

VTS

This document provides somewhat high-level information on the bulk of upcoming data design changes in Vadstena's output format pertaining to the VTS project. It tries to explain why these changes are needed, describes the underlying notions and concepts and addresses their implementation into data structures both from an analytical (a per usage) and syntehtic (as per conception) viewpoint. This, at least as of time of writing, is not a complete format specification. Whilst key areas are worked out in depth, a lot is left to the the gentle programmer.

VTS

1 Introduction

1.1 Motivation

1.2 Concepts

2. Analysis: Understanding VTS

2.1 Reference Frames

2.2 Positions

2.3 Attributions (credits)

2.4 Bound layers

2.5 Surfaces

2.5.1 Metatiles

2.5.1.1 Metatnodes

2.5.2 Navigation tiles

2.6 Glue

2.7 Free layers

2.9 Client side views

2.10 A complete map configuration (client side)

When the project was first conceived, VTS stood for "Vadstena Tile Set". Today, it's a misnomer: as the project progressed we discovered quickly that the scope of changes would go well beyond the current tile set format. In fact, changes to the tileset itself are rather marginal compared to the more substantial reworking of the other elements of Vadstena data design. The name stuck, so we keep using it in this document in a very broad sense, as an umbrella term for data structures involved in map rendering via the Vadstena rendering engine. A term like "Melown Maps Data Model" might be more appropriate for the general public.

1 Introduction

Vadstena output data format, also known as "webexport format" or "tileset" is a major development achievement, both in design and implementation. The 1.5 years of its active usage, however, have revealed some limitations. To keep up with its ambitious nature, we need to enhance it with new functionality.

1.1 Motivation

With VTS, we aim at:

- promoting Melown Maps to global coverage (as opposed to existing "extended Europe")
- easy integration of raster data from **Web/Pseudo Mercator** based web servers, free or commercial
- easy integration of external vector data, especially **OSM data**
- ability to display the world in its physical, geocentric form (without impeding our ability to display it in current planar, projected form)
- seamless, streamlined and efficient merging procedure
- easy deployment and slick design, which will encourage usage by the open source community
- ability to build GIS applications atop of the map, with multiple surfaces and layers driven by user interaction
- data-driven attribution (copyright notice display)

Additionally, we need to be able to work with existing datasets, which hold almost 2 year of commercial computational history.

Some of these goals, most notably the first two, can be easily achieved with very little development work. In fact, the current webexport format requires only change to a suitable reference system, tile alignment and root tile size to model the world in its entirety. And if the reference system is Web Mercator, the tiles will map one-to-one on any popular map service, making meeting of the second goal a rather trivial task. And since **Geodata layers** have been implemented, OSM support is a mere encoding problem, which does not imply inherent changes to the existing data model. This is an important observation, as it will allow the project ot pick up speed quite quickly.

The rest of the goals, however, will take a bit more effort to achieve. The basic prerequisite is data design which will go beyond the current capabilities.

1.2 Concepts

There are two ways to onlook on VTS: either from the client side (interpreting existing data), or from the server side (composing or creating data). We call the first perspective *analysis* and the second perspective *synthesis*. Both perspectives are addressed in this document, with more attention given to analysis (as we believe it deserves).

Typical analytical concepts in VTS are that of *configuration*, *surface* and *layers*, typical synthetic concepts are that of a *tileset*, *storage*, and *storage view*. Let us take a closer look at these, and others.

Reference Frame

- definition of a fixed **spatial reference** for all data within a map
- defines how the space is split in a tile hierarchy
- provides geometrical surface used for navigation within a map
- defines a spatial reference used for the maps public (user facing) interface

Tileset

- a tiled surface (set of meshes with metadata)
- meshes are textured: usually, but not necessarily
- corresponding to a given reference frame
- possibly taking advantage of external texture layers
- containing credits (copyrights, attributions)

Storage

- an array of tilesets
- sharing the same reference frame
- contains glue: a derived array of tilesets containing tiles originating from different tilesets

Storage view

- is basically a masqueraded storage (as a different storage)
- combines multiple tilesets into a single virtual tileset
- storage view needs physical, local access to storage

Extended configuration

- contains definitions of extra surfaces, layers, roi providers, etc.
- may redefine initial position

