```
md`# Mapbox Glyph PBFs`
```

```
md`The [Mapbox Style Specification](https://docs.mapbox.com/mapbox-gl-js/style-spec/) for vector maps assumes that text will be rendered using signed distance fields (SDFs). This method for variable-resolution text rendering was introduced to the computer gaming world by [Valve](https://www.valvesoftware.com/) in a [2007 paper](https://steamcdn-a.akamaihd.net/apps/valve/2007/SIGGRAPH2007_AlphaTestedMagnification.pdf). The application to map labeling was explained by Konstantin Käfer in a [2014 blog post](https://blog.mapbox.com/drawing-text-with-signed-distance-fields-in-mapbox-gl-b0933af6f817).
`
```

```
md`## Retrieving the data
In the style document, the location of the SDF data is specified by a [glyphs](https://docs.mapbox.com/mapbox-gl-js/style-spec/glyphs/) property, which takes values that look something like this:
`
```

```
urlTemplate = "https://api.maptiler.com/fonts/{fontstack}/{range}.pbf?key={key}"
```

```
md`The three tokens are filled in at runtime:
- \`{fontstack}\` is replaced by the string specified in a symbol layer's [text-font](https://docs.mapbox.com/mapbox-gl-js/style-spec/layers/#layout-symbol-text-font) property
- \`{range}\` is replaced by a range of 256 code points in the [UTF-8 character encoding](https://en.wikipedia.org/wiki/UTF-8). The endpoints of the range are supplied as base-10 integers, rather than the hexadecimal representation used by UTF-8 itself
- \`{key}\` is replaced by the user's API access key
`
```

```
function getHref(family, range) {
  return urlTemplate
    .replace('{fontstack}', family.split(" ").join("%20"))
    .replace('{range}', range)
    .replace('{key}', "mrAq6zQEFxOkanukNbGm"); // Get your own key: maptiler.com
}
```

```
md`For example, here is the URL for the [Noto Sans Regular](https://fonts.google.com/specimen/Noto+Sans) font
and the commonly used characters in the [Unicode Basic Latin and Basic Latin-1 Supplement blocks](https://
en.wikipedia.org/wiki/Unicode_block):`
```

```
testHref = getHref("Noto Sans Regular", "0-255")
```

```
md`## Parsing the PBF`
```

```
md`Let's see what the actual data looks like. We parse the data retrieved from the link with the [pbf](https://
www.npmjs.com/package/pbf) library. This gives us a [Protocol Buffer](https://developers.google.com/protocol-
buffers)&mdash;an array of bytes, with attached methods to [deserialize](https://en.wikipedia.org/wiki/
Serialization) the structured data stored inside.`
```

Jump to error

```
glyphBuf = new Pbf(await d3.buffer(testHref))
```

```
md`To make sense of this data, we need to know something about how it was serialized. For now, we just copy
the glyph parsing functions from [mapbox-gl-js](https://github.com/mapbox/mapbox-gl-js) (see [Glyph PBF
parsing functions](#anchor_parsingFunctions)). Here is the result:
`
```

Jump to error

```
testGlyphs = glyphBuf.readFields(readFontstacks, [])
```

Jump to error

```
testGlyphs[200]
```

```
md`Each glyph is a separate object in the array, indexed by the character code. The native Javascript function
[String.codePointAt](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/
codePointAt) can find the index for a given character. For example, here is the object for the character 'B':`
```

Jump to error

```
testGlyphs['B'.codePointAt(0)]
```

```
md`## Bitmap QC
The \`.bitmap\` property contains the actual image data. But it is still just an array of bytes, and only one
byte per pixel, representing the alpha channel. For visualization, we can pad zeroes to generate a 4-channel
[ImageData](https://developer.mozilla.org/en-US/docs/Web/API/ImageData) object, and put it on a Canvas.`
```

Jump to error

```
showBitmap('B')
```

Jump to error

```
function showBitmap(char) {
  let codePoint = char.codePointAt(0);
  let bitmap = testGlyphs[codePoint].bitmap;
  return displayAlpha(bitmap);
}
```

```
function displayAlpha(bitmap) {
  const { width, height, data } = bitmap;
  let nPix = width * height;

  // Generate 4-channel data from the input 1-channel data
  const arr = new Uint8ClampedArray(nPix * 4); // Initialized to 0
  for (let i = 0; i < nPix; i++) {
    arr[4 * i + 3] = data[i];
    //arr[4 * i + 3] = sdfToAlpha(data[i]);
  }

  const ctx = DOM.context2d(width, height);
  let imageData = new ImageData(arr, width);
  ctx.putImageData(imageData, 0, 0);

  return ctx.canvas;
}
```

```
atlas_documentation = md`## Constructing a glyph atlas`
```

```
md`In Mapbox GL, an atlas is constructed for each map tile, containing only the glyphs needed for the labels
in that tile. For this demonstration, we just put the whole character range into the atlas.

The atlas has two components: the alpha image, and a dictionary of 'positions'.`
```

