ‘panose’ fields in the OpenType *OS/2* table. ‘This 10 byte series of numbers is used to describe the visual characteristics of a given typeface. These characteristics are then used to associate the font with other fonts of similar appearance having different names. [...] The PANOSE values are fully described in the PANOSE “greybook” reference, currently owned by Monotype Imaging. The PANOSE definition contains ten digits each of which currently describes up to sixteen variations. Windows uses bFamilyType, bSerifStyle and bProportion in the font mapper to determine family type. It also uses bProportion to determine if the font is monospaced. If the font is a symbol font, the first byte of the PANOSE number (bFamilyType) must be set to “pictorial.”’ At the time of this writing, PANOSE is not required to make a font work anywhere, and, to our knowledge, is hardly in use.

Point To Unit Ratio *float* Integer number that defines how many font units are equivalent to one DTP point. Determines the export scale of PDFs, including artwork copied into the clipboard.

Useful for exchanging vector data with third-party drawing apps, such as Affinity Designer or Sketch.

Post Table Type *integer* Version of the `post` table built into the instance, the default is 2 for TTF, and 3 for CFF fonts.

postscriptFontName *string* PostScript name of the font. Corresponds to the OpenType `name` table ID 6. Should be ASCII-only, short (for maximum backwards compatibility less than 30 characters long), and no whitespace allowed, e.g., ‘MyFont-BoldCdlT’. Do not confuse with `postscriptFullName` (see below).

‘The `FontName` generally consists of a family name (specifically, the one used for `FamilyName`), followed by a hyphen and style attributes in the same order as in the `FullName`. For compatibility with the earliest versions of PostScript interpreters and with the file systems in some operating systems, Adobe limits the number of characters in the `FontName` to 29 characters. As with any PostScript language name, a valid `FontName` must not contain spaces, and may only use characters from the standard ASCII character set. If abbreviations are necessary to meet the 29 character limit, the abbreviations should be used for the entire family’ (Adobe Technote #5088).

Adobe recommends these abbreviations for style names:

- ▶ **Bd** for Bold
- ▶ **Bk** for Book
- ▶ **Blk** for Black
- ▶ **Cm** for Compressed
- ▶ **Cn** for Condensed
- ▶ **Ct** for Compact
- ▶ **Dm** for Demi (*prefix*)
- ▶ **Ds** for Display
- ▶ **Ex** for Extended
- ▶ **Hv** for Heavy
- ▶ **Ic** for Inclined
- ▶ **It** for Italic
- ▶ **Ks** for Kursiv (*German for: Italic*)
- ▶ **Lt** for Light
- ▶ **Md** for Medium
- ▶ **Nd** for Nord (*style name introduced for Antique Olive*)

- ▶ **Nr** for Narrow
- ▶ **Obl** for Oblique
- ▶ **Po** for Poster
- ▶ **Rg** for Regular
- ▶ **Sl** for Slanted
- ▶ **Sm** for Semi (*prefix*)
- ▶ **Su** for Super
- ▶ **Th** for Thin
- ▶ **Ult** for Ultra (*prefix*)
- ▶ **Up** for Upright
- ▶ **X** for Extra (*prefix*)

postscriptFullNames Name to be used for the FullName field in CFF table as well as for name table ID 4 (full font name). This is the complete name of the font as it is supposed to appear to the user, and is thus allowed to contain spaces, e.g., ‘My Font Bold Condensed Italic’. Do not confuse with postscriptFontName (see above).

Some systems match the family name ‘against the FullName for sorting into family groups.’ Therefore, the family name ‘must match the corresponding portion of the FullName, and be suitable for display in font menus. All fonts that are stylistic variations of a unified design should share the same FamilyName. [...] The FullName begins with a copy of the FamilyName and is completed by adding style attributes — generally in this sequence: weight, width, slope, optical size’ (Adobe Technote #5088). The full name ‘would typically be a combination of name IDs 16 and 17’ (typographic family and subfamily names), ‘without needing any additional qualifications regarding “Regular”’.
OpenType spec: name ID 4⁷

preferredFamilyNames Typographic (a.k.a. ‘preferred’) family name. Corresponds to name ID 16 in the OpenType name table. Setting this parameter only makes sense if it is different from the Family Name (name ID 1) as set in *File > Font Info > Font*.

‘The typographic family grouping doesn’t impose any constraints on the number of faces within it, in contrast with the 4-style family grouping (ID 1), which is present both for historical reasons and to express style linking groups. If name ID 16 is absent, then

7 <https://learn.microsoft.com/typography/opentype/spec/name#nid4>

name ID 1 is considered to be the typographic family name.’

preferredSubfamilyNames *Typographic* (a.k.a. ‘preferred’) subfamily name. Corresponds to name ID 17 in the OpenType [name](#) table. Setting this parameter only makes sense if it is different from the Style Name (name ID 2) as set in *File > Font Info > Instances*. ‘This allows font designers to specify a subfamily name within the typographic family grouping. This string must be unique within a particular typographic family. If it is absent, then name ID 2 is considered to be the typographic subfamily name.’

PreFilter *string* Same as *Filter*, but applied before decomposition. Only applies to static fonts.

prep Table Assembly *string* Assembly code for the [prep](#) (Pre-Program or Control Value Program) table. When you open a TTF in Glyphs, the application will store the existing prep code in this parameter. This is intended to preserve existing TT hinting for reexporting, and certainly not to be edited manually. Remove this parameter if you want to do your own TrueType instructing.

Prevent Name ID *string* Will prevent the export of the specified name table entry. Add multiple parameters for multiple name IDs. E.g. if you want Glyphs not to export IDs 16 (Typographic Family Name) and 17 (Typographic Subfamily Name), add two of these parameters, once with 16 and once with 17 as value.

Preview Ascender *float* Master parameter for the distance between baseline and the upper edge of the preview in font units. Useful for scaling the preview at the bottom of the Edit View or in the Preview Panel when you have large ascenders that would otherwise be cut off. The default is 1000.

Preview Descender *float* Similar to [Preview Ascender](#), a master parameter for the distance between baseline and the lower edge of the preview in font units. Defaults to [winAscent](#) if present, or otherwise, the Descender value set in *File > Font Info > Masters*.

Propagate Anchors *boolean* Enable or disable the propagation of top and bottom anchors in composites. This means that top and bottom anchors in the base glyphs of components ‘shine through’ to the composite, unless an anchor with the same name is present in the composite glyph. That way, they enable mark-to-base and mark-to-mark attachment for the composite without needing to add and manage additional anchors. Default is on. Primary use for this parameter is for suppressing generation of [mark](#) and [mkmk](#) rules for composites.