Configuration

- a definition of all resources which form a map, complete with metadata
- lists surfaces, layers, roi providers, initial position, etc.
- the umbrella source of information for map browser (mapConfig)
- tileset, storage, and storage view all provide basic configurations
- typical production configuration = storage view + extended config

Surface

- a client side notion of tileset
- multiple types of surfaces could exist (sampled vs. analytical), but for the time being, tilesets are the only supported type

Layer

- originally, 2D information draped over a surface

- some layers hold 3D information but they never aspire to replace geometrical information on a surface
- spatial division (tiling) is always defined by the reference frame

Bound Layer

- bound layer tiles always correspond (in LOD and index) to the active surface
- bound layers do not need metatiles, they share metatile information with the active surface
- currently, a single type exist: raster
- raster layers are external textures
- they may be directly used at tile level
- or they can be user-controlled at map level, or both

Free Layer

- free layers differ from bound layers in that they may display different tiles (different lod or index) than that currently on display by the active surface
- free layers are not completely decoupled from active surface - if they contain 2D data or AGL 3D data, active surface is used to determine their position
- spatial division (tiling) defined by reference frame
- free layers need metatiles, similar to surfaces
- currently, a single type exists: tiled geodata
- geodata contain vector information
- more types of layers will hopefully arrive in the future (traffic, particle systems, clouds, etc.)

ROIs

- ROI stand for *a resource of interest*
- ROIs form the "explore" bar
- ROI providers are servers providing position based ROI access
- a ROI always has a thumbnail, type and title, possibly a tooltip
- multiple ROI types exist: location, external location, panorama and photo
- locations are interesting places on the map (Kokořín, Frýdlant), possibly with extra metadata
- external locations point to different worlds (Mars)
- in the near future, more ROI types will arrive: photo, panorama, etc.

In summarizing these concepts, we get a little ahead of ourselves, hence not all of this might make sense to the gentle reader. Hopefully, things will clarify themselves as you read on.

2. Analysis: Understanding VTS

2.1 Reference Frames

The concept of reference frames is of crucial importance in VTS design. In order to create and use 3D map data, we need answers to questions such as:

1. Which coordinate system (or more precisely spatial reference) are geometries within polygonal meshes and metatiles?
2. When the user navigates the map, what is logic of motion? For example, what does *pan motion* mean, geometrically? When we rotate around an object, what is the axis of rotation?
3. When we report spatial coordinates to the user, how do they relate to the coordinate system used for geometries?
4. There is a tile hierarchy within the map. How is the map split into tiles? Is there a way to tell the physical extents of a tile with given indices on a given LOD?

It is worth noting that the same questions need to be answered in present webexport format, and indeed they are. If we take a look at the current webexport format, we find the following lines in [mapConfig.json](#):

```
{
  "srs" : "+proj=utm +zone=33 +ellps=WGS84 +datum=WGS84",
  "foat" : [ 0, -3700000, 1300000, 8388608 ],
  "alignment" : [ -3700000, 1300000 ],
  "baseTileSize" : 8388608,
}
```

In this case, we get a complete answer to almost all of the above questions. We know that the mesh coordinate system is **UTM 33N** (question 1), we know that the same system is used for tile hierarchy, we know that the hierarchy starts at LOD 0 with the lower left corner of the root tile positioned at (-3,700,000; 1300000) and the size of the root tile is 8,388,608 meters. Since the tiles are split in half along X and Y axes at each LOD (hardcoded), we know the answer to question 4. Pan motion is always defined as motion along the coordinate's system XY plane, and rotation means orbiting around axes perpendicular to this plane (both are hardcoded), so question 2 is out of the way as well. Answer to question 3 is hardcoded, since we always use WGS84 (end EGM96 vertical datum) to communicate to the user, so these few lines give us a complete picture.

What we have just described is the map's reference frame, a complete answer to the four questions above. So why can't we just stick to this simple, two line (the latter two lines are in fact redundant) definition?

Well we might, but with some limitations. First of all, we could not represent the physical, geocentric world. In the geocentric world, the surface is nonplanar, and there is no way a user could move along the surface in the XY plane. We also cannot split the world into tiles along XY geocentric axis. And we would not be able to model other objects, say Mars, with WGS84/EGM96 hardcoded on the UI - both datums are Earth specific and lead to nonsensical results when used for other objects.