Jump to error

```
alphaImage = displayAlpha(testAtlas.image)
```

Jump to error

```
testAtlas.positions['À'.charCodeAt(0)]
```

```
md`Within each positions object, the .rect property simply defines the area within the GlyphAtlas containing
the SDF data for that character.

The .metrics property is less clear.
`


md`## Understanding the glyph .metrics object
Note the following constants relating to the generation of the SDFs and the GlyphAtlas:
- \`fontSize = 24\` : The font size at which the glyph was rendered when the SDF was generated
- \`GLYPH_PBF_PADDING = 3\` : The extension of the raw SDF bitmap beyond the limits of the actual glyph
- \`glyphPadding = 1\` : An additional padding of 1 pixel around each glyph, added when constructing the
GlyphAtlas

These values are hard-coded defaults. For the first two, see the [\`RenderSDF\` call in node-fontnik](https://
github.com/mapbox/node-fontnik/blob/master/src/glyphs.cpp), and the (header-only) [definition of \`RenderSDF\`
in sdf-glyph-foundry](https://github.com/mapbox/sdf-glyph-foundry/blob/master/include/mapbox/glyph_foundry.hpp).
For \`glyphPadding\`, see the [GlyphAtlas class in mapbox-gl-js](https://github.com/mapbox/mapbox-gl-js/blob/
master/src/render/glyph_atlas.js)

The values in the .metrics object are now:
- \`width = rect.w - 2 * (GLPYH_PBF_PADDING + glyphPadding)\` :   The actual character width, before any
padding
- \`height = rect.h - 2 * (GLYPH_PBF_PADDING + glyphPadding)\` :   The actual character height, before any
padding
- \`advance\` :   The number of pixels to move before drawing the next character. See the [mapbox-gl-native
documentation](https://github.com/mapbox/mapbox-gl-native/wiki/Text-Rendering#shaping). Note that this is a
constant value, with no [kerning](https://en.wikipedia.org/wiki/Kerning), and the sign does **not** change with
[text direction](https://developer.mozilla.org/en-US/docs/Web/CSS/direction)!
- \`left\` :   The shift of the left edge of the glyph relative to the cursor position. This corresponds to
xMin in the [OpenType definition](https://opentype.js.org/glyph-inspector.html), but scaled to pixels, assuming
a font size of 24 pixels.
- \`top\` :   The vertical position of the top edge of the glyph, measured relative to a line somewhere
above the tallest glyph
`
```

```
md`## Positioning the glyph on the Canvas
To position the text vertically, we need to determine the y-axis used for \`metrics.top\`.

The values of \`metrics.top\` are negative, with values for small characters like 'x' being more negative than the values for tall characters like 'A'. This suggests that the reference point for \`metrics.top\` is above the line of text, with \`y\` increasing upwards.

Simple trial and error reveals the following distances from the y-origin to various metrics of the [Noto Sans](https://fonts.google.com/specimen/Noto+Sans) font:
- [Cap height](https://en.wikipedia.org/wiki/Cap_height): y = -9
- [x-height](https://en.wikipedia.org/wiki/X-height): y = -13
- [Mean line](https://en.wikipedia.org/wiki/Mean_line): y = -20
- Alphabetic [baseline](https://en.wikipedia.org/wiki/Baseline_%28typography%29): y = -26
- Bottom of text (including [descenders](https://en.wikipedia.org/wiki/Descender)): y = -32
`


md`The below picture helps visualize each of these metrics. On the left are the SDFs displayed directly; on the right are the same characters as rendered by Canvas2D. Note the padding around each SDF. The yellow dotted lines indicate the actual glyph limits.
`
```

```
{
  const ctx = DOM.context2d(width, 200);
  const draw = initdrawChar(ctx, 26);

  const scale = 4.0;
  const yBaseline = 130.5;
  var xCursor = 20;

  ctx.strokeStyle = "red";
  const text = "jÀClgx";
  text.split("").forEach(char => {
    xCursor += draw(char, xCursor, yBaseline, scale);
  });

  let fontSize = 24 * scale;
  ctx.font = fontSize + "px Noto Sans";
  ctx.fillText(text, xCursor, yBaseline);

  // Draw some lines for reference
  const x0 = 20;
  const length = (xCursor - 20 + 10) * 2;