Reencode Glyphs *list* Takes a list of `glyphname=unicodevalue` pairs, e.g., `smiley=E100`, `logo=E101`, etc. Assign *multiple* Unicode values with a comma as separator, e.g., `hyphen=002D, 2010`. The parameter assigns the Unicode value to the glyph with the specified name at export. Should the Unicode value in question already be assigned to another glyph, the Unicode value of that other glyph will be deleted, but all production names will remain intact. It will *remove* a glyph's Unicode assignment if the Unicode value is left out, e.g., `f_f_i=` and `f_f_j=` will strip `f_f_i` and `f_f_j` from their Unicode value.

Remove Classes *list* Prevents the export of the OpenType classes mentioned in the list. Useful for removing manually written classes when glyphs are removed from the font through the subsetting parameters. Note that automatic classes are removed automatically at export if the triggering glyphs are not in the font anymore, e.g., because they were removed or renamed with parameters.

Remove Features *list* Prevents the export of the OpenType features mentioned in the list. Useful when a glyph name suffix triggers Glyphs to generate a feature you do not want in the font, or you just want to disable a manually added feature for an instance. Note that automatic features are removed automatically at export if the triggering glyphs are not in the font anymore.

Remove Glyphs *list* Will prevent the glyphs and groups of glyphs mentioned in the list from being exported into the font. Automatically generated OpenType features respect changes in the glyph structure, e.g., if you remove all smallcap glyphs, then it will not auto-generate the `smcp` or `c2sc` features. Useful for subsetting. Per line, you can use:

- ▶ **glyph name:** the full unabridged glyph name as it appears in Font view. You can copy a list of glyph names by invoking the context menu on a glyph selection and choosing *Copy Glyph Names > One per line*.
- ▶ **wildcard:** use an asterisk before, inside or after a string. Examples:
 - ▶ `*ogonek` will find `Aogonek`, `aogonek`, `Eogonek`, `eogonek`, etc.
 - ▶ `K*` will find `K`, `Kcommaaccent` and `K.ss01`.
 - ▶ `H*.ss01` will find `H.ss01`, `Hbar.ss01` and `Hcircumflex.ss01`.
- ▶ **category:** use the `category=value` syntax to match glyphs

dynamically. If `value` is a string, you can use wildcards. Possible categories are:

- ▶ `name` The glyph name, equivalent to the glyph name searches described above.
- ▶ `unicode` The first Unicode value of the glyph, e.g., `unicode=03*` excludes all glyphs that have Unicode values between 0300 and 03FF.
- ▶ `note` The note of the glyph, e.g., `note=*delete*` will remove all glyphs that have the word delete in their glyph note.
- ▶ `script` The writing system that is assigned to the glyph, e.g., `script=thai` will remove all Thai glyphs from the font. The spelling of the script is case-sensitive and needs to be exactly as displayed in *Window > Glyph Info*.
- ▶ `category` The group the glyph belongs to as displayed in *Window > Glyph Info*, e.g., `category=Symbol` will remove all glyphs defined as symbol.
- ▶ `subCategory` The subcategory as displayed in *Window > Glyph Info*, e.g., `subCategory=Lowercase` will remove all lowercase glyphs from the exported OpenType font.
- ▶ `production` The production name the glyph is assigned at export.
- ▶ `leftMetricsKey`, `rightMetricsKey`, `widthMetricsKey`, `topMetricsKey`, `bottomMetricsKey`, `vertWidthMetricsKey` The metrics key for LSB, RSB, width, top, bottom or vertical width, e.g., `widthMetricsKey=*.tf*` removes all glyphs where the width is synced with a tabular figure.
- ▶ `leftKerningGroup`, `rightKerningGroup`, `topKerningGroup`, `bottomKerningGroup` The kerning group of a glyph, e.g., `leftKerningGroup=s` removes all glyphs that have a left kerning group called 's'.
- ▶ `colorIndex` The color index of CPAL/COLR layers. E.g., assume you defined a color palette with blue at color index 2, then `colorIndex=2` would remove all blue shapes from the exported color font.
- ▶ `countOfLayers` Number of layers a glyph has. E.g., `countOfLayers=4` removes all glyphs that have 4 layers (including the master layers).
- ▶ `mastersCompatible` Whether the glyph is interpolatable through all masters or not. E.g., `mastersCompatible=0` removes all

incompatible glyphs from the export.

- ▶ **export** Whether the glyph is set to export or not. This also affects glyphs that are contained as components in others. E.g., `export=0` will remove all non-exporting glyphs from the export, including components if the referenced base glyph is not exporting, as well as corners, caps and smart glyphs.
- ▶ **isAnyColorGlyph** Whether the glyph is a COLR/CPAL, sbix, or full color glyph or not, e.g., `isAnyColorGlyph=0` removes all non-color glyphs from the font.
- ▶ **isAppleColorGlyph** Whether the glyph has an iColor layer for the `sbix` table or not.
- ▶ **hasSpecialLayers** Whether the glyph has a bracket or brace layer or not, e.g., `hasSpecialLayers=1` will remove all glyphs with a bracket or brace layer.

Remove post names for webfonts *boolean* Removes glyph names in the webfont export, resulting in smaller file sizes.

Remove Prefixes *list* Takes a list of names for OT feature prefixes as defined in *File > Font Info > Features*. The code in the prefixes will be kept from being compiled and inserted in the exporting OpenType font.

Rename Glyphs *list* Will exchange the glyphs mentioned in the value with each other. Takes a list of rename strings of the form `oldname=newname`, e.g. `e.bold=e`, `g.alt=g`. The glyph previously stored as `newname` will now be called `oldname` and vice versa. The parameter will update composites that employ the glyphs involved, update automatic features where necessary, and also exchange the Exports attributes of glyphs. If you want to avoid the export of one the glyphs, make sure that either their Exports attributes are set accordingly, or use the **Export Glyphs** parameter.

Replace Class *string* Replaces OpenType class code with custom code. The first word must be the class name (without the at sign), followed by a semicolon, and the new class code. Works only if the class exists in *File > Font Info > Features*. This is only necessary for manually set up classes. Automatically generated classes update automatically.

Replace Feature *string* Replaces the content of an OpenType feature with the code specified. The first four letters must be the feature name (such as `Liga`), followed by a semicolon and the new feature

code. Works only if the feature exists in *File > Font Info > Feature*.

Replace Prefix *string* Replaces OpenType feature code listed under *File > Font Info > Features > Prefix*. The value must consist of the prefix name, exactly as entered in *Font Info > Features*, followed by a semicolon and the replacement code.

ROS *string* Sets the ROS (Registry, Ordering, Supplement) for fonts with a Character To Glyph Index Mapping Table (cmap). Currently available values are the public ROSes:

- ▶ Adobe-CNS1-6
- ▶ Adobe-GB1-5
- ▶ Adobe-Japan1-3
- ▶ Adobe-Japan1-6
- ▶ Adobe-Korea1-2
- ▶ Adobe-Identity-0 If you use *Adobe-Identity-0*, a GSUB table will be generated from the available OpenType features. Otherwise, the cmap and GSUB resources supplied by Adobe are used.

