Evolution of the OSM Data Model

by Jochen Topf <jochen@topf.org>

Table of Contents

1 Introduction

We all know and love the OpenStreetMap data model with its nodes, ways, relations, tags, and changesets. But nothing is ever perfect and we have found some problems with it over the years which we might want to address as a community. We are not going to change the open tagging model which has made OSM so successful and we don't want to make the OSM data model more complicated, arguably we want to make it even simpler. Or at least more in tune with the mental model that most mappers and users have already.

The goal of this effort is to build on the strengths of OSM and remove some of the difficulties of the system, not to reinvent Wikidata or switch wholesale to a more classical GIS. Despite all the problems OSM might have, its success shows that it is generally working extremely well. So we want to make it fit for continued growth – not fundamentally change it. The changes proposed here are technical changes that will affect how we write our software, but most of them should not be noticeable to the average mapper or data user, except maybe that things work smoother and quicker.

The goal, in the end, is to keep OSM as that great resource that can be used not only by multi-billion-dollar companies, but by the student who wants to create a map of the world on their notebook or the activist with their donated second-hand computer. Everybody should be able to download the OSM data for the whole planet and use it for their novel and interesting applications without any intermediary involved. For that to work, the data has to be as easy to work with as possible and processing should not need too many resources.

This paper was commissioned by the EWG (Engineering Working Group) of the OSMF. It is written for a technical audience, mostly software developers, with a deep background in OSM technologies.

This paper is based on my many years of experience with OSM including a lot of software development. Over the years I had many discussions with other OSM community members on the topics discussed in this paper, especially after my talks *Towards an Area Datatype for OSM* at SotM 2013 in Birmingham and *Modding the OSM Data Model* at SotM 2018 in Milano and during the writing of this paper. The topics discussed here are not new, but new aspects come up all the time, and my thinking about them has changed over the years. The paper tries to be a snapshot of my current thinking and, more generally, the current state of the discussion in the community, but, of course, I can't claim that I can capture that all. This paper should be understood not as something final, but the next step in the discussion.

While writing this paper a lot of topics were suggested that it could address. But much of this is out of scope. This is about the OSM data model, the nodes, ways, relations, tags, changesets. It is not about anything user facing, it is not about tagging, it is not about the OSM website. The API is only in scope so far as it talks about those OSM data model issues. Other parts of the API, such as user management, notes, GPX upload, etc. are not in scope. (That's why this is paper is not called "Ideas for API 0.7"[1] or something like that). I agree that there are lots of other things in OSM which could be improved, but this paper is not about them.

1 https://wiki.openstreetmap.org/wiki/API_v0.7

2 Background

This chapter contains some useful background information before we tackle the questions of data model change.

2.1 The OSM Data Model

The OSM data model isn't that complex. But if we want to improve it we need to understand a lot of details of the model itself and how it is used in practice, how the data is stored in OSM files and in databases and so on. This chapter gives a terse overview of the data model and relevant issues. It is not intended as an introduction for people who don't know the data model, but as a refresher for people who do and to define some terms we are going to use throughout the paper.

2.1.1 Overview

OSM stores geographic features in *Objects* (or sometimes called *Elements*). The current OSM data model knows three types of Objects: *Nodes*, *Ways*, and *Relations*. Nodes are used for points of interest and other point-shaped objects like shops, individual trees or traffic lights. Ways are used for linear-shaped objects like roads and smaller rivers as well as for polygon-shaped objects like buildings and landuse areas when the way is closed. Relations are used for "everything else", i. e. more complex geometric objects like multipolygons and turn restrictions.

Each object has several properties: a unique *Id*, a list of key-value *Tags*, a *Version*, *Timestamp*, *User Id*, and *Changeset Id.*

2.1.2 Object Lifetime

Objects are *created* with *version* 1, and every time they are *modified*, a new version is created. There is no version 0. Old versions are never deleted but kept in the database. When an object is *deleted*, a new version is created with the *visible* flag set to false. Each object version is also given the current *timestamp* when it is created.

There is nothing to stop anybody from uploading a new version of an object with a completely different semantic meaning, i. e. with different tags and/or a different location (although this is frowned upon).

Deleted objects can be revived simply by uploading a new version of the object but, unless this is to correct a mistake, this is usually not a good idea.

2.1.3 Ids

Each object in OSM has a unique 64 bit signed integer id. The id spaces for the different object types are separate, so to identify an object uniquely you need to know the object type and id. Ids are issued by the central OSM database starting from 1. The id 0 is not used.

Negative ids are not used in published OSM data, they are used in special cases, usually as temporary ids by editors before the data is uploaded. When using the changeset upload API call[2] to update the OSM database, those temporary ids are replaced by the final ids that the database generates.

Ids are used to identify objects in the database uniquely, especially when some kind of change is to be done to those objects and to be able to refer from one object to other objects (nodes in ways, members in relations). Besides that, the ids have no meaning.

2.1.4 Nodes

In addition to the properties, nodes have a pair of coordinates in the WGS84 lon/lat coordinate system. Nodes are the only objects that have a geographic location. Most nodes (more that 97%) don't have any tags, they are only used to store a location which is used in parent ways or relations.

Locations in OSM are stored internally as pairs of 32 bit signed integers. Converting this encoding to floating point numbers allows 7 digits precision after the decimal point. This only needs half of the space of a IEEE-754 double precision floating point number that most systems use for coordinates. It is still enough to specify locations to about 1cm precision[3], more than enough for our use case.

2.1.5 Ways

In addition to the properties, ways have an ordered list of node ids pointing to their *Member Nodes*. The same node id can appear multiple times in the list. If the first and last ids are the same, the way is *closed*. Ways can have between 1 and 2000 nodes. The geometry of a way can always be interpreted as a line (LineString) (which might be invalid if it contains less than two separate points).

If a way is closed, it can be interpreted as an area, a Polygon with a single outer ring and no inner rings. Whether that makes sense depends on the tags.

2.1.6 Relations

In addition to the properties, relations have an ordered list of up to 32,000 (type, id, role) tuples pointing to their *Members*. Members can be of any OSM object type. The role is a UTF-8 string with a maximum of 255 characters.

Because relations can have other relations as members, arbitrarily deep and complex objects can be formed, including loops. A relation can even have itself as a member.

Although this is only a convention, practically all relations (>99.9%) have a `type` tag. This tag describes at a higher level what kind of "thing" the relation is supposed to be and how the relation and its members are to be interpreted in terms of their geometry and semantics. The most often used types are `multipolygon`, `restriction` (for turn restrictions), `route` (public transport routes, hiking and biking routes, etc.), and `boundary` which together make up more than 85% of all relations.

2 https://wiki.openstreetmap.org/wiki/API_v0.6#Diff_upload:_POST_/api/0.6/changeset/#id/upload
3 https://wiki.openstreetmap.org/wiki/Precision_of_coordinates

It is sometimes useful to think of these relation types as data types in their own right. So for instance, a relation with `type=multipolygon` should only have way members, no node members, and the way members are interpreted in a certain way, in this case they must assemble into a valid multipolygon. There is nothing in the API actually stopping you from violating these rules, but then a lof of software will not be able to handle those types of relations.

2.1.7 Tags

Any OSM object can have zero or more tags. There is no upper limit on the number of tags. Tags are (*Key, Value*) tuples, both key and value are UTF-8 strings with zero to 255 Unicode characters. Certain characters are not allowed because they are not allowed in XML documents.[4]

Keys are unique in a tag set on an object. (There was a time when the OSM data model allowed the same key multiple times, but this was changed early on.)

2.1.8 Changesets

Changesets contain metadata about sets of changes to the OSM database. Every version of every object contains a changeset ID referring to the changeset set in which that version was created. Note that unlike what you might expect, changes in changesets are not transactional, so changes from different changesets can and do interleave. Currently no changes to the changeset system are proposed, so we'll leave out the details here.

2.1.9 History of the Data Model

The current OSM data model with its nodes, ways, and relations has evolved from an earlier model with nodes, segments (connecting exactly two nodes), and later ways (an ordered list of segments). Segments were removed and relations added in 2005 with some smaller changes as well as the introduction of changesets in 2007.

2.2 The GIS "Simple Features" Data Model

Most Geographic Information Systems (GIS) use a data model that is different from the one used in OSM. Because OSM data is often used in those other systems and has to be converted into formats these systems can understand, it is important to understand how it works. Any changes that we might want to do to the OSM data model should take the similarities and differences of those systems into account.

It is also a model we can learn from and that can help us think about any future OSM data model. We'll refer to this model in the remainder of this document so it makes sense to outline it here.

The base of that model is the *Simple Features (Access)* model describing the types of geometries, their relations and operations on them. For a complete description see https://en.wikipedia.org/wiki/Simple_Features and the standards linked from there.

The Simple Feature model knows seven geometry object types: the *Point*, *LineString*, and *Polygon* (a zero-, one-, and two-dimensional objects) as well as their variants *MultiPoint*, *MultiLineString*,

4 https://github.com/openstreetmap/openstreetmap-website/blob/master/app/validators/characters_validator.rb

and *MultiPolygon* (plus the *GeometryCollection*).[5] Polygons are made up of *(Linear) Rings*, one *Outer Ring* and zero or more *Inner Rings*. MultiPolygons can have several outer rings. Often rings have a defined *Winding Order* allowing you to differentiate the inside and the outside of that ring, for instance as you go around the boundary of a polygon, everything on the left side is inside and everything on the right side is outside the polygon. So outer and inner rings will have opposite winding order.

Unlike in OSM, all coordinates of a geometry are stored "inside" those geometries, not by reference to other objects.

Although not part of the Simple Feature model, the usual model in GIS uses *Layers* as collections of *Features* of the same type. Each *Feature* has a geometry of one of the types mentioned above, often a (unique) *Id* and a list of *Attributes*. Unlike with OSM tags, all features in a layer have the same list of attributes. Attributes are typed, usually types such as String, Integers, Real Numbers, etc. are allowed, but different systems have different types available.

When talking about Simple Feature data types in this paper, I'll use the capitalized names Point, MultiPolygon, etc. and reserve the lower case point, multipolygon, etc. as more generic names.

2.3 Geometric Validity

Geometries as used in the Simple Feature Model have important properties, they should be *Simple* and *Valid*. Basically this means the data is "clean" and there are no "weird things" going on like self-intersections or spikes.[6] Not all systems require geometries to be valid, but validity is important because many operations on geometries (such as calculation the area of a polygon or cutting a geometry around a bounding box) require the input geometries to be simple and valid, otherwise the operations will fail, or worse, silently produce bad data. Invalid geometries can also lead to broken rendering.

Checking the geometric validity of data is not always cheap. For a complex multipolygon, this can take some time. Ideally the database would only contain simple and valid geometric objects so that everybody relying on the data doesn't have to do the check themselves before using the data. But that would put the burden of the check on the core OSMF infrastructure, either the database or the API servers.