  ctx.textBaseline = "middle";
  ctx.font = "14px Noto Sans";
  function drawLine(y, color, name) {
    ctx.lineWidth = 1;
    ctx.strokeStyle = color;
    ctx.beginPath();
    ctx.moveTo(x0, y);
    ctx.lineTo(x0 + length, y);
    ctx.stroke();
    ctx.fillStyle = color;
    ctx.fillText(name, x0 + length, y);
  }
  ctx.setLineDash([]);
  drawLine(yBaseline - scale * (9 + 17), "red", "   0 origin");
```

```
  drawLine(yBaseline + scale * 0, "blue", " -26 baseline");
  drawLine(yBaseline - scale * 17, "blue", "  -9 cap height");
  ctx.setLineDash([4, 4]);
  drawLine(yBaseline - scale * 13, "blue", " -13 x-height");
  drawLine(yBaseline - scale * 6, "green", " -20 mean line");
  drawLine(yBaseline + scale * 6, "green", " -32 bottom");

  return ctx.canvas;
}

md`Once the relevant y-offsets are known, they can be used to replicate the various options of the [Canvas2D
textBaseline property](https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/
textBaseline). For example, to replicate the default \`textBaseline = "alphabetic"\` behavior, with the
baseline aligned at \`y0\`, the top of the SDF should be positioned as follows:

\`y_sdf = y0 - 26 - metrics.top - GLYPH_PBF_PADDING - glyphPadding;\`
`
```

```
sdfValues_documentation = md`## SDF values
The value of the SDF is an unsigned 8-bit integer. To convert this to a pixel distance from the glyph outline,
we again have to trace back through a few hard-coded constants in mapbox-gl and node-fontnik.

In [node-fontnik](https://github.com/mapbox/node-fontnik/), the [RenderSDF call](https://github.com/mapbox/node-
fontnik/blob/master/src/glyphs.cpp) is as follows:
\`\`\`cpp
    sdf_glyph_foundry::RenderSDF(glyph, 24, 3, 0.25, ft_face);
\`\`\`

where the RenderSDF signature is given in [sdf-glyph-foundry](https://github.com/mapbox/sdf-glyph-foundry/blob/
master/include/mapbox/glyph_foundry.hpp) as
\`\`\`cpp
    struct glyph_info;
    void RenderSDF(glyph_info &glyph,
                   int size,
                   int buffer,
                   float cutoff,
                   FT_Face ft_face);
\`\`\`

This RenderSDF function is from [freetype](https://www.freetype.org/), a widely-used C library. But rather than
try to trace back through a massive C library, let's try a shortcut: [fontnik](https://github.com/mapbox/
fontnik) is a smaller, pure JavaScript library that appears to replicate the main functionality of node-fontnik.
We will assume (!) that the API and hard-coded constants are consistent, and see what we get.

In fontnik, the [glyphToSDF function](https://github.com/mapbox/fontnik/blob/master/lib/sdf.js) appears to be
replicating node-fontnik's RenderSDF.
\`\`\`javascript
exports.glyphToSDF = function(glyph, fontSize, buffer, cutoff) {...
\`\`\`

Assuming that \`fontsize = 24, buffer = 3, cutoff = 0.25\` as in the node-fontnik call, let's see what values
are computed for each pixel. glyphToSDF generates the pixel \`data\` array by a call to the ringsToSDF function:
\`\`\`javascript
```

```
    info.data = ringsToSDF(rings, info.glyphWidth, info.glyphHeight, buffer, cutoff);
\`\`\`

The ringsToSDF function computes the data values as follows:
\`\`\`javascript
    var radius = 8;
```

```
    // ... For each pixel ...
        var d = minDistanceToLineSegment(tree, p, radius) * (256 / radius);

        var inside = polyContainsPoint(rings, { x: x + offset, y: y + offset });
        if (inside) d = -d;

        d += cutoff * 256;

        data[i] = 255 - d; // Will be clamped to 0-255
```

Plugging in the constants, then, we have

```javascript
    sdf = 255 - (distance * 32 + 64);
```

or,

```javascript
    distance = (191 - sdf) / 32;
```

md`Presumably the \`distance\` values are in SDF pixels. We can confirm by looking at one slice across the raw bitmap for the character 'I':`

Jump to error

```
sliceOfI = testGlyphs['I'.codePointAt(0)].bitmap.data.slice(12 * 12, 12 * 13)
```

md`As expected, the numbers on either side of the glyph are decreasing in steps of 32. Converting these to SDF pixels,`