From the Adobe CMap and CIDFont Files Specification, Version1.0: 'Both the CIDFont and the CMap must use CIDs from compatible character collections. The identification of the character collection is accomplished by placing version control information into each CIDFont and CMap file. To identify a character collection uniquely, three name components are concatenated with a hyphen:

- ▶ a registry name is used to identify an issuer of orderings, usually Adobe;
- ▶ an ordering name is used to identify an ordered character collection; and,
- ▶ a supplement number is used to indicate that the ordered character collection for a registry, ordering, and previous supplement has been changed to add new characters assigned CIDs beginning with the next available CID.

These three pieces of information taken together uniquely identify a character collection. In a CIDFont, this information declares what the character collection is. In a CMap, this information specifies which character collection is required for compatibility. A CMap is compatible with a CIDFont if the registry and ordering are the same. If the supplement numbers

are different, some codes may map to the CID index of 0.’

sampleTexts *Sample text.* Corresponds to the OpenType `name` table ID 19. ‘This can be the font name, or any other text that the designer thinks is the best sample to display the font in.’ This sample text is displayed, for instance, by Apple Font Book, when the font is selected in Sample view.

Save as TrueType *boolean* Exports the instance as TTF, regardless of the settings in the Export dialog.

SBIX to SVG *integer* If set, exports the bitmaps built for an `sbix` font in an `SVG` color table. The `SVG` table supports both bitmap and vector images, and with this parameter you can duplicate the `sbix` bitmap information into an equivalent `SVG` bitmap table, making the font more compatible. Adding the `Export sbix` parameter with a deactivated checkbox will export `SVG` only and not include `sbix` in the instance in question.

Scale to UPM *integer* Scales the whole font to the supplied integer value. This is useful for scaling to a UPM of 2048 (or a power of two between 16 and 16,384) for TTF export, or if you are designing in an UPM size other than the default 1000.

shoulderHeight *integer* A vertical metric value for Middle Eastern, South Asian and Southeast Asian scripts. In glyphs of those scripts, the shoulder height will be displayed as a vertical metric line in Edit view instead of the x-height. The algorithm for automatic creation of alignment zones also respects this value.

smallCapHeight *integer* A vertical metric for small caps. The algorithm for automatic creation of alignment zones respects this value. When a small cap glyph is displayed in Edit view and metrics are set to show, the small cap height will be displayed instead of the x-height.

strikeoutPosition *integer* ‘The position of the top of the strikeout stroke relative to the baseline in font design units.’Corresponds to the `yStrikeoutPosition` field in the `OS/2` table. ‘Positive values represent distances above the baseline; negative values represent distances below the baseline. Aligning the strikeout position with the em dash is suggested. Note, however, that the strikeout position should not interfere with the recognition of standard characters, and therefore should not line up with crossbars in the font.’

strikeoutSize *integer* The size of the strike-out dash in units.

Corresponds to the `yStrikeoutSize` field in the `OS/2` table. ‘This field should normally be the thickness of the em dash for the current font, and should also match the underline thickness.’

Style Name as STAT entry *string* For variable fonts, takes the instance style name as combinable display string for an axis range. As value, use the four-letter axis tag to which the display string applies. Use this only in instances that are non-normal on one axis and normal on all others. That is because the normal attributes have elidable names and do not appear in the style name (e.g., ‘Semibold’ or ‘Condensed’).

Example: In the Light instance, use this parameter with the value `wght`, because Light is a value on the weight axis. The Light instance is non-normal on the `wght` axis, but normal (i.e., not condensed nor extended) on the `wdth` axis.

styleMapFamilyNames Family name used for RIBBI style mapping (regular, italic, bold, bold italic). You can use this to create subfamilies within larger font families. ‘Up to four fonts can share the Font Family name, forming a font style linking group.’ Glyphs uses the entries in Style Name field and in the Style Linking section in the Instances tab of the Font Info for linking the four individual weights.

styleMapStyleNames Localised Font Subfamily name. Corresponds to the OpenType `name` table ID 2

styleNames Style Name or Font Subfamily Name. Corresponds to OpenType `name` table ID 2.

‘The Font Subfamily name is used in combination with Font Family name (name ID 1), and distinguishes the fonts in a group with the same Font Family name. This should be used for style and weight variants only.’

subscriptXOffset *integer* The horizontal offset for simulated subscript typesetting, recommended to keep at zero for fonts with an italic angle of zero. Corresponds to the `subscriptXOffset` field in the `OS/2` table.

‘The Subscript X Offset parameter specifies a font designer’s recommended horizontal offset – from the glyph origin to the glyph origin of the subscript’s glyph – for subscript glyphs associated with this font. If a font does not include all of the required subscript glyphs for an application, and the application can substitute glyphs, this parameter specifies the recommended horizontal position from the glyph escapement point of the last

glyph before the first subscript glyph. For upright glyphs, this value is usually zero; however, if the glyphs of a font have an incline (italic or slant), the reference point for subscript glyphs is usually adjusted to compensate for the angle of incline.’

subscriptXSize *integer* The horizontal scale for simulated subscript typesetting. Corresponds to the [subscriptXSize](#) field in the OS/2 table.

‘If a font has two recommended sizes for subscripts, e.g., numerics and other, the numeric sizes should be stressed. This size field maps to the em size of the font being used for a subscript. The horizontal font size specifies a font designer’s recommended horizontal size of subscript glyphs associated with this font. If a font does not include all of the required subscript glyphs for an application, and the application can substitute glyphs by scaling the glyphs of a font or by substituting glyphs from another font, this parameter specifies the recommended nominal width for those subscript glyphs. For example, if the em for a font is 2048 units and ySubScriptXSize is set to 205, then the horizontal size for a simulated subscript glyph would be 1/10th the size of the normal glyph.’

subscriptYOffset *integer* The vertical offset for simulated subscript typesetting, typically a positive number for going below the baseline. Corresponds to the [subscriptYOffset](#) field in the OS/2 table.

‘The Subscript Y Offset parameter specifies a font designer’s recommended vertical offset from the glyph baseline to the glyph baseline for subscript glyphs associated with this font. Values are expressed as a positive offset below the glyph baseline. If a font does not include all of the required subscript glyphs for an application, this parameter specifies the recommended vertical distance below the glyph baseline for those subscript glyphs.’

subscriptYSize *integer* The vertical scale for simulated subscript typesetting. Corresponds to the [subscriptYSize](#) field in the OS/2 table. See [subscriptXSize](#) for more details.

superscriptXOffset *integer* The horizontal offset for simulated superscript typesetting, recommended to keep at zero for fonts with an italic angle of zero. Corresponds to the [superscriptXOffset](#) field in the OS/2 table. See [subscriptXOffset](#) for more details.

superscriptXSize *integer* The horizontal scale for simulated

superscript typesetting. Corresponds to the `superscriptXSize` field in the `OS/2` table. See `subscriptXSize` for more details.