2.4 Growth of OSM

Rethinking the OSM data model is based on our experience with the current data model over more than a decade. In that time OSM has changed and some of the changes make some data model changes more important. But OSM is not standing still, it will grow. So we'll also have to look into likely scenarios for the future to see whether changes make sense and how they might be beneficial to OSM ten years from now.

5 There are extensions for curved geometries etc. which don't interest us here.
6 For a good overview see https://postgis.net/docs/manual-3.1/postgis_usage.html#OGC_Validity

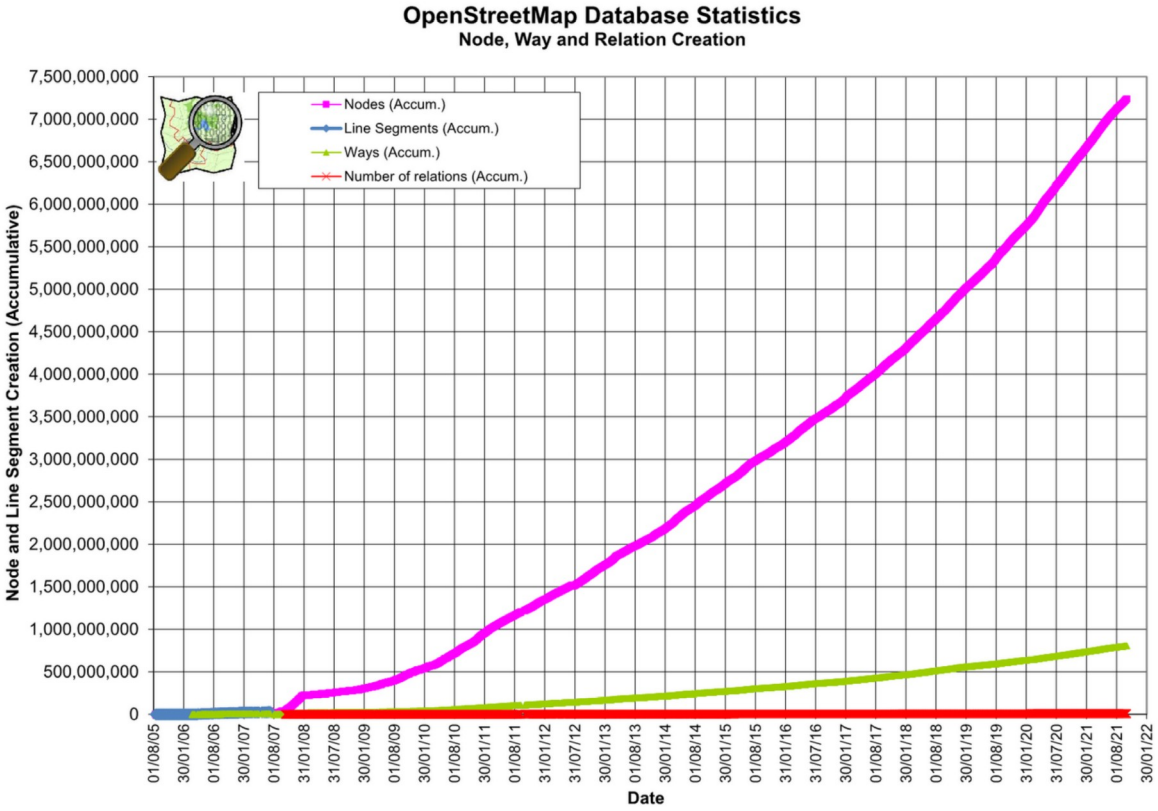

Figure 1: OSM Database Stats from https://wiki.openstreetmap.org/wiki/Stats

The OSM database is growing. Today we have over 7.5 billion nodes and we are adding something like 1 billion nodes per year.

Today just storing the locations of those nodes in memory needs a machine with 64 GB RAM and we'll soon need more.

How much data will OSM contain in the future? There are different ways of approaching this question. Let's look at data density for Germany, which is pretty well mapped. The PBF file is about 3.6 GB, about 5% of the 64 GB planet file. Germany has 360,000 km² and a population of about 80 million. India's PBF file is only 1.1 GB. India has 10 times the area of Germany and 15 times the population. So sooner or later it will need 40 or 50 GB almost doubling the size of our current data. There are 8 billion people on the planet, a factor of 100 compared to Germany, so that means we'd need a 360 GB planet file. Of course these are only very rough calculations and we don't know how long it will take to add all this data to OSM, but sooner or later we'll have an order of magnitude more data.

Here is a different calculation, maybe more short term: One major growth area in OSM are buildings[7], the most popular tag by far is `building`. There are currently about 500 million buildings in OSM which need on average about 5 nodes per building.[8] Microsoft is working on a project to auto-

7 https://taginfo.openstreetmap.org/keys/building#chronology

8 Actually about 6 to 7 nodes per building. But many buildings touch each other so nodes can be reused.

matically generate building footprints[9] for large parts of the world with the explicit goal of making them available to OSM. The last statistic I found (from January 2022[10]) mentions more than 265 million buildings in their data sets, and only a small part of the world is covered. If we assume that each building needs 5 nodes, we already have more than 1.3 billion nodes right there.[11] How many buildings are there in the world? Nobody knows, but a guess somewhere in the billions would probably be in the right ball park. So to map all those we can easily expect to see the number of nodes double from what we have now.

2.5 The Environment we are Working in

With all ideas and proposed changes in this paper we have to keep in mind the environment we are working in. Tackling this will be a huge project by OSM standards. OpenStreetMap is still mostly run by volunteers and some (uncoordinated) efforts by companies. Running a large project like this is something new to OSM and we should limit the scope as much as possible. Nevertheless paid work will be needed for this project.

We also have to keep in mind that only some of the work will be done in the “core” of OSM, the infrastructure and software run by OSMF and core contributors. A lot of changes will have to be done outside this core and in many cases we don’t even have a way of knowing who is out there doing something with OSM, never mind being able to contact them to inform them about changes. And of course we can’t tell them what to do either.

One important thing we always have to keep in mind is the centralized nature of the OSM infrastructure. Everything in OSM goes through one central database on a single server. This architecture has served us well for a long time and it keeps working. There might be reasons to change it, develop a more distributed system with all its advantages and disadvantages, but for the purpose of this project this is way out of scope. We have to live with the potential bottleneck of this system. So any change proposed here must not put undue burden on the OSMF infrastructure, especially the central database.

9 https://www.microsoft.com/en-us/maps/building-footprints
10 https://blogs.bing.com/maps/2022-01/New-and-updated-Building-Footprints
11 Some of the buildings in the Microsoft data set will already be in OSM, but presumably they are doing this predominantly for areas underserved in OSM.

3 Issues with the Current OSM Data Model

There are several issues or shortcomings of the current data model and the way it is used in the real world which are described in this chapter. Not all of them are hugely important and many of them will only show up in some use cases. This chapter tries to list the most commonly encountered issues (as far as I am aware of them). I am using the word “issue” here on purpose, not the word “problem”, because it depends on the context whether these issues are a problem. I’ll leave the evaluation of and possible remedies for these issues for later chapters.

3.1 References Between Objects

OSM has a huge amount of objects. Of course the major reason is that the world is large and there are a lot of objects to map. But some of that is because of our data model. There are more than 7.5 billion nodes in the OSM database, only about 2.5% of these nodes have any tags. So most nodes are only there to store the locations needed for ways and relations. But they still have their own identity and metadata (timestamp, user id, etc.). Storing this takes a lot of disk space and, often, memory while processing the OSM data.

Ideally each OSM object would stand on its own, modeling one object in the real world. For many use cases, for instance rendering many types of maps, this would allow processing one object at a time (“streaming”), without having to look at any other OSM object at that moment. But in our current data model this is usually not the case: all ways depend on their member nodes for basically any kind of processing, and all relations depend on their members which might again refer to even more objects.

This makes many operations on OSM data much more expensive than they would otherwise be. Instead of reading from a file you need some kind of database (in memory or on disk) indexed by object ids to quickly assemble the data you need from several objects. This is one of the major stumbling blocks when working with OSM data. A typical approach for many programs is to keep a huge table of all node locations indexed by the node id in memory.[12] Then, when the ways are processed, the locations of those nodes can be looked up quickly. With the around 7.5 billion nodes such a table needs around 60 GB in the simplest case. There are more memory-efficient ways of storing this information, because nodes with similar ids are often close to each other. A clever implementation might be able to reduce memory consumption by half, but for the price of making the creation of that table and the lookup somewhat more expensive. And half of 60GB is still a lot of memory. If this “lookup table” has to work with updates from OSM it will need even more memory. With the continuing growth of OSM this “lookup table” approach becomes more and more cumbersome and a growing bottleneck in the processing of OSM data. For people with limited computing resources, it becomes increasingly difficult to work with all of the OSM data. Not everybody has access to or can afford large cloud machines.

The references between relations and their members have similar issues as with the nodes in ways. Arguably they are even more complicated, because of the different types of objects involved and the

12 The data can also be stored on disk and this is also done. But access will be much slower and there is still the need for quite a bit of memory for caching.

complex web of direct and indirect references. But because there are comparatively few relations in OSM, this issue is less pressing.

The references are an especially thorny issue in the context of changing OSM data. Changing the geometry of a way by moving a member point doesn't actually change the way itself, but the member node. So the way appears unchanged, and the node gets a new version number. Change files will only contain the changed node and it falls into the responsibility of the software interpreting the change file to propagate the changes in the node to its parent way. This can get complicated rather quickly and often necessitates keeping a complete database of all OSM objects just to be able to process changes correctly.

The Simple Feature model works differently. In that model each feature includes all the geometry data as well as all attributes for that object.

Storing the nodes with their metadata without any tags in PBF format takes about 6 bytes per node.[13] This is extremely efficient. A "naive" implementation would need at least something like 32 bytes (8 bytes id + 8 bytes location + 4 * 4 bytes version/timestamp/changeset/uid). So storing these nodes on disk isn't the problem. As long as we always read them in the order they are in the file, access is simple. But random access into that file isn't possible due to the large blocks of data that need to be decoded to get just a little bit of information out.

Of course the OSM model has its benefits:

- We actually have a topological dataset where, for instance, streets build an actual road network, not just a bunch of streets that happen to end where another street starts and so on. This could be important for routing and related use cases.
- We can model more complex real-world objects using relations and their members, such as turn restrictions and public transport routes. It is expected that these kinds of uses will rise in the future.
- A location change of one vertex of a 1000-vertex polygon only necessitates the transmission of one new node, instead of a new 1000-vertex geometry.