So we need a more generic way to define a reference frame. Hence a **reference frame** consists of:

1. *Physical spatial reference system*, employed by meta node bounding boxes, by mesh and free layer geometries
2. *Navigational spatial reference system*, whose XY plane defines tangential movement (pan) and objective rotational movement (orbit) as movement along the plane and rotation around axis perpendicular to the plane, respectively; its Z component is employed by navigation tiles and it is the system used in **positioning**, both to define position and orientation
3. *Public spatial reference system*, which is used when interfacing with the user (when informing user of current position in the map or of altitude or when handling user input)
4. *Spatial division*, which defines physical extents of the model and extents of every tile on every level-of-detail (LOD), either explicitly (for a specific tile) or by setting out a rule by which tiles are split into subtiles.

Several points should be made here.

First, a reference frame is not the same thing as a spatial reference system (SRS), though the two are closely related. A reference frame defines multiple SRSeS, and each of these serves a different purpose. In the simple example we discussed, some of these systems were identical, and others hardcoded and thus did not call for an explicit definition.

Second, the nature of physical SRS defines two basic categories of reference frames. If the physical SRS is projected (such as **UTM 33N** or **Web Mercator**), we deal with a planar 3D map, or a *projected reference frame*. If the physical SRS is geocentric (such as **Geocentric WGS84**), we deal with a physical depiction of the planet (or another object), or a *geocentric reference frame*. If things turn out the way they should, this distinction will be purely observational, since the algorithmic processing is precisely the same in both cases.

Third, spatial division makes use of SRSeS as well. In fact, we are splitting the three

dimensional world by a hierarchy of two dimensional tiles, and we try to do it as uniformly as possible. We could generalize our model to allow for different approaches, but this will do for the time to come.

To get more of the same, let us go back to our example and define a VTS reference frame for it. Here it is:

```
{
  "version" : 1,
  "id": "ppspace",
  "description": "Europe, legacy UTM33N planar reference frame (Seznam.cz 05)",

  "model": {
    "physicalSrs": "utm33n-va",
    "navigationSrs": "utm33n-va",
    "publicSrs": "geographic-wgs84-egm96"
  },

  "division": {
    "extents" : {
      "ll": [-3700000,1300000,-500], "ur": [4688608,9688608,7000] },
    "heightRange" : [-500,7000],
    "nodes": [
      {
        "id": { "lod": 0, "position": [0,0] },
        "srs": "utm33n",
        "extents": {
          "ll": [-3700000,1300000], "ur": [4688608,9688608] },
        "partitioning": "bisection"
      }
    ]
  },

  "parameters" : {
    "metaBinaryOrder" : 5,
  }
}
```

In the example above we reference multiple SRSEs (spatial reference systems). The definitions of the referenced SRSEs need to be provided elsewhere in the map configuration, and indeed they are:

```
{
  "utm33n": {
    "comment": "Projected, UTM 33N",
    "srsDef": "+proj=utm +zone=33 +datum=WGS84 +no_defs",
    "type" : "projected",
  },
  "geographic-wgs84-egm96": {
    "comment": "Geodetic, WGS84 with EGM96 vertical (epsg:4326+5773)",
    "srsDef": "+proj=longlat +datum=WGS84 +geoidgrids=egm96_15.gtx +vunits=m",
    "type" : "geographic",
    "geoidGrid": {
      "extents": {"ll": [-180,-90], "ur": [180,90]},
      "valueRange" : [-107, 85.4],
      "definition": "srs/geographic-wgs84-egm96/geoidgrid.jpg",
      "srsDefEllps": "+proj=longlat +datum=WGS84 +no_defs",
    }
  },
  "utm33n-va": {
    "comment": "Projected, UTM 33N, vertical adjusted",
    "srsDef": "+proj=utm +zone=33 +datum=WGS84 +geoidgrids=egm96_15.gtx +vunits=m",
    "srsModifiers": [ "adjustVertical" ],
  }
}
```

```

    "type" : "projected",
    "geoidGrid": {
      "extents": {"ll": [-2009979, 3000861], "ur": [2999421, 8260731]},
      "valueRange": [-17.6, 67.3],
      "definition": "srs/utm33n-va/geoidgrid.jpg",
      "srsDefEllps" : "+proj=utm +zone=33 +datum=WGS84 +no_defs"
    }
  }
}

```

Definition of geographic-wgs84 should be easily understandable to anyone familiar with the [proj4](#) library. In fact, a PROJ4 string in this case amounts to the entire definition. The type attribute (which can take one of values projected, geographic or cartesian) is in fact redundant, since these characteristics follow directly from the PROJ4 definition.