`superscriptYOffset` *integer* The vertical offset for simulated superscript typesetting, typically a positive value for going below the baseline. Corresponds to the `superscriptYOffset` field in the `OS/2` table. See `subscriptYOffset` for more details.

`superscriptYSize` *integer* The vertical scale for simulated superscript typesetting. Corresponds to the `superscriptYSize` field in the `OS/2` table. See `subscriptYSize` for more details.

`trademarks` Trademark statement. Corresponds to the OpenType `name` table ID 7. According to Microsoft, ‘this is used to save any trademark notice / information for this font. Such information should be based on legal advice. This is distinctly separate from the copyright.’

`TrueType Curve Error` *float* Maximum deviance of the approximated TrueType curve in units. Default is 0.6. A higher curve error allows the TrueType converter to use fewer quadratic splines to approximate the cubic splines of your design. This can result in a significantly smaller `glyf` table (containing the quadratic outline data), and smaller overall file size.

`TrueType Keep GlyphOrder` *boolean* Keeps the glyph order as it is in the `.glyphs` file, except `.notdef` and `space` which always have to be in the first two positions. If disabled (the default), the font will resort the first four glyphs to: `.notdef`, `NULL`, `CR`, `space`. While `NULL` and `CR` will only be reordered if they exist in the `.glyphs` file, `.notdef` and `space` will be automatically generated if they are missing. **Use this parameter only if you know what you are doing.**

`TTFAutohint binary path` *string* File path to a precompiled TTFAutohint binary that should be used instead of the built-in TTFAutohint. This can be useful if you need to stick to a specific version or want to employ a newer version of TTFAutohint than Glyphs incorporates.

`TTFAutohint control instructions` *string* This allows you to specify TTFAutohint control instructions. It is recommended to prepare the control code in a separate file and then paste it into the value of the parameter. Possible instructions are:

- ▶ `glyphnames left pointIDs offset`
- ▶ `glyphnames right pointIDs offset`

- ▶ `glyphnames nodir pointIDs`
- ▶ `glyphnames touch pointIDs xshift x yshift y @ PPMs`
- ▶ `glyphnames point pointIDs xshift x yshift y @ PPMs`

Values for `offset` are optional and assumed as zero when omitted. In the `touch` and `point` instructions, either or both of the shifts can be specified. `x` and `y` must be between 0.0 and 1.0. `glyphnames` can be one or more comma-separated glyph names, specified as production names (i.e., the names as they are written into the font file). `PPMs` can be a single PPM size, a size range of PPMs with a hyphen, or a comma-separated list of sizes and size ranges. A line that starts with a hashtag `#` is considered a comment and therefore ignored. The instructions can be abbreviated with their respective first letters, e.g., ‘right’ can be written as ‘r’.

TTFAutohint options *string* Specifies commandline options for the TrueType autohinter ‘`ttfautohint`’. Use the dialog sheet to configure your settings:

- ▶ *Hint Set Range*: the PPM range for which the instructions will be optimized. Large ranges can cause huge file sizes.
- ▶ *Default Script*: ‘default script for OpenType features’.
- ▶ *Fallback Script*: ‘default script for glyphs that can’t be mapped to a script automatically’.
- ▶ *Hinting Limit*: the PPM size ‘where hinting gets switched off’. Default is 200 pixels, must be larger than the maximum of the hint set range. Pixel sizes up to this size use the hinting configuration for the range maximum.
- ▶ *Fallback Stem Width*: ‘the horizontal stem width (hinting) value for all scripts that lack proper standard characters in the font. The value is given in font units and must be a positive integer. If not set, `ttfautohint` uses a hard-coded default (50 units at 2048 units per em, and linearly scaled for other UPM values, for example 24 units at 1000 UPM).’ For symbol fonts, you also need to specify a Fallback Script ‘to set up a script at all’.
- ▶ *x-Height Increase Limit*: from this pixel size down to 6 PPM, the x-height is more likely to be rounded up. Default is 14 PPM. ‘Normally, `ttfautohint` rounds the x height to the pixel grid, with a slight preference for rounding up. (...) Use this flag to increase the legibility of small sizes if necessary.’ Set to 0 if you want to switch off rounding up the x-height.

- ▶ *x-Height Snapping Exceptions*: ‘list of comma-separated PPM values or value ranges at which no x-height snapping shall be applied’, e.g., ‘8, 10-13, 16’ disables x-height snapping for sizes 8, 10, 11, 12, 13, and 16. An empty string means no exceptions, and a mere dash (‘-’) disables snapping for all sizes.
- ▶ *Adjust Subglyphs (formerly known as Pre-Hinting)*: If enabled, ‘makes a font’s original bytecode be applied to all glyphs before it is replaced with bytecode created by ttfautohint. This makes only sense if your font already has some hints in it that modify the shape even at EM size (normally 2048px); in particular, some CJK fonts need this because the bytecode is used to scale and shift subglyphs (hence the option’s long name). For most fonts, however, this is not the case.’
- ▶ *Dehint*: Disables all TT hinting, and therefore overrides all other options. Use only for testing.
- ▶ *Detailed Info*: if enabled, adds ‘ttfautohint version and command line information to the version string or strings (with name ID 5) in the font’s `name` table’. This option is mutually exclusive with the *No Autohint Info* option (see below). ‘If neither is set, the string “ttfautohint (vNNN)” gets added to the `name` table’, NNN being the ttfAutohint version.
- ▶ *Hint Composites*: ‘By default, the components of a composite glyph get hinted separately. If this flag is set, the composite glyph itself gets hinted (and the hints of the components are ignored). Using this flag increases the bytecode size a lot, however, it might yield better hinting results – usually, it doesn’t.’ Also adds a ghost component called `.ttfautohint` to all glyphs. ‘Direct rendering of the `.ttfautohint` subglyph (this is, rendering as a stand-alone glyph) disables proper hinting of all glyphs in the font! Under normal circumstances this never happens because `.ttfautohint` doesn’t have an entry in the font’s `cmap` table.’ But it can happen, e.g., in a glyph overview.
- ▶ *Ignore Restrictions*: ‘By default, fonts that have bit 1 set in the `fsType` field of the `OS/2` table are rejected. If you have a permission of the font’s legal owner to modify the font, specify this command line option.’
- ▶ *No Autohint Info*: if checked, prevents adding ‘ttfautohint version and command line information to the version string or strings (with name ID 5) in the font’s `name` table.’
- ▶ *Symbol Font*: ‘Process a font that ttfautohint would refuse