3.2 Missing Area Data Type

OSM doesn't have a dedicated area datatype.[14] Instead area objects are modeled as either

- a closed way, or
- a relation of `type=multipolygon`.[15]

Compared to a Simple Feature (Multi)Polygon, the closed way model is rather simple and can not be used for all cases. Relations have to be used a) when an inner ring is needed (example: building with courtyard), b) when multiple outer rings are needed, c) when there are more than 2000 points

13 Measured by creating a PBF file with only the nodes from the planet with all tags removed and deviding the file size by the number of nodes in the file.

14 Very early versions of the OSM data model did actually have an area datatype but that was removed with the switch away from segments. It was never really used.

15 Relations of `type=boundary` can also be understood as multipolygons, from a geometric perspective they are often handles in the same way.

in the ring, d) when we want to use existing features as (part of) the boundary (example: river is part of the boundary between countries).

There are several issues with this:

- Having two completely different ways of doing the same thing (or similar things) makes everything more complex. All software has to handle the different cases. An object can start out as, say, a rough outline of a forest modeled as closed way, when refinements are made later, it might have to be turned into a multipolygon. On the other hand a small polygon (say a building) is a very different beast than a large multipolygon spanning a whole continent (like a country boundary). So maybe having different ways of expressing the same thing isn't so bad. But it then becomes the question whether the choice we have made with separating the "space" of all areas into those exact two types is the best choice.
- A closed way can model either a line that happens to come back to itself (like a a roundabout), or it can be the area itself (like a building). In fact it can be both, because we can tag the same way with line-like tags (`highway`) as well as polygon-like tags (`landuse`). So it depends on the tags what it actually is. This makes it impossible to implement a generic conversion to the GIS Simple Feature data models which always sees LineStrings and Polygons as different geometry types. And everything rendering the map (like an OSM editor) has to know about all sorts of tags to even decide whether to draw an outline of the object or fill it in. To make matters worse, some tags (for instance `man_made=pier`) are used for both linear and 2-dimensional features. Sometimes ways do double duty as outline and area having two sets of tags (for instance `barrier=fence` and `leisure=park`). The `area=no/yes` tag can be used to differentiate between the two cases, but it is only used rarely (on about 0.2% of the ways).
- There is a mismatch between the terms Polygon and MultiPolygon as used by the GIS model vs. the terms used by OSM. An OSM way can only model a polygon with a single outer and no inner ring. An OSM multipolygon relation actually models either a Polygon or a MultiPolygon in the GIS sense with any amounts of inner and outer rings. Sometimes a multipolygon relation has to be used just because a polygon is too large to model as a closed way (2000 node limit).
- Another consequence of the tag dependency is that we can't automatically check the geometric validity of polygons. A self-intersecting line that happens to end up in the same place where it started (think of a bus route describing a figure 8) is not invalid, but a self-intersecting polygon boundary is. Neither an editor nor the database itself can check validity before uploading/inserting new data. This puts the burden of these checks on the data users.
- When closed OSM ways are used as polygons, the winding order is not specified. Strictly speaking we don't know whether a polygon describes the area of, say a building, or the area of the whole world without that building. While this sounds esoteric it prevents correctly handing some corner cases like polygons crossing the anti-meridian or very large polygons. (The coastline uses a different way of solving this.)

- Multipolygon relations should have the roles `inner` and `outer` on their rings. This is not always the case and was a problem historically, but these days 99.9% of members have either of those roles (whether they are all correct is a different matter[16]).
- There is some processing overhead for multipolygon relations for the double indirection from relation to member ways to member nodes. This is especially pronounced for small (multi)polygons with more than one ring that have to be modeled as relations.
- Because of the difficulty with handling large polygons, large areas are often simply split into smaller (way based) polygons. This often happens, for instance, with waterways or large forests. This means there isn't one OSM object for a real world feature but several which puts the burden on the data processing tool chain which possibly has to re-assemble that data based on tags and geometric connection. This can also limit the rendering options, because only part of the outline of a polygon is the real "border" of that polygon, some of the outline is directly connected to a neighboring polygon, so the border doesn't exists in the real world and must not be rendered.
- Any operation on multipolygon relations, like cutting at a bounding box, requires that the multipolygon geometry be assembled from its parts first. Otherwise results can be invalid.

3.3 Coastline

The coastline in OSM is a very special case. Geometrically, what it "really" is, is a huge multipolygon of all the land areas of the world. (Or alternatively, a huge multipolygon of all the seas of the world.) But we don't map it as such because the multipolygon relation would be huge (more than 68 million nodes, an order of magnitude larger than the largest currently existing relation). Instead we use ways tagged `natural=coastline` and their direction to model this.

That's a valid way of doing things and it has some advantages, but it is different from how all other large (multi)polygons are mapped, which always confuses users. And it needs a very different tool chain[17] to process which leads to different update cycles and a dependency of most data users on an external site[18] which runs this tool chain for them.

3.4 Fuzzy and Large Areas

OSM usually maps reasonably small and well-defined things. But there are other geometric objects, like "the Alps" or "the Atlantic Ocean".

The Alps are a mountain range, the edges are somewhat fuzzy. We probably don't want to show them on the map, but we still might want to put a large label on the map, roughly covering the area.

The Atlantic Ocean is obviously bordered by the coastline of, say, Morocco. Apart from the normal fuzziness of the coastline, that's pretty well defined. But where does the Atlantic end and the Pacific

16 https://tools.geofabrik.de/osmi/?view=areas&overlays=role_should_be_inner%2Crole_should_be_outer
17 https://osmcode.org/osmcoastline/
18 https://osmdata.openstreetmap.de/

begin? Or is it the Southern Ocean? And is the Caribbean Sea part of the Atlantic or something else?[19]

We have objects with some fuzziness and some “hard edges”. You can map these with conventional relations, maybe adding some fuzziness marker to the member roles. But there is a problem with the size of those things. An “Atlantic Ocean” relation would be huge if we want use the precise edges. If we don’t, the perfectionists will never be happy. Mapping these objects with all detail, on the other hand, makes them prone to breakage like the coastline and large multipolygons and makes them harder to use because of all the, for most use cases, unnecessary detail.[20]

If we are only looking at the size of those things, they take up a space between what we currently have as already unwieldy huge multipolygon relations and the coastline which has its own special format, because it is too big to model as a multipolygon relation.

3.5 Building a Navigation Graph

For navigation you need a graph that your routing engine will use to find the best route on. You should think that OSM data already comes in the form of a graph with the OSM nodes as nodes of the routing graph (after all that’s where their name comes from) and the ways as edges. But that graph is much larger than needed. For a routing graph you only need nodes to be at those places where more than two edges meet, everything else is overhead. So a long windy road up a hill might have hundreds if not thousands of OSM nodes in it, but for the routing engine it is just one edge in the routing graph. (You still might want to look at the geometry of the road to, say, decide that the average speed on that roads is slower because it is windy, but that’s a different issue.)

So while OSM does have an underlying “proper” graph model, that graph is actually not useful for a routing engine. When the data is prepared for routing the graph needs to thinned out. Basically the software needs to mark for each node whether it is an actual node in the graph sense or whether it is only there to make up the geometry of the road. This is a fairly expensive process and the existence of nodes as separate entities doesn’t really help us much here compared to just using coordinates.

Data formats used by proprietary routing engines differentiate between the routing graph and the geometry and keep those as separate entities (that are still connected in their data model though). This allows the same data to be used for rendering and routing on several levels.

3.6 Mismatch Between OSM and Usual GIS Model

Because there is a lot of software using the usual Simple Feature GIS data model, OSM data is often transformed into that data model as a first processing step before doing other operations. In some cases the opposite is also done, for instance when proprietary data is used with some of the excellent Open Source routing software that only understands the OSM data format.

19 For some of these fuzzy areas there are “official” definitions where they start and end. But only because there is an official definition somewhere that doesn’t mean that OSM has to use that or that there can’t be other definitions. That discussion is out of scope for this paper. Here we are talking about how to actually put the object into our data model assuming we have decided that we want to add it to OSM at some location.

20 See also https://github.com/gravitystorm/openstreetmap-carto/issues/4546

While there are some similarities between the models, there are many mismatches and a roundtrip conversion from one to the other format and back is usually not possible.

Here is a comparison between the models:

OSM model	GIS model	Notes
OSM object	Feature	Often more than one OSM object is needed for a feature (way plus member nodes, relations with their direct and indirect members).
Node	Point	Only when nodes have tags, also used to “store” coordinates for ways and relations.
-	MultiPoint	MultiPoint concept doesn’t exist in OSM, but isn’t used that often in classical GIS systems either. Could be modeled using a relation, or better, by simply giving all participating nodes a certain tag.
Way	LineString	But needs nodes. When closed, the way can also be a Polygon (or both). Ways can be invalid LineStrings (when they have fewer then two distinct locations).
Way	Polygon	When way is closed and has tags that make it a polygon. There is no list of all tags which make it a polygon. Can be both. Ways cannot model Polygons with inner rings.
Relation	MultiPolygon	When relation has `type=multipolygon` tag (or sometimes `type=boundary`). Can be a Polygon if it has only a single outer ring.
Relation	?	Can be lots of other things not expressible in GIS data model (or only as GeometryCollection type).
Relation	MultiLineString	Relations with `type=route` can often be modeled as a LineString or MultiLineString, but relations can have nodes or other relations as members which don’t fit into that model.
Tag	Attribute	Some tags describe the attributes of an objects. But any number of tags allowed while attribute list is usually fixed.
Tag	Layer	Some tags are sometimes better thought of as defining the layer in the GIS model.
Unique Ids	Ids	No general requirement for unique ids, although they are often used. GIS model sometimes allows for non-integer ids.

In the last years the GIS community has been slowly moving towards more flexible data models where some of the limitations of the classical GIS model have been relaxed. Examples are the use of JSON-encoded attribute values (which the PostgreSQL database supports for instance) or the FlatBuffers data format which allows each feature in a data set to have different attributes. But most software and formats are not able to deal with this, so the mismatch between the data formats is still there.

3.7 Level of Detail

The level of detail in which we are mapping the world is not defined in any way. Sometimes you have a single closed way modeling a huge forest, sometimes each tree in a park is mapped as individual node. What's more important is that we have different levels of details side by side in the data. A way with a `highway` tag can mean: Here is a road with lots of typical other stuff around it like a sidewalk. Or it can mean just the one carriageway with more bicycle lanes, sidewalks, bus bays, parking lots mapped in detail around it.

We have no way to signify the level of detail that's used in a particular case and we have no way of marking larger level objects that are in some way assemblies of smaller objects (except, in theory, with relations).

So far this hasn't been a huge problem for many use cases, but some, like correct lane-based navigation or navigation for pedestrians are really difficult.