Definition of geographic-wgs84-egm96 is an example of a compound coordinate system containing a combination of horizontal and vertical datum. It is fully described by a PROJ4 string, yet the applicability of this definition relies on availability of the EGM96 vertical datum grid. This grid is not directly available to the web browser (for good reason, as it is 4 megabytes in size). For usage in analytical contexts, SRSes with orthometric (gravity model based) vertical datums are thus provided with a simplified mechanism defined in the geoidGrid attribute: there is a definition of an ellipsoidal version of the same SRS, a URL to a vertical datum grid, and georeferencing information for that grid. Geoid grids used in this context are compressed and have lower precision than datum grids used in synthesis; this is an acceptable trade-off for their small size and usability in web rendering context (note: grid, not pixel registration is used for georeferencing geoid grid extents for practical reasons).

utm33n-va is the most complex beast. It is a coordinate system with well known definition and a nonstandard modifier, named "adjustVertical", applied to it. This modifier is applicable to conformal SRSes and means that the Z coordinate (which would represent orthometric height without modification) is scaled proportionally to the local scale in XY plane to make the system conformal in all three coordinates. There is no way to represent this modifier in PROJ4, hence the SRS definition needs an extra attribute. (note: in the current implementation, the very same modifier is hardcoded into synthesis. *All* projected physical srses should probably employ this modifier, otherwise things will look wierd in parts of the scene where scale gets distorted).

So putting it altogether, what does the above reference frame definition tell us? It tells us that

- the SRS used in mesh and free layer geometries, and in metatile bounding boxes is UTM33N with vertical adjustment performed on orthometric height
- that panning is carried out in UTM33N xy plane, and rotation around axis perpendicular to this plane, and that navigation heightfields and positions are expressed in the same SRS
- that we talk WGS84 latitudes and longitudes and EGM96 MSL altitudes to the user
- that the working space is a rectangular cuboid occupying 8388608x8388608x7500 meters in physical SRS
- that, with the working space divided into tiles with each tile containing 4 subtiles with levels of detail starting from 0 (hardcoded), that the tile at lod 0 and position (0,0) is defined by coordinates [-3700000,1300000] and [4688608,9688608] at UTM33N, and that it is split by bisection into four equally sized tiles
- that since no other tiles are explicitly described, the bisection mechanism at UTM33N is applicable to each and every tile, thus defining the entire tile hierarchy.

So we arrive at pretty much the same reference frame definition we started from. Our abilities to define reference frames, however, are vastly expanded. To demonstrate this, let us take a look at a different example.

Let us define a couple more SRSes:

```
{
  "geocentric-wgs84": {
    "comment": "Geocentric, WGS84 (epsg:4978)",
    "srsDef": "+proj=geocent +datum=WGS84 +units=m +no_defs",
    "type": "cartesian"
  },
  "geographic-wgs84": {
    "comment": "Geographic WGS84 (epsg:4326)",
    "srsDef": "+proj=longlat +datum=WGS84 +vunits=m +no_defs",
    "type": "geographic"
  },
  "pseudomerc": {
    "comment": "Projected, Web/Pseudo Mercator (epsg:3857)",
    "srsDef": "+proj=merc +a=6378137 +b=6378137 +lat_ts=0.0 +lon_0=0.0 +x_0=0",
    "type": "projected",
    "periodicity": { "type": "X", "period": 40075016.685578 }
  }
}
```

