

Restyling Utah’s Vector Basemap for MapLibre

Dark and warm-light themes for UGRC’s Lite basemap, plus a browser-based style editor

Pukar Bhandari

pukar.bhandari@outlook.com

2026-09-29

Table of contents

1 Try the editor	1
2 The themes	1
3 The editor	2
4 Using a downloaded style	2

Utah’s UGRC publishes a very good free vector basemap, the “Lite” basemap, as three Esri vector-tile services: [VectorHillshade](#), [LiteBase](#) and [LiteLabels](#). It has one look. If your app is dark, or you want something warmer, you are rewriting hundreds of layer styles by hand.

[ugrc-styles](#) generates alternate themes from the live UGRC services, and ships a visual editor so you can tweak, compare and download them without touching JSON.

1 Try the editor

Pick a built-in theme (or drop in your own style), click a layer to edit its colors, opacity, filters or zoom range, and use **Compare against** to swipe your edit against another theme or UGRC’s own live basemap.

[Open app GitHub](#)

2 The themes

- **shadow**: a dark restyling built from the color tokens of Protomaps’ dark flavor.
- **sol**: a warm light restyling based on CARTO’s Voyager colors, using UGRC’s colorful [Vector_Overlay](#) icon set in place of the muted grayscale icons.

Neither theme vendors UGRC’s style definitions. Every build **fetches the three services live**, then rewrites their [paint](#) colors against the theme’s palette. The palette itself also comes from a live source (Protomaps’ flavor, or CARTO’s public Voyager style JSON), plus a small file of theme-specific tokens that source has no equivalent for. Only the generated output is committed, so it can be used without running any code.

Sprites are the other half. Highway shields and base icons are raster PNGs, which can't be recolored through paint properties, so the generator reworks the sprite sheets too (inverting them for a dark theme, for example).

3 The editor

The editor is a single static page. It can:

- group layers the way UGRC's own layer ids do (for example "PARKS & REC") and filter them by id;
- edit colors, opacity, numbers, enums, filters and zoom ranges, with a raw-JSON fallback for anything else;
- **apply one edit to every layer that shares the same value**, which matters because UGRC splits one road class across a dozen or more zoom-band layers;
- inspect the map: hover or click to see which layer is drawing what is under the cursor, then jump to it;
- swipe-compare against any theme, UGRC's live basemap, or your upload, with an address search on either map;
- autosave for crash recovery, and download one combined style or the three separate service files.

4 Using a downloaded style

A style JSON is a set of instructions that MapLibre reads and renders live, still pulling the actual data from UGRC. There is one UGRC-specific gotcha: the vector source is shaped the way Esri's `VectorTileServer` publishes it, which Esri's own tools resolve but generic renderers do not. MapLibre fails with `Failed to parse URL from tile/...` until the source is rewritten into the `tiles` array the style spec expects:

```
fetch("./ugrc-lite-edited.json")
  .then((r) => r.json())
  .then((style) => {
    for (const [id, src] of Object.entries(style.sources)) {
      if (src.type === "vector" && typeof src.url === "string") {
        style.sources[id] = { ...src, tiles: [src.url + "tile/{z}/{y}/{x}.pbf"] };
        delete style.sources[id].url;
      }
    }
    new maplibregl.Map({ container: "map", style, center: [-111.891, 40.761],
      zoom: 10 });
  });
```

Line 1

Whatever combined style you downloaded from the editor.

Line 4

Find every vector source that still uses Esri's `url` form.

Line 6

Rewrite it into an explicit `tiles` template.

Line 10

Hand the fixed style to MapLibre, centered on Salt Lake City.

For the fastest look with no code, drop the file into Maputnik. Regenerating the themes is `uv run ugrc-styles`, and adding a theme is a new directory under `config/themes/`. The code is on GitHub.