Jump to error

```
Array.from(sliceOfI).map(d => (191 - d) / 32)
```

```
md`Conversion to opacity values will depend on the actual displayed size of the glyph. The antialiasing taper
width should be one screen pixel, centered on the glyph outline.`
```

```
function sdfToAlpha(val, scale = 1.0) {
  // Scale is the ratio of the displayed pixel size to the native (24px) glyph size,
  // Or, the inverse of the change in the texture coordinate per screen pixel
  let distance = scale * ((191 - val) / 32);
  let clamp = Math.min(Math.max(0, 0.5 - distance), 1);
  return clamp * 255;
}
```

Jump to error

```
sliceOfI.map(d => sdfToAlpha(d))
```

```
md`## Missing glyph metrics
The current metrics.top property is not well-defined. There are several open issues related to this in mapbox-
gl. See for example issue [#191](https://github.com/mapbox/mapbox-gl-js/issues/191).

What would be a better property? One simple option, based on the principle of "let other people do your
documentation for you," would be to add a yMin property to each glyph, following the [opentype definition]
(https://opentype.js.org/glyph-inspector.html), with \`y = 0\` at the alphabetic baseline. (The current
metrics.top property could be retained for backwards compatibility).

This yMin property by itself would enable the default \`textBaseline = "alphabetic"\` behavior. Implementation
of the [other options](https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/textBaseline)
would require additional properties relating to the *font* rather than to the individual glyph, such as the
extreme values of yMin and yMax over all characters in the font.
`
```

```
md`---

## Atlas constructor and glyph drawing functions`
```

```
testAtlas = {
  const padding = 1;

  const positions = {};
  const bins = [];

  testGlyphs.forEach(glyph => {
    const { id, bitmap, metrics } = glyph;
    const { width, height } = bitmap;
    if (width === 0 || height === 0) return;

    const bin = { x: 0, y: 0, w: width + 2 * padding, h: height + 2 * padding };
    bins.push(bin);
    positions[id] = { rect: bin, metrics };
  });

  const { w, h } = potpack(bins);
  const image = new AlphaImage({ width: w || 1, height: h || 1 });

  testGlyphs.forEach(glyph => {
    const { id, bitmap, metrics } = glyph;
    const bin = positions[id].rect;
    let srcPt = { x: 0, y: 0 };
    let dstPt = { x: bin.x + padding, y: bin.y + padding };
    AlphaImage.copy(bitmap, image, srcPt, dstPt, bitmap);
  });

  return { image, positions };
}
```

```
function initdrawChar(ctx, yShift) {
  const glyphPadding = 1.0;
  const rectBuffer = GLYPH_PBF_BORDER + glyphPadding;

  return function(character, xCursor, yBaseline, scale) {
    const {
      rect: { x, y, w, h },
      metrics: { left, top, advance }
    } = testAtlas.positions[character.charCodeAt(0)];

    let dx = xCursor + (left - rectBuffer) * scale;
    let dy = yBaseline - (top + rectBuffer + yShift) * scale;

    ctx.drawImage(alphaImage, x, y, w, h, dx, dy, w * scale, h * scale);

    // Draw the bounding box of the character itself (not including padding)
    ctx.strokeStyle = "yellow";
    ctx.setLineDash([1, 2]);
    ctx.strokeRect(
      dx + rectBuffer * scale,
      dy + rectBuffer * scale,
      (w - 2 * rectBuffer) * scale,
      (h - 2 * rectBuffer) * scale
    );

    return advance * scale;
  };
}
```

```
anchor_parsingFunctions = md`## Glyph PBF parsing functions`
```

```
md`Three relevant functions copied from mapbox-gl-js/src/style/parse_glyph_pbf.js`
```

```
function readFontstacks(tag, glyphs, pbf) {
  // Usage: pbf.readFields(readFontStacks, [])
  if (tag === 1) pbf.readMessage(readFontstack, glyphs);
}


function readFontstack(tag, glyphs, pbf) {
  if (tag !== 3) return;

  const { id, bitmap, width, height, left, top, advance } = pbf.readMessage(
    readGlyph,
    {}
  );
  const borders = 2 * GLYPH_PBF_BORDER;
  const size = { width: width + borders, height: height + borders };
  glyphs.push({
    id,
    bitmap: new AlphaImage(size, bitmap),
    metrics: { width, height, left, top, advance }