3.8 Limited Coordinate Precision

The OSM data model uses coordinates with a precision of 7 digits after the decimal point. This allows coordinates to be stored in 32 bit integers, so 8 bytes are enough for a coordinate pair. Compared to the common IEEE double precision real numbers this needs only half the space which allows for significant savings in large databases.

The 7 digits give a precision of 1 centimeter or better, so in practice this is usually not a problem with the kinds of things OSM is mapping. But with more and more micro-mapping in OSM (for instance for indoor maps) this could in the future become more of a problem. It also prevents round-trip conversion of other data formats into OSM and back.

Because this doesn't seem to be a problem in practice and the benefits are obvious, I will not consider this further in this document.

3.9 Coordinate Wraparound at the Anti-Meridian

In GIS systems the world is often flat. Objects that cross the anti-meridian at 180° west/east are not possible, because most software would interpret them in the wrong way. This is not a problem specific to OpenStreetMap, other GIS software suffers from the same problem. The underlying issue is simply that the world doesn't end somewhere but coordinates must, which introduces an ambiguity: Does a line going from 179° East to 179° West go all the way around the world or is it the smaller direct connection. Intuitively we might choose the shorter connection, but that's not how computers work.

This issue isn't a huge one because there is not much out there that we want to map in the middle of the Pacific and that actually straddles the anti-meridian, but it is at least annoying for some people. Solutions would have to take into account more than OSM though, because OSM data is processed in so many other systems. I will not consider this further in this document.

3.10 Relations and Extracts

Nothing seems more “natural” than creating and using geographic extracts of OSM data (typically by bounding box or by country). But this is more difficult than it seems at first, because only the nodes have an actual geographic location. The algorithm looks something like this:

1. All nodes inside the area we are interested in have to end up in the extract.
2. All ways that reference the nodes from step 1 have to be in the extract, too.
3. But what about nodes outside the area that are referenced by the ways from step 2? We can either add them to the abstract completing the ways or not add them, cutting the ways at the boundary.
4. In the next step we have to decide which relations should end up in the extract. Relations with all their members in the area must be in the extract, of course. And relations that have no (direct or indirect) members must not appear in the extract. But what about relations that have some (direct or indirect) members in the extract? This is a problem in practice, because some huge relations (for instance boundaries) tend to “pull” lots of data into extracts that doesn’t really belong there.

Ideally we would be able to cut the geometries at the boundary of the area we are interested in. But this can only be done if we know what this geometry is. There are several problems with this:

- A LineString is cut differently than a Polygon (but for a way we don’t know this unless we look at the tags)
- We can only cut polygons sensibly if they are valid.
- When cutting a polygon we have to add new points at the boundary or outside the area to complete the polygon. But those points don’t exist in the original OSM data. We would have to “invent” new nodes for them.
- For relations of `type=multipolygon` we would need to assemble the MultiPolygon geometry from the member ways and their member nodes, then do the cut and “invent” nodes and ways to model them.
- For relations of other types than multipolygon it isn’t always clear how to interpret them geometrically. For a relation of `type=boundary` interpreted as lines cutting is easier than if we interpret the geometry as a polygon. What about more complex public transport relations or `type=associatedStreet` relations or relations of relations? Current software doesn’t interpret these types of relations in special ways because of these difficulties.

3.11 Versioning in Extracts

There is a bit of an obscure problem when using (geographic) extracts of OSM data and then updating those extracts with change files. When an object is deleted, a new version of that object with the flag `visible=false` is created. But what happens when an object is moved out of the area covered by the extract? It is not deleted, the object is still there. But as far as the extract is concerned, the object needs to be removed, so we need to “invent” a deletion change here.

By the normal rules, a moved node would get a new version number and the `visible=false` flag. But now we have two different nodes with the same version number, one “inside the extract” with `visible=false`, one “outside the extract” with the new location. For a way that has this node as a member, it is worse. There is no new version generated for that way, because only the node changed. But “inside the extract”, the way is now deleted and we need to create a new version number. We can’t “invent” a new version number, though, because that would collide with future changes of that way object.

Existing software[21] has to jump through hoops to create something that sort-of works most of the time.

3.12 Stability of Ids

One of the most of frequently mentioned problems with our current data model is the missing stability and uniqueness of ids. A shop can be tagged as a node, later the building outline is added and the tags from the shops are moved to the new way. So the shop changed not only its id, but its object type also. Or a way modeling a street is split into two ways to add different `maxspeed` tags. This is a more complex case, there is no unique id any more for the street itself. This points to the real underlying problem: What we perceive as a real world object (a building, a street with a distinct name) isn’t the “unit of mapping”. What we put on the map depends on the primitive kinds of objects our database allows: nodes, ways, and relations. Yes, we are creating a database, not just drawing a map, but our database format is still tied very much to the geographical form of a map.

Wikidata[22] works fundamentally different. Once something has been classified on Wikidata it has its Q-Code that will never change. OSM doesn’t work that way. There simply is no way to make OSM have unique and stable ids for real world objects. Our ids just work differently. Instead there are tags and they fit the bill. We can do what we have been doing for a while now: leverage Wikidata by adding Wikidata ids (Qxxxx) as tags to OSM objects. This is not perfect, because we still have the ambiguities (is the Wikidata tag referring to the building or the hairdressers?), but it works in many cases.

Note that one little part of the problem could be solved: A node that’s later changed into a way for more detailed mapping could keep its id if we wouldn’t have separate id spaces for different OSM object types, but use a joint id space for all of them. Its rather questionable whether that’s a good idea, though. Fast processing and efficient encoding of objects is easier when objects of the same type are stored and processed together and always keep their type (think of working with history data where an object changes type for instance), so keeping things separate makes sense.

3.13 File Types

OSM data is distributed in three types of files:

- *OSM Data Files* contain the current content of the OSM database at a specific point in time (or an extract thereof). Files usually use the suffix `.osm`.

21 https://docs.osmcode.org/osmium/latest/osmium-derive-changes.html
22 https://www.wikidata.org

- OSM History Files contain all versions of all OSM objects up to a specific point in time (or an extract thereof). Files usually use the suffix `.osm` or `.osh`. There is basically no difference between OSM Data Files and OSM History Files except that the latter can contain several versions of the same object.
- OSM Change Files contain the changes to the OSM data over a time range. They can contain either all changes (replication changes) or only the last change to all objects in the time range (simplified changes). Files usually use the suffix `.osc`. They differ from OSM History Files in the file format although technically they contain the same information.

These types of files can be encoded in different formats (XML, PBF, …) the details of which do not concern us here, but not all file types can be encoded in all formats. The original file formats were several different XML-based formats and they are still available. For `.osc` files they are still commonly used, because the PBF format doesn't support that.

The use of the different file format variants is confusing and leads to extra effort for implementers. It is not always clear from the file name suffix what's in a file.

3.14 Tags

A lot of the perceived simplicity of the OSM data model comes from the free tagging system. Basically anything goes, the complexity of the real world is not visible in the data model, because it only shows up in the tags. That's not an accident. It was a deliberate decision to allow OSM to be flexible and open-ended. And it is certainly part of the reason why OSM was successful. But it also comes at a price. I don't want to discuss the tagging itself in this paper, whether it is better to tag something as `landuse=forest` or `natural=wood` or what the exact differences between different tags are. Those questions are clearly out of scope here. But there are some issues around tags that "border on" the data model itself or affect our thinking about it. I want to highlight some of these issues.

3.14.1 "Main" vs. "Attribute Tags"

When mapping and using the OSM data there often is a more or less clear differentiation in our minds between what I call, for lack of better terms, "main" tags[23] and "attribute" tags. This way *is a* street, it *has* the name X and *has* the speed limit Y. The `highway` tag in this case tells us what something "is", the other tags specify additional attributes. In theory we can also say that this object is a "Main Street" object and it has the attribute "streetness", but the awkward language shows this doesn't come naturally. We can certainly discuss the details and there are many corner cases and certainly cases where an OSM object hast multiple "main" tags, but the distinction is undeniably there most of the time. Outside of OSM in the GIS world this distinction is usally expressed by putting objects in different layers which specify the main type of an object.

Typical processing of OSM data, it doesn't matter if this is in editors, renderers or whatever else, usually depends on this distinction between "main" and "attribute" tags. First the software tries to

23 Simon Poole called these "top-level tags" in his talk at SotM 2018: https://2018.stateofthemap.org/2018/T061-An_excursion_in_to_the_world_of_OSM_tagging_presets/

find the “main” tag and branches off into different processing subsystems dependent on that “main” tag. But it isn’t always clear what that “main” tag is, some tags can be both depending on context.

3.14.2 Meta Tags

Not all tags on OSM objects are about the “real world” object. Some tags are only used for “internal bookkeeping”. Historically the most important examples were the `source` and `created_by` tags, but today those are mainly handled by the changeset, although there are still more than 200 million objects with a `source` tag and its variants. Other tags in use are `attribution` and `note`. What seems to be becoming more prevalent are tags such as `start_date` (specifying when the real-world object was created) or `survey:date` (used by StreetComplete for instance to mark when a real-world object was last checked).

The problem with all these tags is that it isn’t clear what they refer to. Normal tags refer to the object as a whole. But sometimes they are meant only for parts of the object, like its location, or a specific other tag. So then we’d need to get into more and more complex tagging schemes (`start_date:name:John F Kennedy International Airport=1963-12-24`) to support that. If we’d really add a start date and end date and survey date and attribution to every tag we’d get into trouble.

Another problem is that changing something like the `survey:date` creates a new object version, although the object itself didn’t change. This triggers changes down the line in all data consumers for something that is really OSM internal and doesn’t interest most of them.

3.14.3 Data Model Tags

Relations have a somewhat special tag. The `type` tag is a kind of meta tag that can be understood as an extension of the OSM data model type system. Similar for ways we have the `area` tag which tells us if a way is of linear or area type. And the tags and roles `forward` and `backward` and their variants are really related to the order in which the nodes appear in a way or members in a relation, they are part of the geometry of the object that we can’t express in a different way.

In a way the existence of those tags points towards a deficiency in our type system. We can not explicitly express the (geometry) types other than by using those tags or roles.

You can also see it the other way around: We could get rid of specific types (nodes, ways, and relations), just have one type and use tags for the rest. A way is really just a relation that has only node members where all the roles are empty and the number of members is limited to 2000. It seems arbitrary that we have a special type for “way”, but not for “polygon”, or “multipolygon”, or something like that. This is a situation that has evolved; these distinctions are not set in stone. We can move the line between “very specific type” on the one side and “very general type plus tags” on the other to somewhere where it makes the most sense for our use case.

3.14.4 Tag Structure

In OSM tags always have a key and a value and both are Unicode strings. This is a simple system and it allows quite a lot of flexibility, because, in the end, you can always present some information in the form of a string.

Most tag keys only use (usually lower case) characters in the range a to z, the underscore (_), colon (:), and sometimes digits 0-9. OSM is an international project, so keys in other scripts are possible and sometimes, although rarely, used.