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

- ▶ *ttfa table*: Adds an OpenType table 'called `TTFA` to the output font that holds a dump of all parameters. In particular, it lists all `ttfautohint` control instructions (which are not shown in the name table info). This option is mainly for archival purposes so that all information used to create a font is stored in the font itself. Note that such a `TTFA` table gets ignored by all TrueType rendering engines. Forthcoming versions of the `ttfautohint` front-ends will be able to use this data so that a font can be processed another time with exactly the same parameters, thus providing a means for round-tripping fonts.'
- ▶ *Windows Compatibility*: 'This option makes `ttfAutohint` add two artificial blue zones, positioned at the `winAscent` and `winDescent` values (from the font's `OS/2` table). The idea is to help `ttfAutohint` so that the hinted glyphs stay within this horizontal stripe since Windows clips everything falling outside.' Use this option if clipping occurs in Microsoft Windows and you cannot adjust `winAscent` and `winDescent` instead (which would usually be the better option). In combination with '-' as value for `xHeight Snapping Exceptions` (see above), it should both 'suppress any vertical enlargement' and 'prevent almost all clipping.'
- ▶ *Strong Stems*: specifies which algorithm to use 'for computing horizontal stem widths and the positioning of blue zones' for the three rendering targets: Grayscale (Android), GDI ClearType (old Windows versions including XP), DW ClearType (IE 9 and later, and Windows 7 and later). If disabled, stems will be quantized: 'Both stem widths and blue zone positions are slightly quantized to take discrete values. For example, stem values 50, 51, 72, 76, and 100 would become 50, 74, and 100 (or something similar). More glyph shape distortion but increased contrast.' If enabled, stems will be strong: 'Stem widths and blue zones are snapped and positioned to integer pixel values as much as possible. This gives high contrast, but glyph shape distortion can be significant.'

TTFBlueFuzz *integer* Much like PostScript's BlueFuzz, extends the range of TrueType alignment zones by the given amount in both directions. Default and fallback value is 1. Only affects zones defined in the TTFZones parameter.

TTFDontPreserveDiagonals *boolean* In manual TT hinting, the apparent angles of slanted stems are preserved, even when another stem crosses it and threatens to make it appear broken. E.g. in an uppercase A, the two diagonal stems are preserved in their angles, even though a (hinted) crossbar interrupts the outline in their middles. With this parameter, stem angles are *not* preserved. Technically, the parameter suppresses (projections onto) freedom vectors. Useful for making TT hinting smaller, e.g., for webfont export.

TTFFamilyZonesThreshold *integer* The PPM size until which Family Alignment Zones are applied. For adding family alignment zones, add a TTFZones parameter to *Font Info > Font* rather than *Font Info > Master* and make sure the number of defined zones is the same in both places.

TTFMinimumDistance *float* Any hinted stem will be drawn with this minimum length in pixels, no matter which PPM size, if it has a stem hint applied to it. The default is 0.25. This value can be important in small pixel sizes, where small parts are in danger of disappearing.

TTFovershootSuppressionBelowPPM *integer* The pixel size (PPM) up to which overshoots are reliably flattened out. Only applies to manual TT hinting, not ttfAutohint.

TTFStems *list* A list of stem definitions for TrueType manual hinting only. When you click in the parameter value, a dialog sheet will drop down. Use the gear menu to add or remove stem definitions, or import the currently available horizontal PostScript stems from the Horizontal Stems and Vertical Stems fields in *File > Font Info > Masters*. For each stem, you can define an orientation, a name and a width. In the main master, a delta and a globe symbol will be shown in addition: they provide access to dialogs for defining PPM deltas as well as a glyph filter for the stem in question. Add stems by pressing on the plus button, and remove a stem by selecting it and clicking the minus button.

- ▶ *Orientation*: Switch between horizontal stem (e.g., for the crossbars in e, f, t, or the top and bottom curves of o, c, e, a) and vertical stem (e.g., for the vertical stems of h, m, n, u, or the left

and right curves of o) by clicking on the double arrow symbol.

- ▶ *Name*: The stem name is arbitrary, but should be unique. Will show up in the pop-up menu in the grey info box when the TT Instructor tool (I) is active and a stem hint (S) is selected.
- ▶ *Width*: The average size of the stem towards which the stems will be rounded. Also, when applying the Autohint command from the context menu of the TT Instructor tool (I), stems will be identified with this size.
- ▶ *Deltas*: PPM-specific size adjustments for the effective pixel-size of a stem in an instance. In any PPM/instance field, click repeatedly to switch between no change (blank field), size increase (arrow up), size decrease (arrow down). The deltas are only accessible in the first master, or the master defined in the Get Hints From Master parameter.
- ▶ *Filter (Scope)*: Define the glyph scope of the stem by adding logical filters. Click on the plus button to add additional filters, and the minus button to remove a selected filter. Opt-click on the plus button to add logical AND and OR operators for the following (indented) conditions. Available filters are Name, Category, Subcategory and Script. TrueType stems with a scope will only be available in glyphs that fulfill the logical conditions of its scope. Scopes are only accessible in the first master, or the master defined in the Get Hints From Master parameter.

TTFZoneRoundingThreshold *float* Controls the likelihood of a positive zone being pushed up a pixel. It takes a small decimal number as value, typically something around 0.1 or 0.2. The value is added to any positive zone position before rounding, and added twice to the x-height zone (the one named 'xHeight' in the TTFZones parameter). Default is 0.09375.

Example: At a certain font size, the smallcap zone ends up at 6.45 pixels, and the x-height at 5.25 pixels. Without any change, the smallcap zone would round and snap to a height of 6 pixels, while the x-height would end up with 5 pixels. If you set a value of 0.2, the smallcap height ends up at $(6.45+0.2=6.65\approx) 7$ pixels, and the x-height at $(5.25+2\times 0.2=5.65\approx) 6$ pixels.

TTFZones *list* A list of zone definitions for horizontal TrueType stems, in manual TrueType hinting only. When you click in the parameter value, a dialog sheet will drop down. Use the gear menu to add or remove zone definitions, or import the currently available PostScript zones from Alignment Zones field in *File > Font Info >*

Masters. For each zone, you can define a name, a position, a size and an alignment. In the main master, a delta and a globe symbol will be shown in addition: they provide access to dialogs for defining PPM deltas as well as a glyph filter for the zone in question. Add zones by pressing on the plus button, and remove a zone by selecting it and clicking the minus button.