  });
}


function readGlyph(tag, glyph, pbf) {
  if (tag === 1) glyph.id = pbf.readVarint();
  else if (tag === 2) glyph.bitmap = pbf.readBytes();
  else if (tag === 3) glyph.width = pbf.readVarint();
  else if (tag === 4) glyph.height = pbf.readVarint();
  else if (tag === 5) glyph.left = pbf.readSVarint();
  else if (tag === 6) glyph.top = pbf.readSVarint();
  else if (tag === 7) glyph.advance = pbf.readVarint();
}


md`The Alpha Image class is defined in mapbox-gl-js/src/util/image.js`
```

```
function createImage(image, { width, height }, channels, data) {
  if (!data) {
    data = new Uint8Array(width * height * channels);
  } else if (data.length !== width * height * channels) {
    throw new RangeError('mismatched image size');
  }
  return Object.assign(image, { width, height, data });
}


function resizeImage(image, { width, height }, channels) {
  if (width === image.width && height === image.height) return;

  const newImage = createImage({}, { width, height }, channels);

  const size = {
    width: Math.min(image.width, width),
    height: Math.min(image.height, height)
  };

  copyImage(image, newImage, { x: 0, y: 0 }, { x: 0, y: 0 }, size, channels);

  Object.assign(image, { width, height, data: newImage.data });
}
```

```
function copyImage(srcImg, dstImg, srcPt, dstPt, size, channels) {
  if (size.width === 0 || size.height === 0) return dstImg;

  if (
    size.width > srcImg.width ||
    size.height > srcImg.height ||
    srcPt.x > srcImg.width - size.width ||
    srcPt.y > srcImg.height - size.height
  ) {
    throw new RangeError('out of range source coordinates for image copy');
  }

  if (
    size.width > dstImg.width ||
    size.height > dstImg.height ||
    dstPt.x > dstImg.width - size.width ||
    dstPt.y > dstImg.height - size.height
  ) {
    throw new RangeError('out of range destination coordinates for image copy');
  }

  const srcData = srcImg.data;
  const dstData = dstImg.data;

  console.assert(
    srcData !== dstData,
    "copyImage: src and dst data are identical!"
  );

  for (let y = 0; y < size.height; y++) {
    const srcOffset = ((srcPt.y + y) * srcImg.width + srcPt.x) * channels;
    const dstOffset = ((dstPt.y + y) * dstImg.width + dstPt.x) * channels;
    for (let i = 0; i < size.width * channels; i++) {
      dstData[dstOffset + i] = srcData[srcOffset + i];
    }
```

```
  }

  return dstImg;
}

class AlphaImage {
  constructor(size, data) {
    createImage(this, size, 1, data);
  }

  resize(size) {
    resizeImage(this, size, 1);
  }

  clone() {
    return new AlphaImage(
      { width: this.width, height: this.height },
      new Uint8Array(this.data)
    );
  }

  static copy(srcImg, dstImg, srcPt, dstPt, size) {
    copyImage(srcImg, dstImg, srcPt, dstPt, size, 1);
  }
}


md`## Other references`
```

```
md`
- [Mapbox glyph management](https://github.com/mapbox/mapbox-gl-native/wiki/Text-Rendering)
- [Mapbox label placement](https://blog.mapbox.com/map-label-placement-in-mapbox-gl-c6f843a7caaa)
- [Optimum line breaks for prettier labels](https://blog.mapbox.com/beautifying-map-labels-with-better-line-
breaking-2a6ce3ed432)
- [Label collision detection](https://github.com/mapbox/mapbox-gl-native/wiki/Collision-Detection)

Overall, the mapbox-gl-native wiki seems to have a lot of useful explanations
`


md`## Data`


GLYPH_PBF_BORDER = 3


testHrefs = [
  ["Noto Sans Regular", "0-255"],
  ["Noto Sans Regular", "256-511"],
  ["Noto Sans Bold", "0-255"],
  ["Noto Sans Regular", "512-767"],
  ["Noto Sans Bold", "256-511"],
  ["Noto Sans Regular", "1536-1791"],
  ["Noto Sans Regular", "11520-11775"],
  ["Noto Sans Regular", "65024-65279"]
]


html`<style>
@import url('https://fonts.googleapis.com/css?family=Noto+Sans&display=swap');
</style>
<!--HTML element with text is needed to make sure the font actually loads-->
<div style='font-family: "Noto Sans";'>Noto Sans: The quick brown fox jumped over the lazy dog.</div>`


md`## Imports`
```

```
potpack = require('potpack')
```

```
Pbf = require('pbf')
```

```
d3 = require("d3-fetch@1")
```