By convention the colon (:) is often used as a delimiter in a sort-of hierarchically organized key. So `name:it` is the name in Italian, all parts of an address are in tags beginning with `addr:`, and `sidewalk:left` is used to indicate the existence and type of sidewalk on the left side of the road. More than 50% of unique keys contain one or more colons.

As mentioned tag keys and values are always strings, in practice there are some conventions on which tag keys allow which kinds of tag values. This leads to a sort-of weak, unenforced type system. Typical types are boolean (`lit=yes/no`), enumerations (`highway=motorway, primary, secondary, …`), integers (`level`), numbers with optional units (`width`). But there are also rather complex schemes for values of some tags like `opening_hours`[24]. The semicolon (;) is often used when a value is really an (ordered or unordered) list of values.

All this structure in the tags keys and values isn't in the low-level data model. It makes the data model simpler, but in actual use everybody has to agree on these things anyway, so saying that tags are just strings hides the complexity that's there in reality. Now there is a good reason why we might want to keep it this way: it is something we can change more easily as our use of OSM changes and expands than if we bake those concepts into code. But there are also good reasons why we might want to think about other solutions here. Pretending the complexity isn't there doesn't help us.

Most data processing systems, whatever kind of data they work with, use a richer set of data types. They have strings, numbers (integers, floats, …), boolean values, etc. Often arrays of values and something like (possibly nested) key-value data type is supported. Users of OSM data often have to convert the OSM tag strings into something more usable in all those systems and have to deal with values that don't map to those other formats.

And we can see in the complex preset configuration files used by most editors how (at least some of) these types are handled in practice and how everybody tries to wrangle with an ad-hoc type system that has been built on top of our string-only system.

3.15 Robustness and Security

Nobody has yet conducted a comprehensive security assessment of the OSM data model and OSM data formats. The more OSM data is used the larger the chances are that problems manifest itself.

There are different possible problems. Some are more due to the data model itself, some are related to the way the data is encoded in file formats (for instance limits on block sizes in the files) and APIs. Further problems are related to the way we process the data, for example when somebody does excessive number of changes, how the id space is used, etc. A comprehensive assessment is outside the scope of this document, but we should take security considerations into account if we change anything.

Here are some issues around the data model itself:

24 https://wiki.openstreetmap.org/wiki/Key%3Aopening_hours

Number of objects	There isn't anything we can do about that, we want the database to grow. But possibly limits on number of objects uploaded in a certain time etc.
References between objects	More references need more memory in typical processing tool chains.
Number of nodes in a way	These are limited to 2000 nodes.
Number of members in a relation	Relations can have a maximum of 32,000 members.
Relation members in relations	Relations can have relation members allowing chains of relationships and loops. It is even possible to have a relation a member of itself.
Number of tags on OSM objects	There is no limit on the number of tags on an OSM object. This can easily overwhelm any kind of buffer.
Length of user names, tag keys and values and member roles and characters allowed	These can all be 255 UTF-8 encoded Unicode characters long. Because all valid Unicode characters can be encoded in a maximum of 4 bytes[25] they each can take a maximum of 1020 bytes. Certain characters are not allowed (0x00-0x08, 0x0b, 0x0c, 0x0e-0x1f, 0x7f, 0xfffe, 0xffff). This is mostly due to them not being allowed in certain XML versions.[26] There have been issues with this multiple times over the years.[27] The maximum size of 255 UTF-8 characters was a historical accident, the MySQL database we used back then couldn't support more. All Unicode characters are allowed including invisible and control characters.[28]

3.16 More Issues

There are many issues that pop up now and again that I have not considered, because they move OSM too far away from what it currently is.

One is the wish to give OSM a "time" dimension, a way to not only map what the world looks now, but what it used to look like. The Open Historical Map (OHM)[29] project is attempting something like this. This adds a huge amount of complexity for everybody while only serving a small minority of users. But it's good that OHM is experimenting with this.

Software developers love version control systems like git and there could potentially be some value in allowing something like branching and merging and related concepts in OSM. There were attempts to create something like this for the GIS world, for instance the (now defunct) GeoGit project. I do see why this could be interesting for OSM, but it would be a huge complex project and totally change the interaction inside the community. This is far beyond the scope for this paper.

25 https://en.wikipedia.org/wiki/UTF-8
26 https://en.wikipedia.org/wiki/Valid_characters_in_XML
27 https://github.com/openstreetmap/openstreetmap-website/issues/1213
https://github.com/openstreetmap/openstreetmap-website/issues/2025
28 https://en.wikipedia.org/wiki/Unicode#Security_issues
29 https://wiki.openstreetmap.org/wiki/Open_Historical_Map

4 Discussions and Possible Improvements

This chapter discusses the issues raised in chapter 3 in broader contexts and, where possible, proposes some improvements. There are quite a few issues which I have thought about and discussed with others for this paper. But in the end I am not proposing any action or, at most, minor steps. This is not because those are not important issues, but because I cannot see a clear path to any improvements. Often the goals are too vague to be actionable. I include some discussions in this paper to document what we are thinking about these and explain why some proposed solutions might not work. We should continue the discussions, maybe some solution will present itself, either inside the scope of this effort or possibly for future changes.

4.1 Getting Rid of Untagged Nodes

Nodes have three distinct uses:

- They model real world objects (like post boxes or traffic lights). What kind of real world object a node represents depends on the tags, so this makes only sense for tagged nodes.
- They “lend” their locations to other objects, the ways and relations they are members of.
- They explicitly connect real world objects to each other. So a node can connect two ways if it is a member of both. Or a tagged node (say a traffic light) can be connected to the way it is a member of (say a street). This is what we mean when we talk about topology, the connection between objects which happens not because of the location they happen to be in, but because they are explicitly connected through nodes of the same id.

Each node can have one, two, or all three of these uses.[30] 97.5% of nodes don’t have tags, so they are only there for their location (about 90% of all nodes), or for their location and topology (about 10% of all nodes).

Note: Everything I am saying here about untagged nodes (i. e. nodes with no tags at all) applies in the same way to nodes with only have one or more meta-tags like `source` or `created_by`. But this doesn’t make a big difference, it adds only about 0.002% of the nodes to the “untagged” list.

If we could get rid of all untagged nodes as distinct objects, we’d reduce the number of nodes by two orders of magnitude. The idea is pretty simple: We add the locations formerly on the member nodes to the “parent” objects themselves. So the way which currently has an “internal” list of node ids, has an “internal” list of locations instead. This has several advantages:

- Most use cases of OSM data needs the locations of the points in the way together with the way tags, for instance to render the way to a map. So most processing of OSM data starts with assembling the LineString or Polygon geometry of the way from the node locations. To do this some kind of in-memory or on-disk database of nodes is needed. If the locations are on the way itself, this step can be completely eliminated, making processing of OSM data for most use cases much simpler.

30 As I am writing this there are also about half a million “orphan” nodes that don’t have any tags and are not referenced by any other objects.

- Because changes to the geometry on the way will show up as new versions of the way and not just new versions of the member nodes, changes to ways can directly be processed. Users don't need a database any more pointing from nodes to their parent way to propagate changes of nodes to changes of the ways which are usually the objects the user cares about that actually need the change.

4.1.1 Ways vs. Relation Handling

So far we have talked only about node locations on ways. With relations the situation is similar.

The two simplest options of handling relations are:

1. We'll keep the structure of relations as it is. Everything that would be modeled as a node member of a relation today, would still require a node member in the future.
2. We'll extend the relation data type so that it can also have locations as "point members".

To see what might make sense here we'll need to look at the most common relation types (in this case I am looking at types of relation which appear at least a 100,000 times in OSM). We'll differentiate between members where it is predominantly the location we are interested in or where the actual node with its tags is interesting. We'll also look at whether the topology is important in each case.

Relation tag `type=`	**Use of nodes/locations**
`multipolygon`	Usually has only way members. Sometimes nodes members are mapped (with unclear semantics). Almost all of them have tags, so we'll probably want to point to the node.
`restriction`	Node members with role `via` are an essential part of this type of relation. Most member nodes (about 90%) have no tags, but the location and topology are important. (If the `from/to` ways of a restriction only connect in one place, the `via` node isn't strictly necessary.)
`route`	Node members are used for stops and platforms. They usually have their own tags.
`boundary`	Node members are used as `label` (only location needed) and to point to the `admin_centre` (which is a tagged node).
`associatedStreet`	Always points to a node with tags, otherwise this relation doesn't make sense.
`public_transport`	Almost all member nodes have important tags.
`site`	Almost all member nodes have tags.
`destination_sign`	Similar in structure to `type=restriction`. Most member nodes don't have tags, so the location and topology are the main use here.

The only popular relation types where the node members usually do not point to tagged nodes are the `restriction` and `destination_sign` types, both are used for navigation. For these we'll either still need untagged nodes or store the location in the relation. For all others the member nodes usually have tags anyway, so they'll still be available in the new model.

4.1.2 The Topology Question

Nodes provide topology, i. e. they provide an explicit connection between two (or more) objects. All objects referencing the same node are understood to be connected in some way. That's different than if they reference different nodes with the same location. In that case there is no strong connection between those objects, they just happen to share (part of) their location.

Currently about 10% of all nodes in the planet are in two or more ways, so these are the "candidates" for topologically important nodes. Also all tagged nodes in ways might be topologically important (for instance barriers or traffic lights), that's about 1% of all nodes. But note that these numbers vary widely across the planet. In Germany 22%, in the Netherlands 24% of all nodes are in multiple ways, probably due to the many rows of buildings and detailed landcover mapping. So if we keep all those nodes, we are not going to save much and have the additional cost of two different "points", the node-based ones and the location-only-points, which doesn't look like a very attractive solution.

We can put the locations of the nodes onto the objects that reference them, that's no problem. The cost of storing a 64 bit node id on the ways and relations is (more or less) the same as storing the locations directly.[31] But if we need the topology, that's a different matter. So let's take a look at that issue.

We have to differentiate between two cases:

- Do we need the topology when editing?
- Do users need the topology?

Let's look at editing first: When moving a node that is attached to two or more ways, those ways all "move with the node". More often than not, this is the intended behavior, for instance when a node connecting two highways is moved or when a node on the boundary of two connecting landuse polygons is moved. Today, this is just what happens in the editor and everybody expects it to happen. If you want a different behavior, you have to "split" the node first, into two nodes. If we were to store the locations inside the ways, the editor would probably by default still consider two identical locations to be the same "pseudo-node" and move them together. This isn't difficult to do. Because coordinates will still have limited precision, we can still be sure that they are the same.[32] And editors have to download all data in some bounding box we are editing anyway. So from the point of view of the user, we can still have the old behavior.