- ▶ *Name:* The zone name is arbitrary, but should be unique. Will show up in the pop-up menu in the grey info box when the TT Instructor tool (I) is active and an align hint (A) is selected.
- ▶ *Position:* Position of the zone, or ‘flat end’ of overshooting shapes, similar to alignment zones in PostScript.
- ▶ *Size:* Size of the zone, or distance from ‘flat end’ to the outermost edge of overshooting shapes. Use positive values for top zones, negative values for bottom zones. If the zone size is calculated to be less than half a pixel in any given PPM size, any hinted shape that reaches into the zone will be flattened to the same height.
- ▶ *Align:* Link a zone to another zone with the Align option. If a zone is aligned to another, the distance between the zone positions is rounded and applied to the zone. This will result in more consistent transitions when you step your font through pixel sizes. Use this for heights that are very close to each other, perhaps even overlapping, and may appear next to each other in typesetting, and where it may be perceived as problematic if the heights diverge too far in low-res pixel renderings, e.g., align the small-cap height to the x-height. Aligned zones will be displayed at the same height if the difference is less than half a pixel in a given PPM size; and at least one pixel apart if the difference is half a pixel or more.
- ▶ *Delta:* PPM-specific position rounding for the effective pixel-size of each zone in each instance. In any PPM/instance field, click repeatedly to switch between no change (blank field), shifting up (arrow up), shifting down (arrow down). The deltas are only accessible in the first master, or the master defined in the Get Hints From Master parameter.
- ▶ *Filter (Scope):* Define the glyph scope of the stem by adding logical filters. Click on the plus button to add additional filters, and the minus button to remove a selected filter. Opt-click on the plus button to add logical AND and OR operators for the following (indented) conditions. Available filters are Name, Category, Subcategory and Script. TrueType stems with a scope

will only be available in glyphs that fulfill the logical conditions of its scope. Scopes are only accessible in the first master, or the master defined in the Get Hints From Master parameter.

typoAscender *integer* The height of the ascenders in units. Corresponds to the OpenType *OS/2* table *sTypoAscender* field. ‘The typographic ascender for this font. This field should be combined with the *typoDescender* and *typoLineGap* values to determine default line spacing. This field is similar to the *hheaAscender* field as well as to the *winAscent* field. However, legacy platform implementations used those fields with platform-specific behaviors. As a result, those fields are constrained by backward-compatibility requirements, and they do not ensure consistent layout across implementations. The *typoAscender*, *typoDescender* and *typoLineGap* fields are intended to allow applications to lay out documents in a typographically-correct and portable fashion. The *Use Typo Metrics* flag (*fsSelection* bit 7) is used to choose between using *sTypo** values or *usWin** values for default line metrics. It is *not* a general requirement that *typoAscender* - *typoDescender* be equal to *unitsPerEm*. These values should be set to provide default line spacing appropriate for the primary languages the font is designed to support.

For CJK (Chinese, Japanese, and Korean) fonts that are intended to be used for vertical (as well as horizontal) layout, the required value for *typoAscender* is that which describes the top of the ideographic em-box. For example, if the ideographic em-box of the font extends from coordinates 0,-120 to 1000,880 (that is, a 1000 × 1000 box set 120 design units below the Latin baseline), then the value of *typoAscender* must be set to 880. Failing to adhere to these requirements will result in incorrect vertical layout.

For a detailed discussion of vertical metrics, see the Vertical Metrics⁸ tutorial.

typoDescender *integer* A *negative* integer describing the depth of the descenders in units. Corresponds to the *sTypoDescender* field of the OpenType *OS/2* table.

‘The typographic descender for this font. This field should be combined with the *typoAscender* and *typoLineGap* values to determine default line spacing.’ See *typoAscender* for more details.

⁸ <https://glyphsapp.com/tutorials/vertical-metrics>

- typoLineGap** *integer* The recommended whitespace between lines, measured in units. Corresponds to the OpenType OS/2 table `sTypoLineGap` field.
‘The typographic line gap for this font. This field should be combined with the `typoAscender` and `typoDescender` values to determine default line spacing.’ See `typoAscender` for more details.
- underlinePosition** *integer* The suggested distance from the baseline to the top of the underline. Negative values indicate a position below the baseline. Corresponds to the `post` table entry `underlinePosition`. Default is -100.
- underlineThickness** *integer* Underline thickness value. Corresponds to the `post` table entry `underlineThickness`. Default is 50. ‘In general, the underline thickness should match the thickness of the underscore character (U+005F), and should also match the strikethrough thickness, which is specified in the OS/2 table.’
- unicodeRanges** *list* A list of supported Unicode ranges in the font. Corresponds to the OpenType OS/2 table `ulUnicodeRange1`, `ulUnicodeRange2`, `ulUnicodeRange3` and `ulUnicodeRange4` fields. The dialog offers a search field, so you can quickly spot the proper ranges for your fonts. E.g., if you want to cover all Latin ranges, simply search for ‘latin’ and all corresponding ranges in the list will be displayed.
‘If a bit is set, then the Unicode ranges assigned to that bit are considered functional. If the bit is clear, then the range is not considered functional. Each of the bits is treated as an independent flag and the bits can be set in any combination. The determination of “functional” is left up to the font designer, although character set selection should attempt to be functional by ranges if at all possible.’
- uniqueID** *string* Unique font identifier. Corresponds to the OpenType `name` table ID 3
- unitsPerEm** *integer* Units per em. Default is 1000 for PostScript-flavored OpenType fonts and a power of two between 16 and 16,384 (usually 2048) for TrueType-flavored OpenType fonts. The value specified is the amount of units that will be used for the font size. A smaller value will cause the font to appear larger on screen, and vice versa. This parameter will only set the UPM value, and not scale node coordinates and other measurements. If you do want to scale, see Scale to UPM.

Update Features *boolean* Forces an update of all automatic feature code. This is especially useful in a phase of font production where the glyph set changes a lot, or, if explicitly turned off, for suppressing the automatic feature code generation.

Use Arabic Presentation Form Unicodes *boolean* Use legacy Presentation Form Unicode values for Arabic glyphs

Use Extension Kerning *boolean* If checked, additional kern lookups will be created with a [GPOS](#) Extension lookup type (a.k.a. lookup type 9), allowing the font to store more kerning values. Use this when the attempt to export your font results in an offset overflow error in the [GPOS](#) table, and you cannot or do not want to delete kern pairs, especially exceptions.

‘This lookup provides a mechanism whereby any other lookup type’s subtables are stored at a 32-bit offset location in the [GPOS](#) table. This is needed if the total size of the subtables exceeds the 16-bit limits of the various other offsets in the [GPOS](#) table. In this specification, the subtable stored at the 32-bit offset location is termed the “extension” subtable.’

Use Line Breaks *boolean* If checked, line breaks inside OpenType features will not be escaped (i.e., not replaced with `\012`) when stored in a `.glyphs` file. If unchecked, can facilitate version control, and thus makes sense, e.g., in a git-based workflow.

Use Typo Metrics *boolean* If checked, applications that respect this setting (in particular, versions of Microsoft Office since 2006) will prefer [typoAscender](#), [typoDescender](#), and [typoLineGap](#) over [winAscent](#) and [winDescent](#) for determining the vertical positioning. Default is off. Corresponds to bit 7 (‘don’t use Win line metrics’) in the [OS/2](#) table [fsSelection](#) field. According to Adobe’s [MakeOTF User Guide](#), this bit was introduced ‘so that reflow of documents will happen less often than if Microsoft just changed the behavior for all fonts.’

Microsoft: ‘If set, it is strongly recommended that applications use `typoAscender – typoDescender + typoLineGap` as the default line spacing for this font.’

