Open Zoning

Software and data standards for identifying housing capacity

# Abstract

100 words

# Introduction

Describe the need

Describe prior work

Describe challenges

Usefulness and significance

# Data structures

The OZFS data standard includes three files: a file with a \*.zoning extension to describe the zoning regulations for a particular municipality; a file with a \*.parcel extension to describe the geometry of all parcels (or all parcels of interest) within a municipality; and a file with a \*.building extension to describe the geometry of a proposed building.

## Zoning regulations

Zoning data are encoded in a separate \*.zoning file for each municipality. The \*.zoning file takes the structure of geojson file, which offers the advantages of a nested data structure. The specific structure of the \*.zoning file is illustrated in [Figure 1](#fig-zoning-structure).

|  |
| --- |
| Figure 1: Structure of the zoning file |

The top level of the file is an array with five key-value pairs. type takes the value of “FeatureCollection” as for all geojson files. muni\_name is the name of the municipality this zoning code refers to. date is the most recent date on which the zoning regulations described in the file are known to have been valid.

definitions is an optional array of definitions of terms (primarily height) that may very from one municipality to the next and are defined in the text of the zoning code.

features contains information on each zoning district, where each district represented by an array of three elements: type, properties, and geometry. As for all geojson files, the value for the type key will be “Feature”. The geometry key takes an array with the boundaries of each district. The value for the properties key is an array with the following key-value pairs: dist\_name, dist\_abbr, res\_uses, planned\_dev, and constraints. dist\_name is the name of the district and can be omitted if there is a defined value for dist\_abbr. dist\_abbr is the abbreviated name of the district and can be omitted if there is a defined value for dist\_name. res\_uses is a list of residential land uses that are allowed in the district including one or more of 1\_unit (single-family detached homes), 2\_unit (two-unit residential buildings) 3\_unit (three-unit residential buildings), 4\_plus (four or more units), and/or townhome (single-family attached). planned\_dev is an optional binary variable, where a value of true indicates that this is a planned development district (where the entire district will be developed by a single developer who negotiates constraints directly with the municipality). If this this key is missing, the assumption is that the district is *not* a planned development district. constraints is an array of constraints that define building geometries and more detailed land uses. This key must be present if the value of the planned\_dev key is not true.

The definitions array (defined for the municipality as a whole) and the constraints array (defined for each district –other than planned development districts– in the features array) are described in greater detail below.

### Height definitions

There may be terms that are used in many different zoning codes, but with definitions that vary across municipalities. Height is one such term. Others may be included in future extensions of the data standard. Height is defined in an array under the definitions key as illustrated in [Figure 1](#fig-zoning-structure). Municipalities may define height for each of multiple roof types (e.g. flat, hip, mansard, gable, skillion, or gambrel). These roof types are illustrated in [Figure 2](#fig-roofs).

|  |
| --- |
| Figure 2: Roof types defined in the zoning file |

For each roof type that is included, an expression must be given to indicate how to calculate the building height from one or more of the values of height\_top (the distance from the ground to the highest point on the roof), height\_eave (the distance from the ground to the lowest point on the roof), height deck (the distance from the ground to the top of the mansard on a mansard roof), and/or height\_tower (the greatest distance *from the roof* to the top of any chimneys, antennae, towers, or mechanical structures.

As an example, if height for a roof type is defined as the midpoint between the height of the eave and the height of the top of the roof, exclusive of any chimneys of mechanical structures, the expression would be "0.5 \* (height\_top + height\_eave)". If the height of a building with a mansard roof is defined as the height of the deck, the expression would be "height\_deck". A building’s roof type is not defined in the zoning code (or if no definitions are included in the zoning file), the height of the top of the roof, exclusive of any chimneys, antennae, towers, or mechanical structures will be used as the building height.

### Constraints

Constraints are described by numeric values or expressions and are stored in the structure illustrated in [Figure 3](#fig-constraints). Possible constraints include setback\_front (the front setback), far (the floor area ratio), and lot\_cov\_bldg (the lot coverage). Appendix A includes the variable names that have been defined fro the \*.zoning file, together with their descriptions. Each constraint can be defined for one or more of the uses that are allowed within the district (for cases where, for example, the required setback or floor-area ratio is different for single-family buildings than for multi-family buildings). To further complicate matters, for each combination of constraint and use there may be multiple conditions that may apply (for example, different setbacks may apply to corner lots). There is an array for each combination of constraints, uses, uses, and conditions that identifies one or both of the minimum and maximum values for that constraint.

|  |
| --- |
| Figure 3: Data structure for constraints in the \*.zoning file |

Minimum and maximum values for constraints are stored as arrays including the following key-value pairs:

