Vector Databases for Maply

This document discusses the Maply Vector Database (MVD) format, what is is, how it is read, and the command line tool used to generate it.

MVD is used to store and retrieve simple vector data appropriate for display as a map on mobile devices. Its emphasis is on speed and display and is not intended for data interchange.

We'll start with the program used to generate MVD.

carto_vector_dice

This is a command line program contained in the WhirlyGlobe-Maply github repository. It constructs Maply Vector Databases from Mapnik XML config files.

Mapnix config files contain references to source data as well as style information. At present, we support reading shapefiles, but PostGIS is likely in the future. If the program encounters a data type it can't read, it will let you know.

The program starts by reading your Mapnik config file and trying to understand the contents. If it runs into something tricky, it'll let you know immediately. Mapnik has a lot of features, only a subset of which we support.

After reading the config we iterate over the levels, looking for layers that have data within those levels. We'll chop each of the inputs that qualify into the *targetdir* directory.

The styles are applied while the input data is sorted and chopped. The chopped shapefiles contain enough information to render the map, along with a styles.json file.

Finally, we convert the chopped shapefiles into a MVD sqlite database.

The important command line switches are as follows.

targetdir	Directory we're chopping the shapefiles into
targetdb	Name of the sqlite database we'll build from the chopped shapefiles.
config	Mapnik XML config file
levels <min> <max>	The levels we want to build.

Example database:

carto_vector_dice -targetdir db_tiles -targetdb db_tiles.sqlite -config MyConfig.xml -levels 6 12

That will build a database in spherical mercator (web mercator) for levels 6 through 12. It will use the Mapnik config file MyConfig.xml and write out to db_tiles.sqlite. Intermediate Shapefiles will go out to the db_tiles directory.

Once built, you can load the db_tiles.sqlite file into Maply.

Maply Vector Database Support

Once built, the MVD sqlite file is very easy to use. In a Maply based app, just create a MaplyVectorTiles object with the database, add it to a MaplyQuadPagingLayer and then add that to the view controller. That looks like this:

```
MaplyVectorTiles *vecTiles = [[MaplyVectorTiles alloc] initWithDatabase:name];
if (vecTiles)
{
    MaplyQuadPagingLayer *layer = [[MaplyQuadPagingLayer alloc]
      initWithCoordSystem:[[MaplySphericalMercator alloc] initWebStandard]
      delegate:vecTiles];
    [baseViewC addLayer:layer];
}
```

SQLite Structure

The data is stored in a sqlite database with specific tables and fields. Every database is a quad tree, starting with tile (0,0,0) at the top and working down. Not all levels or tiles are required to be present.

The tables and fields in the sqlite database are as follows.

manifest Table

Used to store basic information about the database itself. This table has one row with the following:

minx	Extents in the local coordinate system (tilesrs)
miny	
maxx	
maxy	
compressed	1 if the individual tiles are zipped.
minlevel	The minimum level present. This is often not 0.
maxlevel	The maximum level present.
gridsrs	Spatial Reference System description for the grid. This is probably

	Web Mercator.
tilesrs	Spatial Reference System description for the data in the tiles. Yes, it usually differs. This is typically WGS84 geographic. That is, longitude and latitude values in radians.
version	Version number of compatibility

layers Table

You're allowed to store multiple layers in the MVD. This table lists all of the layers present. Layers live in their own tables, so this is important. The only field is "name" and there will be one row per layer.

attributes Table

For efficiency we store all the attribute names and types in one place and then reference those elsewhere. The only fields in this table are "name" and "type". Look at the code for a description of the type.

Each row in this table designates an attribute that can be encountered in the tiles.

styles Table

Individual pieces of vector data reference styles via the attributes table. This takes the form of attributes such as "style0", "style1", and so forth. Those are entries into this table.

The table itself is fairly simple, with only two fields: "name" and "style". The name is constructed from the Mapnik XML used to generate the style. The style is a JSON version of the XML Symbolizers found in the Mapnik XML.

A typical style might look like this:

```
{
        "type": "LineSymbolizer",
        "tilegeom": "add",
        "stroke-width": 4,
        "stroke-opacity": 0.4,
        "stroke": "#d8d3d3"
}
```

Most of those fields come from the Mapnik XML Symbolizers with a few added by us. In this case "tilegeom" refers to whether the vectors are per-tile or additive.

Vector Tables

The vector tables themselves are named <layer>_table where <layer> is the name of a given layer. The fields are as follows.

x	The horizontal tile location at this level
y	The vertical tile location at this level
level	The level (starting from zero) for this tile
quadindex	An index that contains x,y,level in one number of quick lookup
data	The vector data itself. This is a blob in a specialized format. Consult the code for details on the format if you must.

Vector tables can be sparse, with individual tiles or whole levels missing.