Now to the users: Let's look at the three most important use cases: Rendering, Geocoding, and Routing.

Rendering: Most rendering software is based not directly on the OSM model, but the Simple Features GIS model which doesn't support topology. This has never been a real problem[33], we routinely join ways into longer LineStrings before rendering, for instance, but that's usually done after con-

31 It is not necessarily the same when some kind of compression is involved. 64bit integers given out as consecutive ids store differently than two "randomly distributed" 32bit coordinates.

32 When coordinates are stored as floating point numbers we have to keep this in mind because of rounding issues.

33 Although in theory the topology could give us valuable information, for instance when merging smaller polygons into larger ones.

version to the Simple Feature model. I think we can safely assume that the topology is not needed for this use case.

Routing: For any kind of routing we do need a topology. We need to know whether two roads just "happen" to cross, for instance at a bridge, or whether they are connected. But just assuming that two roads with a common point in their ways connect should almost always be enough. To model a bridge we can have to ways crossing, but they don't need a common node/point where they cross (just as we do it now). Assembling a routing graph is an extra step in any case, OSM data can't really be used directly as a routing graph. And it doesn't make much difference whether we create the graph from the OSM nodes and ways or from a new format where we have to compare locations instead of node ids for this.

There is one use case where this is not quite true: When multiple levels are involved like in a building with elaborate indoor tagging. Here it could matter whether staircases between different stories connect or not connect to floors etc. But detailed indoor tagging is a minority use for OSM and we can come up with a solution for this special case using `level` tags or so.

(Reverse) Geocoding: For geocoding a different kind of topology is relevant. To locate the entrance of a building at a specific address you'll need to find the entrance node[34] on the way modeling the building outline. So what's important here is to find the connection between a tagged node and the tagged way it shares a location with. Similarly, address interpolation lines[35] connect nodes tagged with house numbers. So, as in the routing case, we have to create those connections in the geocoding database from the locations of the objects and not the node ids.

4.1.3 How Many Co-located Nodes are there?

The tool `osp-find-colocated-nodes`[36] can find all the co-located nodes and the objects referencing them. In a planet from this year there were about 3.5 million locations with two or more nodes on them. Looking at a sample of these, it seems most are just errors. To figure out the problematic cases we should fix all those cases and see what's left.

4.1.4 Alternative Solutions

Getting rid of untagged nodes is a major change in the OSM data model, so there is the question of whether it is worth it. Is there some other solution to the issues we have that might be easier to achieve?

There is always the option to solve this not in the core of OSM but outside. Convert the planet file to some format that doesn't have all those nodes any more and offer that in addition to the original planet file for download. In fact there is such a format already. The Osmium library supports a variant of the OSM format with node locations on the ways and the Osmium tool has the necessary commands to create such a file from OSM data. You still need a machine with lots of RAM to create such a file, but afterwards, people could just use that and don't have to do the way assembly themselves. This does in large part solve one of the problems with the current setup as long as you only want to download a planet file. But other issues are not solved at all, specifically anything to

34 https://wiki.openstreetmap.org/wiki/Key%3Aentrance
35 https://taginfo.openstreetmap.org/keys/addr:interpolation
36 https://github.com/osmcode/osmium-surplus

do with change files is still complicated if at all possible. There is no clear way of handling (minutely) changes with that solution. We'd have to create change files that update that file and find a way to create way versions that change whenever a node changes in the original. At some point we'd have to ask ourselves whether it isn't better to solve the problem at the root.

And even if we come up with a solution for all the issues, it is unclear who would be creating such a file regularly and reliably enough that people will actually change their software to use that format. We'd have two formats to support indefinitely.

4.1.5 Impact on Core OSMF Infrastructure

The main OSM database will need some changes. The `nodes` and `current_nodes` tables and the corresponding indexes will become much smaller, because they will only contain a fraction of the nodes. The `way_nodes` and `current_way_nodes` tables can be removed (together with any indexes on them and the foreign key constraints) and replaced by an integer array for the locations and/or a PostGIS geometry in the `ways` and `current_ways` tables, maybe storing the bounding box explicitly or something like that. And we'll need some kind of geometry index on that. Because the new `ways` and `current_ways` tables have now a variable sized field for the geometry the table entry might overflow to a TOAST table[37], but most ways are small so the table entry will be below the 2k threshold and not be TOASTed.

All of this should, in effect, reduce the size of the database and simplify its structure. On the other hand there will be more way versions, because, different from before, they change when their geometry changes, so the `ways` table will grow larger and there will be more churn on the `current_ways` table. Considering that most nodes never changed (on average there are about 1.2 versions of every node) and that several nodes in a way changing will probably often happen in the same upload, because they are changed together, this is probably not going to be a large problem. Also any change to the number of nodes does already result in a new way version (of which there are only about 1.7 per way).

All in all I expect a similar database load than before, maybe even a light reduction due to the decoupling of the nodes and ways.

In the API all calls related to ways will have to change to accommodate the changed structure.

4.1.6 Impact on Editors

Editors will have to be changed significantly. On the one hand life becomes easier for them, because there are considerably fewer nodes with their meta-data to track. On the other hand they have to decide when to make or keep a topological relationship and when not to. When a point is moved that is "used" by several nodes, ways, or relations, the editor has to decide which objects to modify. This will probably depend on the tags of the involved objects and might depend on some kind of editing mode.

If operations on any points are done in the editor, it is important that they are done in integer space, not done on floating point numbers for the coordinates. Otherwise there is the risk of breakage

37 https://www.postgresql.org/docs/current/storage-toast.html

when rounding or other arithmetic errors lead to two points that are very close, but not identical and so the topological connection is broken.

4.1.7 Impact on Mappers

The impact on mappers should be minimal. Occasional mappers will probably not see any difference, because the editors will hide them from the user. If at all, the new simpler and more straightforward behavior should match their mental model better than the old behavior did.

Power mappers who are aware of the data model with the nodes and ways will, of course, see a difference and they need to switch their mental model. But actual differences in operation will most likely be limited to complex cases involving relations and so on, but the simple day to day mapping will not change at all.

4.1.8 Impact on Data Users

Most data users will see a significant improvement. They don't have to deal with huge databases of node locations any more. Streamed processing of data become possible for more users. Users of change files can use a simplified logic.

For data users who need topological data, their code needs to change. Currently they are building their internal topological data based on the node ids, in the future they'll have to do the same based on the (integer) locations of the points.

4.2 Creating an Area Data Type

This chapter talks about creating one or more area data types. I am using the word "area" on purpose here, and not (multi)polygon, to make sure we don't mix it up with the (Multi)Polygon datatype defined by the Simple Features model.

4.2.1 Design Requirements

Apart from fixing as many of the problems mentioned earlier as possible, here are some design requirements for any new polygon solution.

- Must scale from tiny polygons (4-corner buildings) to enormous polygons spanning a whole country and allow efficient representations of all of them. This doesn't mean there has to be one data type that can do everything, but if there are several they have to work together.
- A solution that can make sure polygons are always valid is preferable, but the burden on the main database server must be taken into account.
- We must have a clear migration path from where we are now to any new solution.

4.2.2 Small Areas

There are several possible variants of area data types:

Variant WAY-WITH-FLAG: Add a "flag" to the existing way datatype. The flag specifies whether this way is a "linear" way or a "area-type" way. For migration we can add another option "un-

known” which all existing ways will initially have. For backwards-compatibility with existing systems which don’t understand the flag, it can easily be expressed as an `area=yes/no` tag. The disadvantage is that no inner rings or multiple outer rings are possible.

Variant POLYGON: Introduce a Polygon datatype with geometry semantics as in the Simple Feature standard. This gives us inner rings and the compatibility with Simple Features. With nodes which are compatible to Simple Feature Points and ways which are compatible to Simple Feature LineStrings, we have complete compatibility with (non-multi) Simple Feature geometries.

Variant MULTIPOLYGON: Introduce a MultiPolygon datatype with geometry semantics as in the Simple Feature standard. This gives us multiple outer and inner rings and the compatibility with Simple Features. Most areas will still be non-multi polygons, but that’s a subset of MultiPolygons, so that’s no problem, we just have to make sure non-multi polygons are efficiently encoded in the file formats.

I would expect that we keep the limit of 2000 nodes/points for this area datatype to keep them a manageable size. See below for larger areas.

A major advantage of a distinct area datatype would be the option of guaranteeing validity. If we switch from nodes to having the point locations inside the way and area data types, this can be done easily. While checking can be expensive computationally, this is not a major concern with a 2000 point limit, and, more importantly it doesn’t affect the central bottleneck, the database, because on every change all coordinates are available for a check in the API servers without any extra information from the database

If we keep the nodes, though, checking would mean getting all the member nodes of a way from the database when the way is created or changed and each time a node changes, all parent ways have to be re-checked. This would probably be too expensive.

For all variants, we can specify a winding order for the outer and inner rings. This would ensure that data can easily be converted into other formats that require a winding order. And it would allow us to map areas that would otherwise be ambiguous, such as areas crossing the anti-meridian or around the poles. Users will usually not see any difference as the editor can just handle this.

4.2.3 Large Areas

The proposal still leaves out the question of large (multi)polygons. The main problem with them is the difficulty of ensuring their geometric validity. Our largest multipolygons have hundreds of thousands of points, some even millions.[38] And they are easily broken. But checking validity for every small change would involve getting all the member ways and their member nodes out of the database, which is considered too expensive.

With the introduction of point locations on ways, this is becoming much more tractable. We only have to resolve a single indirection to get the full geometry, not two. But still the number of points involved and the large geometric extent of the polygons make this difficult. If we check validity in the API, we’ll also have to check validity in the editor first to give the user a sensible user interface.

38 You can use the `oat_large_area` tool from https://github.com/osmcode/osm-area-tools to find those multipolygons.

But that would mean the editor would have to download all members of all relations that even touch the bounding box the editor is currently working in.

We could probably define an area data type that allows partial downloads and edits while still preserving validity of the whole. We need to test validity on first creation and then make sure that every edit keeps that validity intact. In theory it should be possible to only test the parts that changed if they don't interact with the parts that didn't. But the details for this would need to be worked out and we'd have to come up with an API for this that allows partial changes, something quite different from what we do today.

Until we have a perfect solution maybe one that moves us in the right direction while still using multipolygon relations might be possible: If we order ways in multipolygon relations, use the member role to store forwards/backwards info (so ways can be in the multipolygons in either direction), check "connections" between ways and check that each member way is geometrically simple, we will be much better off than today, although we still can't guarantee overall validity.

It puts the burden on editors to check these things, they have to download all member ways inside the bounding box being edited (same as before) plus maybe one more "on each side". The forwards/backwards info on the roles (together with the inner/outer info which still has to be on the roles, too) helps in visualizing, because we always know what's inside and what's outside the area, even with a partial download.