‘In variable fonts, default line metrics should always be set using the [typoAscender](#), [typoDescender](#) and [typoLineGap](#) values, and the Use Typo Metrics flag should be set. The ascender, descender and lineGap fields in the [hhea](#) table should be set to the same values as [typoAscender](#), [typoDescender](#) and [typoLineGap](#). The [winAscent](#) and [winDescent](#) fields should be used to specify a

recommended clipping rectangle.’

Variable Font Family Name *string* Family name for the variable font export. It makes sense to have a different family name for the likely use case that both static and variable fonts are in use at the same time.

Variable Font File Name *string* File name for the variable font export. Overrides the default ‘VF.ttf’ ending that Glyphs employs.

Variable Font Optimize Deltas *boolean* Will drop OpenType Variation deltas from a contour if none of its nodes moves more than half a unit. Default is on. Set this parameter to off in order to also keep low-significant deltas.

Variable Font Origin *string* Master to be used for the set of outlines that will be stored in the variable font file. All other masters and instances will be reached by adding point deltas to these default outlines.

Variable Font Style Name *string* Family name for the variable font export. It makes sense to have a different family name for the likely use case that both static and variable fonts are in use at the same time.

Variable SubfamilyName *string* Style name for the variable font’s origin master, the (default) outlines stored in the exported variable font.

variablePostscriptFontName PostScript name of the variable font instance. Corresponds to the OpenType *name* table ID 6 and the *postscriptName* for *NamedInstance* entries in the OpenType *fvar* table. Should be ASCII-only, and no whitespace allowed, e.g., ‘MyFontVar-BoldCondensedItalic’. Do not confuse with *postscriptFullName*.

‘The *FontName* generally consists of a family name (specifically, the one used for *FamilyName*), followed by a hyphen and style attributes in the same order as in the *FullName*.’

Must be distinct from the static PostScript Font Name, or may cause a font conflict if both the static and variable font are installed by a user. It is recommended that it be *Variations PostScript Name Prefix* (ID 25), followed by a hyphen, followed by the style name (ASCII, no spaces). OpenType spec: *name* ID 6⁹

variableStyleName *string* Family name for the variable font export. It makes sense to have a different family name for the likely use case

9 <https://learn.microsoft.com/typography/opentype/spec/name#nid6>

that both static and variable fonts are in use at the same time.

variableStyleNames *string* Style name used for variable fonts.

This can be useful when (static) instance names are reused with different static family names (e.g., ‘Bold’ in ‘Myfamily Condensed’ and ‘Bold’ in ‘Myfamily’), which would otherwise cause duplicate entries in the `fvar` table.

Variation Font Origin *string* Master to be used for the set of outlines that will be stored in the variable font file. All other masters and instances will be reached by adding point deltas to these default outlines.

variationsPostScriptNamePrefix *string* PostScript Name Prefix for Variable Fonts. Corresponds to name ID 25 in the OpenType `name` table. For processing of a variable font instance, e.g., in a PDF, a specific PostScript name is calculated automatically, based on the PostScript Font Name, axis tags, and axis positions (design space coordinates). This constructed PostScript name cannot be longer than 127 characters. If no prefix is provided, the US English string for typographic (‘preferred’) family name (name ID 16) will be used, minus any characters not within ASCII A-Z, a-z and 0-9. Microsoft: used as ‘family prefix in the PostScript Name Generation for Variation Fonts algorithm. The character set is restricted to ASCII-range uppercase Latin letters, lowercase Latin letters, and digits. All name strings for name ID 25 within a font, when converted to ASCII, must be identical.’

Adobe: ‘Including a Variable PostScript Name Prefix string (name ID 25) in a font could be useful in the following cases:

- ▶ if the US English typographic family name, US English named instance `fvar` `subfamilyNameID`, or the number of axis descriptors in the font could tip the length of the generated PostScript names to over 127 characters, or
- ▶ if the US English typographic family name contains accented or other characters that when removed by the algorithm ... could cause confusion or even ambiguity in PostScript names. For example, both typographic family names ‘André Sans’ and ‘Andró Sans’ resolve to family prefix ‘AndrSans’, an ambiguity that could be avoided by providing Variation PostScript Name Prefixes ‘AndreSans’ and ‘AndroSans’ in the fonts.’

vendorID *string* Four-character identifier for the creator of the font, consisting of printable ASCII characters (U+0020 through U+007E) only. Corresponds to the `achVendID` field in the

OpenType *OS/2* table. If not set, Glyphs will use ‘UKWN’ (‘unknown’) as Vendor ID. ‘This is not the royalty owner of the original artwork. This is the company responsible for the marketing and distribution of the typeface that is being classified. For example, there may be multiple vendors of ITC Zapf Dingbats, with some vendors providing differentiating benefits in their fonts (more kern pairs, unregularized data, hand hinted, etc.). This identifier will allow for the correct vendor’s type to be used over another, possibly inferior, font file.

Microsoft maintains a registry of vendor IDs. Registered IDs must be unique to a single vendor. Non-registered IDs can also be used, but are discouraged: vendors are strongly encouraged to register an ID to ensure that there are no conflicts between different vendors in use of a given ID, and that customers are able to find vendor contact information for a given font. This field can also be left blank (set to null, or a tag comprised of four space characters):’

versionString *string* Version string. Should begin with the syntax ‘Version.’ (with a space between ‘Version’ and the number). A placeholder string into which the version number will be inserted automatically, e.g., *Version %d.%03d*, where %d stands for an integer, and %03d for integer represented with three digits, e.g., *008*. The string must contain a version number of the following form: one or more digits (0-9) of value less than 65,535, followed by a period, followed by one or more digits of value less than 65,535. Any character other than a digit will terminate the minor number. A character such as ‘;’ is helpful to separate different pieces of version information. The first such match in the string can be used by installation software to compare font versions. Note that some installers may require the string to start with ‘Version.’, followed by a version number as above.

vheaVertAscender *integer* Ascender value for vertical typesetting. Corresponds to the *vertTypoAscender* field in the OpenType *vhea* table. ‘The vertical typographic ascender for this font. It is the distance in units from the ideographic em-box center baseline for the vertical axis to the right edge of the ideographic em-box. It is usually set to $UPM \div 2$. For example, a font with an em of 1000 units will set this field to 500.’

vheaVertDescender *integer* Descender value for vertical typesetting, typically a negative number. Corresponds to the

`vertTypoDescender` field in the OpenType `vhea` table.

‘The vertical typographic descender for this font. It is the distance in units from the ideographic em-box center baseline for the vertical axis to the left edge of the ideographic em-box. It is usually set to $-UPM \div 2$. For example, a font with an em of 1000 units will set this field to -500 .’

vheaVertLineGap *integer* Line gap value for vertical typesetting. Corresponds to the `vertTypoLineGap` field in the OpenType `vhea` table.