* condition: The condition under which the minimum or maximum value applies. This key is required if the number of elements in the min\_value (or max\_value) array is greater than one. The condition will be an expression (in Python syntax) defining the condition under which the minimum (or maximum) value applies. Appendix B includes a full list of the of the variables that can be used in constraint and condition expressions, along with a description of each variable. If the condition under which the value applies cannot be described with one of those variables, it should be described using the more\_restrictive key, with a value of dependent for the criterion key.
* expression: A list of possible values. These can either be constant numeric values or expressions referring to variables such as far (floor area ratio), height (the building height), or lot\_width (the width of the parcel). Refer to Appendix B for a full list of variables that can be used in these expressions.
* criterion: This key is required if the list of expressions has a more than one element in it. It is a character string that can take one of three values: min, max, or dependent. A value of min indicates that the governing constraint is the minimum of the possible values listed in the expression key. A value of maximum indicates that the governing constraint is the maximum of the possible values listed in the expression key. A value of dependent means that the selection of the governing constraint depends on a criteria that may not be available in the \*.zoning file, the \*.bldg file, or the \*.parcel file.
* more\_restrictive: This key is required if the criterion key takes a value of dependent. It is a character string that describes the situation in which the more restrictive expression would apply. As an example, if the maximum height is generally 36 feet, but it is 24 feet when the lot is within a quarter mile of the waterfront, the max\_val values under height for expression would be [24, 36], the criterion value would be "dependent", and the value for more\_restrictive would be something like “within a quarter mile of waterfront,” since for max\_val expressions, the more restictive value is the lower value.

## Parcel geometry

Parcel geometry data representing parcel boundaries as polygons are commonly available in GIS files from state, county, or municipal open-data portals. These require some pre-processing for zoning analysis because applying required setbacks to determine the buildable area of a parcel requires information not only about the shape of the parcel, but also about its orientation with respect to the street, since zoning codes will specify difference setbacks for the front, sides, and rear of a parcel, respectively.

For use with the zoneR and zonepy packages, parcels must be represented in a geojson file that includes, for each parcel, line strings representing each parcel edge (front, back, and side(s) and a point representing the parcel centroid. All features have a parcel\_id key with a value that uniquely identifies which parcel each edge or centroid is associated with.

Each feature in the parcel dataset will also have a key side that can take one of six values:

* front indicates that this is a line string representing the front of a parcel.
* rear indicates that this is line string representing the rear of a parcel.
* interior side indicates that this is a line string representing the interior side of a parcel (the side adjacent to another parcel).
* exterior side indicates that this is a line string representing the exterior side of a parcel. Only a corner lot can have an exterior side. This is the side of a corner lot that is adjacent (and approximately parallel) to the street that is not indicated in the parcel’s address.
* unknown indicates that this is a line string representing the side of a parcel that has not been classified into one of the above categories (for irregular parcel geometries and/or parcels where the relationship to the adjacent street network is unclear).
* centroid indicates that this is a point representing the parcel’s centroid.

Parcel centroids have three additional key/value pairs. lot\_width indicates the width of the parcel in feet. lot\_depth indicates the depth of the parcel in feet. lot\_area indicates the area of the parcel in acres.

## Building characteristics

Building characteristics for a single building are stored in json file with the structure illustrated in [Figure 4](#fig-bldg-data). The file includes three arrays: bldg\_info includes information on the characteristics of the overall building (building dimensions and number of parking spaces within the structure). unit\_info includes information in each type of unit within the building, level\_info contains information on each level within the building, and parking\_info optionally contains information about each on-site parking area.

|  |
| --- |
| Figure 4: Structure of JSON file defining building attributes |

### Building dimensions

The building information array includes the height in feet from the ground to the top of the building, as well as the building width, and the building depth. There is also an optional key to indicate the number of parking spaces contained within the building’s structure (i.e. in a garage).

As noted in the section on zoning constraints, there are differences among zoning codes with regards to how a building’s height is defined for various roof types. A building’s roof type must be identified in the roof\_type key. For all roof types other than a flat roof, the eave height must also be specified in the height\_eave key.

If the building includes towers, chimneys, antennas, or mechanical structures, the (maximum) height of these (from the roof) can optionally be specified with the height\_tower key.

### Unit information

The unit array includes an array specifying the gross floor area, number of bedrooms, and number of units for each unit type in the building, where units are classified as being of the same type if they have the same number of bedrooms and the same floor area.

### Level information

The level array includes, for each level of the building, a two-element array with the level number and the gross floor area of that level. Above-ground levels are numbered with positive sequential numbers beginning at one (for the lowest above-ground level), and below-ground level are numbered with numbers decreasing from negative one (for the level closest to the ground).

# Analysis methods

This part gets into how the analysis works.

# Illustrative Example

To illustrate the methods above, we define a four-unit residential building and identify all parcels across two adjacent municipalities in Texas (Dallas and Irving) where that building would be compliant with existing zoning regulations. [Figure 5](#fig-boundary-map) shows the extents of these two cities.

|  |
| --- |
| Figure 5: Boundaries of municipalites included in illustrative example |

# Opportunities for future development

Parking analysis

The building data structure can store information about on-site parking, but we don’t handle that

Directly calculating building attributes from building geometry

Grade - all of our analysis assumes a flat grade - height calculations may differ depending on grade.

# Conclusion

Return to usefulness and use cases.

References