It's not perfect and still easy to break on purpose, but should be more stable for normal use. We can use existing QA software for the rest.

We'd still have to decide if we want to have at least some of these checks on the API, too. But then we'd have to make those checks dependent on the `type` tag of the relation.

4.2.4 The Coastline

If we have multipolygon relations with an ordered list of member ways going into the right direction that are checked so that the connectivity is not broken, we are getting quite close to how the coastline mapping works. Only without the relation. We can think of the coastline as a "virtual multipolygon relation" where all the members are the ones with a `natural=coastline` tag. A relation with over one million members and 68 million nodes.

The coastline is somewhat special because there is only one coastline for the whole world. That's why it works without the relation, there are no extra tags we need to add to the coastline. I am not sure yet how this can turn out in the future, but maybe we can find a way of consolidating the coastline model with the multipolygon relation model.

4.2.5 Migration

The migration path for all the proposals above is reasonably clear: We'll have to inspect all closed ways and decide whether they are linear or are representing an area. For some ways this can be detected automatically, for others this has to be done manually. If a way is used both for modeling a linear feature and an area feature, the way has to be "duplicated". We'll need some tools to help with all of that.

For the WAY-WITH-FLAG variant no changes are done to multipolygon relations. For the POLYGON and MULTIPOLYGON variant candidates for migration from relations to the new data types can be automatically found and objects can be migrated automatically, manually, or semi-automatically. This can be done unhurriedly after the changeover of the data model, because multipolygon relations remain a valid way of expressing those objects.

If validity checks are introduced for multipolygon relations, we can do this in several steps:

1. Clean up existing data in a concerted effort to minimize problems later.
2. Change editing software to include the new checks for any new objects or changes made, only warn for existing objects as before.
3. Possibly change the API to check new (versions of) objects.
4. Clean up the rest of the invalid objects.
5. Now the editors and any other software can be changed to assume whatever validity we are guaranteeing (unless they work also with historical data).

4.2.6 Impact on Core OSMF Infrastructure

The impact on the core OSMF infrastructure by these proposals is reasonably small. There is the new datatype to be implemented (or the way datatype to be augmented), but most parts of the code will just be the same as in the existing way datatype. The number of objects is also not greatly affected. There will be some additional objects though, when a former way is doing double duty as line and area.

Depending on which variant is implemented, some multipolygon relations can be converted to the new area datatype. This simplifies the data, fewer relations are used.

Validity checks in the API don’t need extra database accesses if we store the geometry in the ways, so they should not have a large impact.

4.2.7 Impact on Mappers

Mappers will see no large change. For iD users most of the change will be totally hidden, because the user interface already pretends that there are area objects. For JOSM users the differences will be a bit bigger. But this should be something users will learn quickly.

Depending on variant, the largest visible change for users will be the now possible inner rings (and possibly multiple outer rings) which required a multipolygon relation before. It will take some getting used to especially for JOSM users to use the new datatype instead of creating multipolygon relations all the time.

4.2.8 Impact on Data Users

For OSM data users there are some immediate benefits. Any software working with polygons doesn’t have to validate the geometry any more. Generic processing of line vs. area data is now possible without having lists of tags. Depending on variant, the data processing is somewhat simplified because there are fewer relations with their extra indirection to handle.

4.3 Limiting Length and Allowed Characters in Strings

The limits on string length in OSM are kind of weird. They are mostly a historical accident, not from any design. For somebody who wants to create robust software they are too large, especially for user names, tag keys, and roles. On the other hand, for tag values they might be too short for some of things we are doing with them.[39] But we have been living with them for a long time and haven't seen any problems, so maybe it's better not to touch them.

Tag keys are typically short[40] and when they are not they are often clearly bogus. A maximum length of 64 or 100 bytes would be possible. But note that key length has been growing due to ever more keys with colon-separated hierarchical keys which to some extent make up for the missing complex data types in OSM. Whether this is to be encouraged is another question. But this question is definitely tied to the question of how much structure we want to have in tags.

For relation member roles the situation is similar to the keys, because roles should really be only used for structural information (like the `inner/outer` roles on multipolygon relations). In practice roles are often used as some kind of free-text marker which arguably is something to be cleaned up anyway.

I think this is a topic where much more discussion in the community is needed before we make any changes.

4.4 Database

Perhaps surprisingly at first, the current core OSM database doesn't use the PostGIS extension to store locations of nodes. Instead, node locations are stored in the `nodes` and `current_nodes` tables in a pair of 32bit integers (`latitude/longitude`) and additionally in the 64bit integer field `tile`, which contains the quad tile of the point, calculated by interweaving the bits from the latitude and longitude coordinates.[41] The `tile` field always contains the same information as the `latitude` and `longitude` fields together. In addition there is a btree index on those `tile` fields.

If we are going to store locations in ways, too, there is the question of how to store them. If we add some kind of area datatype, we'll face the same question.

One option would be to remove our home-grown system of storing location and simply use the PostGIS extension and their Point, LineString, and (Multi)Polygon data types. The main problem with that approach is that we are currently storing coordinates as integers and this has some advantages (see elsewhere in this paper) that we'd loose, because PostGIS geometries always use doubles for their coordinates. We could store the locations twice, once in our own integer-based system and once in the PostGIS data types to be able to use their indexing and processing support, but the question is whether that's worth it.

If we think about the operations that need to be supported by our geometry datatype, there aren't many. Basically the only thing we do with those coordinates is store them and find all objects inside a bounding box. No other operations are done inside the database. The API and the tools to dump

39 An `opening_hours` tag can easily have more than 100 characters.

40 https://taginfo.openstreetmap.org/reports/key_lengths#histogram

41 https://github.com/openstreetmap/openstreetmap-website/blob/master/db/functions/

the planet file and generate change files are the only access points into the database. They all basically read or write OSM objects as a whole doing the coordinate encoding and decoding in the process, but they don't ask the database to do anything more complicated with those locations. The only exception is the `/api/0.6/map` API call[42] which returns all objects inside the specified bounding box.

There are sometimes calls for extending the API and allowing more complex queries, but the consensus seems to be that it is important to keep the API small and focused on its main job, to allow editing of OSM data. Everything else can be solved outside the OSM core.

Based on all this it might make more sense to extend our home-grown system to store those geometries. We could store the coordinates in a PostgreSQL integer array. An additional array with the quad tiles and a GIN index on that array should allow reasonably fast access, but would be somewhat wasteful.

If we only think about storage we can compress the data to keep the amount of data stored small. For instance using varint[43] encoded delta values like it is done in PBF files. Most ways are small and the coordinates from one node to the next don't differ much, so the deltas between coordinates are usually small and can be encoded efficiently in a variable sized integer. The database would only see a `bytea` binary datatype and interpretation is left for the API and applications. If we'd do that, though, we need some custom code in the database for index access.

I am not sure what the best solution is, but there are several options and we can figure out the details and test them once we have decided on the other design issues in this paper.

4.5 File Formats

Most changes in the data model will result in changes to the file formats currently in use in the OSM world. Depending on the size and nature of the changes it might make sense to either adapt the old formats, invent new formats, or use existing formats defined somewhere else.

Currently the following file formats are in use:

4.5.1 XML

This is the original and still the "reference" format. There are variants for OSM data, OSM history data, and OSM changes, as well as formats for changesets. The XML formats is used for distribution of OSM data in files (planet, extracts) as well as for the API.

XML files are typically used with gzip or bzip2 compression.

When OSM was new, XML was popular. Today XML has fallen out of favor. If invented today, OSM would probably use the more popular JSON as the primary format. If larger changes to the file format are needed anyway, we can consider switching to a JSON-based format, which would allow us to get rid of the over-complex XML format and the complex parsers needed.

If we keep the XML format, any changes can be easily integrated.

42 https://wiki.openstreetmap.org/wiki/API_v0.6#Retrieving_map_data_by_bounding_box:_GET_/api/0.6/map
43 https://en.wikipedia.org/wiki/Variable-length_quantity

4.5.2 PBF

This binary format was invented as a very space-efficient encoding of the OSM data. It is widely used as a distribution format (planet, extracts). It can be used for OSM data and OSM history data, but is not used for changes.

The PBF format is rather complex but encodes the OSM data rather efficiently. We'll definitely need an extension or replacement of this format, because it is the most widely used and most efficient format we have.

We can use this opportunity to fix one of the largest problem with the PBF format: You always have to read the whole thing, even if you only want to access some of the data, for instance only the relations. PBF puts all data in blocks, but there is no index[44] and blocks are usually compressed and you need to decompress them before you know what's inside. Being able to read only certain object types would go a long way towards making processing more efficient.

4.5.3 JSON

The API allows access to OSM data in JSON format[45]. It is currently not used for planet files or other downloadable files.

If we want to use this for larger files the JSON lines[46] format is probably a good candidate.

4.5.4 OPL

This format was introduced by the Osmium family of tools. It is a very simple and terse text-based format with one OSM object per line (hence the name "opl").[47] It is mostly used for writing test cases etc. (because of its terseness compared to XML) and for processing OSM data with command line tools and simple scripts.

4.5.5 O3M/O3C

This binary format[48] was introduced by the osmc tools (osmconvert[49], osmfilter[50], osmupdate[51]) and is supported by a few other tools. There are two subtypes, the `.o5m` and the `.o5c` formats, corresponding to the `.osm` and `.osc` formats for OSM data and changes files. The format is usually used as an efficient intermediary format and not for distribution of data.

4.5.6 Order of Objects in the Files

In general OSM doesn't force a specific order on the OSM objects in a file. All file formats support putting objects into the file in any order.[52] But by far the most often used order is nodes, then ways,

44 The format has a "hook" for adding some index data, but that was never really defined or used.
45 https://wiki.openstreetmap.org/wiki/OSM_JSON
46 https://jsonlines.org/
47 https://osmcode.org/opl-file-format/
48 https://wiki.openstreetmap.org/wiki/O5m
49 https://wiki.openstreetmap.org/wiki/Osmconvert
50 https://wiki.openstreetmap.org/wiki/Osmfilter
51 https://wiki.openstreetmap.org/wiki/Osmupdate
52 Some formats are more efficient if all objects of the same type are encoded together, though.

then relations, and for each type the objects are ordered by id. A lot of software depends on this order.

If the changes suggested in this paper are implemented we should think about order in the file, what order we want to have and whether we want to require a specific order or not. The current order makes sense because we can read the file, storing the node locations while we do that, and when we come to the ways the node locations are already available and can be added to the way information. If we don't need the nodes for the ways any more, the order isn't important any more for this.

Depending on exactly what changes we want to make, we might have nodes (points), ways (linestrings), and area objects (polygons) in the future that can stand on their own completely. They don't need other objects to be interpreted, so their order doesn't matter any more. But we'll still have relations. And relations are different, they always need other objects to be interpreted. At first glance it makes sense to have them at the end, because then all the other objects the relations might need have been read already.