‘The vertical typographic gap for this font. An application can determine the recommended line spacing for single spaced vertical text for an OpenType font by the following expression: ideographic embox width + `vertTypoLineGap`.’

Virtual Master Defines a font master for a variable font, thereby extending its design space accordingly. Rather than a ‘real’ master, which is defined in *File > Font Info > Masters*, a virtual master can only be drawn in the form of a Brace layer, i.e., a glyph layer with a name consisting of or ending in the comma-separated design space coordinates between curly braces, e.g., ‘{100, 300}’. A virtual master makes most sense for design axes that only apply to a limited number of glyphs, e.g., an axis that controls the middle crossbar heights of letters like A, E, F and H. The main advantages are that only affected glyphs need to be managed, and kerning does not need to be redone for an extra ‘real’ master.

Webfont Formats *list* For the instance in which this parameter is specified, the listed webfont formats will be exported, regardless of the settings in the Export dialog. Possible values: TTF, EOT, WOFF or WOFF2.

Webfont Only *boolean* If activated, it removes some of the information stored in the font file necessary for desktop use. This makes it harder to convert the webfont into a different format or to install it locally in an operating system like Windows or macOS. Careful: Technically, this option produces a damaged font, which, however, still works as webfont in browsers.

winAscent *integer* A positive integer describing the top extremum of the font rendering box for Windows, beyond which glyph renderings may be clipped. Thus, `winAscent` should be high enough to include all parts of all important glyphs. Corresponds to the `usWinAscent` field in the OpenType `OS/2` table. ‘The “Windows ascender” metric. This should be used to specify

the height above the baseline for a clipping region. This is similar to the `typoAscender` field, and also to the `hheaAscender` field. There are important differences between these, however. In the Windows GDI implementation, the `winAscent` and `winDescent` values have been used to determine the size of the bitmap surface in the TrueType rasterizer. Windows GDI will clip any portion of a TrueType glyph outline that appears above the `winAscent` value. If any clipping is unacceptable, then the value should be set greater than or equal to `yMax`. *Note:* This pertains to the default position of glyphs, not their final position in layout after data from the `GPOS` or `kern` table has been applied. Some legacy applications use the `winAscent` and `winDescent` values to determine default line spacing. This is strongly discouraged. The `typoAscender`, `typoDescender` and `typoLineGap` fields should be used for this purpose. Note that some applications use either the `winAscent` / `winDescent` values or the `typoAscender`/`typoDescender`/`typoLineGap` values to determine default line spacing, depending on whether the `Use Typo Metrics` flag is set. This may be useful to provide compatibility with legacy documents using older fonts, while also providing better and more-portable layout using newer fonts. Applications that use the `typoAscender`/`typoDescender`/`typoLineGap` fields for default line spacing can use the `winAscent`/`winDescent` values to determine the size of a clipping region. Some applications use a clipping region for editing scenarios to determine what portion of the display surface to re-draw when text is edited, or how large a selection rectangle to draw when text is selected. Early versions of this specification suggested that the `winAscent` value be computed as the `yMax` for all characters in the Windows ANSI character set. For new fonts, the value should be determined based on the primary languages the font is designed to support, and should take into consideration additional height that may be required to accommodate tall glyphs or mark positioning.⁷ For a detailed discussion of vertical metrics, see the Vertical Metrics¹⁰ tutorial.

`winDescent` *integer* A positive integer describing the bottom extremum of the font rendering box for Windows. Thus, `winDescent` should be large enough to encompass the descenders of lowercase letters like `g`, `p`, `q`, and `y`. Corresponds to

¹⁰ <https://glyphsapp.com/tutorials/vertical-metrics>

the `usWinDescent` field of the OpenType `OS/2` table.

‘Early versions of this specification suggested that the `winDescent` value be computed as `-yMin` for all characters in the Windows ANSI character set. For new fonts, the value should be determined based on the primary languages the font is designed to support, and should take into consideration additional vertical extent that may be required to accommodate glyphs with low descenders or mark positioning.’

Write DisplayStrings *boolean* If disabled, prevents the *DisplayStrings* from being written into the `.glyphs` file. *DisplayStrings* store the text content of Edit tabs. This can facilitate version control. Default is enabled.

Write lastChange *boolean* If disabled, prevents the *Last Changed Date* from being written into the `.glyphs` file. This can facilitate version control. Default is enabled.

WWSFamilyName *string* WWS family name. WWS stands for ‘Weight Width Slope’. Corresponds to the OpenType `name` table ID 21. ‘Used to provide a WWS-conformant family name in case the entries for IDs 16 (`preferredFamilyName`) and 17 (`preferredSubfamilyName`) do not conform to the WWS model. (That is, in case the entry for ID 17 includes qualifiers for some attribute other than weight, width or slope.)’ Frequent use cases are family names that indicate optical sizes: ‘Examples of name ID 21: “Minion Pro Caption” and “Minion Pro Display”. (Name ID 16 would be “Minion Pro” for these examples.)’

WWSSubfamilyName *string* WWS Subfamily name. Corresponds to the OpenType `name` table ID 22. ‘Used in conjunction with ID 21, this ID provides a WWS-conformant subfamily name (reflecting only weight, width and slope attributes) in case the entries for IDs 16 and 17 do not conform to the WWS model. [...] Examples of name ID 22: “Semibold Italic”, “Bold Condensed”. (Name ID 17 could be “Semibold Italic Caption”, or “Bold Condensed Display”, for example.)’ For name IDs 16 and 17, see the entries for `preferredFamilyName` and `preferredSubfamilyName`, respectively.

The text face is ABC Arizona, designed by Elias Hanzer for Dinamo. This is a variable font ranging from thin to bold, sans serif to serif, and upright to italic.

Computer code is set in Cascadia Code, designed by Aaron Bell for Microsoft. The version used in this handbook has been slightly adjusted to accompany the text face.

The illustrations placed throughout the pages use glyphs of the following typefaces: ABC Arizona; Alegreya, Alegreya Sans, & Piazzolla by Juan Pablo del Peral/Huerta Tipográfica; Apple Color Emoji by Apple; Cerne by Peter S. Baker; Cormorant by Christian Thalmann/Catharsis Fonts; Graublau Sans & Graublau Slab by Georg Seifert; Hola Pixel by Rainer Erich Scheichelbauer/Schriftlabor; Lapture by Tim Ahrens/Just Another Foundry; Liebe Heide by Ulrike Rausch/LiebeFonts; Literata by Veronika Burian & José Scaglione/TypeTogether; Lyon Arabic by Khajag Apelian, Wael Morcos, & Kai Bernau/Commercial Type; Mada by Khaled Hosny/Alif Type; Proxima Vara by Mark Simonson; Rasa by Anna Giedrys & David Brezina/Rosetta Type; Sapperlot by Thomas Maier; and Work Sans by Wei Huang.