But doing this the other way round actually does make more sense. Because there are so many objects that are not needed by any relation, it is often better to read the relations first, remember what all the objects are that we are interested in and only store those objects when they come around later. Every time a relation is "complete", i. e. when all members are available, it can be processed.

This order of processing also makes more sense when relation members are involved that are relations themselves. We can't always make sure that relation members appear in the file before parent relations, but once we have read all relations into memory anyway, processing those cases becomes relatively easy.

But note that this type of processing only works as long as we can store all relations in memory, so it only works if the number of relations is "small". There are far fewer relations than nodes or ways, so this is currently the case. If we add more and more relations to model more complicated things, this might change in the future, though.

We might also sidestep the whole discussion about order and put some kind of index in the header of the file that tells us which object types are where. Then the order doesn't matter much any more. The downside is that streaming out a file isn't possible any more then, because we don't know yet where the parts of the file will end up when writing the header. We can put the index at the end instead, which makes streaming reads more difficult.

5 Next Steps

This study is only a first step in tackling some of these problems. Any change will need a lot of discussion in the wider OSM community and enough buy-in from the stakeholders so that we can move forward.

There is no precedent in the OSM community for such a large change to the basic data model. The largest changes so far were the removal of segments in 2007 when OSM was in its infancy and the number of users, programs, and systems affected was tiny in comparison to what we have now. The introduction of relations (also in 2007) added a major new capability, but didn't change existing ones, so it didn't disrupt existing use cases. Maybe a better comparison is the effort to get rid of old-style multipolygons which started in 2016 and was concluded in 2017.[53] All in all this took about a year, half a year for preparations like documentation, writing tools etc. and another half year of community effort to change 250.000 relations. That effort was possible, because Mapbox paid for my time organizing and running it. And while it did "only" change tags and didn't change anything in the core node/way/relation model, it arguably was a real data-model change and lots of programs dealing with OSM data including the editors were changed to adapt to this. The following draws on my experience managing this effort.

After this paper is published the next steps are up to the OSM community, especially the OSMF and the EWG who commissioned this paper. Here is the outline of a phased plan that I believe could be implemented:

5.1 Phase I: Community Consultation and Decision

The widest possible community needs to be consulted and, ultimately, be part of the decision process. This paper lays the groundwork for an informed discussion in the EWG, OSMF, and all parts of the wider OSM community. We'll need a series of workshops (online and/or in person) to discuss this paper and the whole effort with all stakeholders who are interested in this topic.

We'll need a (paid) project manager and technical lead. This is a huge project and, despite years of thinking and discussions about the issues and wide community support for changes, this project never got started because nobody had the time and resources to start and run it. Without paying at least some key people to work on it over a long period of time this project will never come off the ground. We also need to form a working group of some kind with the people who actually want to work on the details. The job of this working group would be to come up with the final goal and implementation plan for the changes.

Phase I is completed when a decision is reached to go forward with this effort including a more or less refined idea on which things we want to tackle and which we don't want to bring into this effort. For some issues this might already be quite specific, but some details can be left open for the next phase, as long as they are not needed for the decision to go ahead.

In Phase I we can already start with cleaning up some OSM data. Nodes without tags that are not referenced from anywhere can be cleaned up, co-located nodes can be checked, invalid polygon ge-

53 https://blog.jochentopf.com/2017-08-28-polygon-fixing-effort-concluded.html

ometries fixed, conflicting line vs. area tagging can be resolved. Regardless of which changes are implemented (or not) later, those things are useful anyway and the cleanup helps prepare for later data model changes. Starting early means there is more time and any issues encountered during this cleanup can further inform the data model changes and change process.

Once a decision is made and the necessary funds are allocated we can move to Phase II.

Phase I will need about 6 to 9 months.

5.2 Phase II: Detailed Planning and Preparations

In Phase II we do the detailed design for all planned changes. This will involve a lot of discussions on the nitty-gritty details, and some (at least) proof-of-concept implementations for major pieces of our infrastructure including the API, editors, important libraries etc. The main goal of this phase is to lock down all the details and allow everybody in the wider community access to everything that's needed for them to start updating their software.

At the end of this phase we need a final specification for the data model, API, and file formats. We should also offer tools for converting between old and new file formats (as much as this is possible) as part of a migration strategy. As far as this didn't happen in Phase I, we should also implement any tools needed for mappers to help with cleaning up the data before it is moved to the new data model (for instance to fix invalid polygons which the new format might not allow).

Phase II will need about 9 to 12 months depending on funding availability. If more work can be done by paid staff, things will go quicker.

5.3 Phase III: Implementation and Changeover

In Phase III the wider OSM community uses the tools developed in Phase II to adapt their data processing etc. to the new setup. They will use new libraries, a test API and test database etc. to make sure they can work with the new systems.

We have to be prepared to help community members switch over with tutorials, guides, help in online forums, courses, etc. All OSMF core services have to be prepared.

At the end of this phase the changeover to the new system will take place.

Phase III will need about 6 to 12 months.

5.4 Phase IV: Follow-Up

After the change over at the end of Phase III there will be stragglers who only now realize they need to change something. And, as with any new system, there will be issues with this one that need to be resolved. A major effort will also be needed to update documentations, best-practice tutorials, etc. so that possibly confusing references to the old way of things are removed.

Phase IV will need about 3 to 6 months after which we go back to business as usual.

5.5 Manpower Needed

To implement all of this a lot of work is needed, much of which is done by volunteers of the OSM community or, for instance, software developers paid for by their respective companies. But there are some core jobs which need to be filled and paid for (probably by OSMF) to give this effort a chance of succeeding. These are not necessarily full-time jobs (and some of these jobs might be done by the same person) and the time allocated might change over the different phases.

We'll need a **project manager** for the overall coordination of this project, a **technical lead** with a deep knowledge of the OSM data model who leads the technical discussion and draws up the technical documentation and specs together with the working group. We'll need somebody (also with a technical background) for **community relations** who reaches out to the wider OSM community and helps with explanations and documentation and supports developers and other with their changes.

We'll also have to think about helping key projects such as the Rails Port, iD, JOSM, Overpass, etc. with implementing these changes, possibly with paid staff or by giving existing developers stipends, so that changes needed can be implemented in a reasonable amount of time. Large projects such as JOSM might also need (financial) help with project management.

5.6 Servers needed

We'll need some servers for this effort, nothing major though. From phase I onward we want to run software to help the community clean up the data. We also want to have regularly updated OSM data using the new model in the new file formats to allow developers, and later users, to test it. One server at something like 1.500 € a year is enough for this.

Starting from phase II we'll need an additional server of the same class for database system/API development and testing and possibly more systems to help the OSM community developers with their software, if they don't have the resources themselves.

We might need new or upgraded servers starting from phase III for the OSMF core services for the switchover, but it is to soon to tell if this is needed and what exactly. This must be determined together with other EWG/OWG planning during phase II.

6 Software and Systems

The proposed changes will affect almost all software in the OSM ecosystem. Here is a (necessarily incomplete) list of the software and systems affected.

6.1 Core Software

The *Core Software* is most important, without updating it, we can't switch to a new data model. Updating this software has to be part of the plan, funding has to be available for at least some of it.

OSM API and Website "The Rails Port"	Core database, API, and website. Without this we can not do much. Ruby on Rails. https://github.com/openstreetmap/openstreetmap-website
CGImap	Access to core database, API. C++. https://github.com/zerebubuth/openstreetmap-cgimap
Planet Dump NG	Tool for converting an OpenStreetMap database dump into planet files. C++. https://github.com/zerebubuth/planet-dump-ng
OSM Database Tools	Create change files from database changes. C++. https://github.com/openstreetmap/osmdbt
OSMPBF	Library for PBF format. Used by other software. If PBF format will be used in the future, this needs an update. If PBF format is replaced by something new, this won't be needed any more. C++/Java. https://github.com/openstreetmap/OSM-binary
Libosmium	Important C++ library used by a lot of OSM software. C++. https://osmcode.org/libosmium/
Osmium Tool	Widely used general OSM data tool. C++. https://osmcode.org/osmium-tool/
osm2pgsql	Important conversion/database import tool. Used by carto map and others. C++. https://osm2pgsql.org/
taginfo	Create statistics about tag use etc. C++/Ruby/SQL. https://github.com/taginfo
iD	Important editor. Javascript. https://github.com/openstreetmap/iD
JOSM	Important editor. Java. https://josm.openstreetmap.de/
Nominatim	OSM database search engine. PHP/Python/SQL. https://nominatim.org/
Osmosis	Still used often, but not actively maintained. Tied very much to the existing data model. Unclear whether all functions can be taken over by other software. At least populating the API database from an OSM file needs Osmosis. https://github.com/openstreetmap/osmosis

6.2 Major Software

The *Major Software* includes some other projects that are used by many people or that are otherwise important for the OSM ecosystem. We'll need to reach out to the maintainers and help them update their systems.

Overpass, Overpass Turbo	Widely used OSM database query tool. C++, Javascript. https://wiki.openstreetmap.org/wiki/Overpass_API https://overpass-turbo.eu/
OSM Inspector	Widely used QA tool. https://tools.geofabrik.de/osmi/
Osmose	Widely used QA tool. https://osmose.openstreetmap.fr/
Maproulette	Widely used QA tool. Javascript, Scala. https://maproulette.org/
Imposm	Widely used database import tool. https://imposm.org/
GraphHopper	Routing Engine. Java. https://github.com/graphhopper/graphhopper
OSRM	Routing Engine. C++. https://github.com/Project-OSRM/osrm-backend
Valhalla	Routing Engine. C++.
StreetComplete	Android Editor. Kotlin. https://github.com/streetcomplete/StreetComplete
Vespucci	Android Editor. Java. https://vespucci.io/
hdyc	Statistics viewer for contributions of any OSM user. http://hdyc.neis-one.org/

6.3 Other Software

There is a lot of other software out there. For an (incomplete) list see https://github.com/osmlab/awesome-openstreetmap.

7 References

SotM 2018: Talk by Jochen Topf: *Modding the OSM Data Model* at SotM 2018, Milano, Italy. https://2018.stateofthemap.org/2018/T107-Modding_the_OSM_Data_Model/ https://media.jochen-topf.com/media/2018-07-30-talk-sotm2018-data-model-en-slides.pdf

And discussion afterwards:
https://2018.stateofthemap.org/2018/W019-Present_and_Future_of_the_OSM_data_model_from_the_Overpass_API_perspective/

FOSSGIS 2018:
Session lead by Jochen Topf: *Evolution des OpenStreetMap-Datenmodells* at FOSSGIS 2018, Bonn, Germany.

Wiki:
https://wiki.openstreetmap.org/wiki/Area/The_Future_of_Areas