and the following reference frame:

```
{
  "version": 1,
  "id": "webmerc-unprojected",
  "description": "Earth, WGS84 spheroid based on Pseudo/Web Mercator",
  "model": {
    "physicalSrs": "geocentric-wgs84",
    "navigationSrs": "geographic-wgs84",
    "publicSrs": "geographic-wgs84-egm96"
  },
  "division": {
    "extents": {
      "ll": [-7500000, -7500000, -7500000],
      "ur": [7500000, 7500000, 7500000] },
    "heightRange": [-12500, 9000],
    "nodes": [
      {
        "id": { "lod": 0, "position": [0,0] },
        "srs": "pseudomerc",
        "extents": {
          "ll": [-20037508.342789, -20037508.342789],
          "ur": [20037508.342789, 20037508.342789] },
        "partitioning": "bisection"
      }
    ]
  },
  "parameters": {
    "metaBinaryOrder": 5
  }
}
```

What we get here is a geocentric system where all metanode bounding boxes, mesh and free layer geometries are represented in **XYZ cartesian coordinates based on the WGS84 datum** (with the SRS origin at the ellipsoid centre XY plane corresponding to the equator and X axis passing through zero longitude, with navigation controls based on the same ellipsoid

(tangential pan motion and rotation around axis perpendicular to the ellipsoid surface), positioning given by the ellipsoid latitude and longitude, public UI readings based on WGS84/EGM96, spatial division into tiles based on [Pseudo/Web Mercator](#). This is a full blown, physical model of the Earth between roughly 85.05112878°S and 85.05112878°N.

Not only these extents, but the tetranary hierarchy of tiles exactly corresponds to the one used by 2D maps on Here, Bing Maps, Google Maps, OSM and elsewhere, meaning that a tile at given lod/x-index/-yindex combination has exactly the same geographic extents in our reference frame as the Google Maps API tile with the same lod and x and y index combination. This trait means, among other things, that the public map services can play the role of *external bound layers*: we will encounter this later in this document.

The working space is set to a cube centered at WGS84 center, with diameter of 15,000 kilometers - this even allows us to include some orbital phenomenons into the model if we want to.

Note that we could achieve virtually identical results with a slightly different reference frame. For example, our navigation SRS could use a spherical, as opposed to ellipsoidal model of earth. This would make no difference to the user, and all of the above would still apply. Really. The only difference would be that the internal content of navigation tiles would be different, containing spherical height as apposed to ellipsoidal height, and same would apply to positioning. And the axis of orbital motion would not point perpendicular to the (roughly ellipsoidal) earth surface, but straight to the earth centre (or even more precisely, very close to the earth centre).

Helas, our model of earth is missing polar caps. This is merely a limitation of our definition, or more precisely, of the spatial division we employ. It is possible to define a reference frame with polar caps included, and indeed this is the one we will use the for the Melown Maps earth model. This reference frame definition [can be found within our SCM repository](#) (under identifier melown2015). Another interesting example in this sampler is a Mars reference frame, based on IAU datums defined for that planet (mars_mpc). The corresponding SRS definitions [are included](#).

The most important thing to remember about reference frames is that **there exists only one reference frame definition for each map configuration**. Thus all surfaces, glue, bound and free layers, ROIs within the same configuration need to stick to the same reference frame. The same, as we will see, applies to synthesis, or map creation. This is a basic requirement of homogeneity: combining map resources with different frames of reference is not possible unless a prior conversion is performed.

2.2 Positions

Taking a look at the [current mapConfig.json](#), we find the following specification of initial position:

```
{
  "defaultPosition" : [ -1111328.0, 5625376.0, 1703936.0 ],
  "defaultOrientation" : [ 0.0, -90.0, 0.0 ],
}
```

The same positioning scheme is used elsewhere (vadstena web database, persistent links to Melown Maps). It reflects very low-level approach to positioning. The center of perspective is specified by XY coordinates taken from the physical SRS, Z coordinate is altitude above ground level (AGL), and orientation is specified by three [NED based Euler angles](#) (more precisely, it mimics the said angles, since the physical SRS is a projected system, not an NED system in which Euler angles work). The FOV used by the browser camera is hardcoded (90 degrees vertical).

It has been argued that the term "position" as we employ it is rather inappropriate, since it encompasses not only the actual narrow-sense position (the center of perspectivity), but also the orientation of his view and (to an extent) internal camera parameters. The term "view" might indeed be more appropriate, but we use it in a completely different sense in VTS and

we deem usage of "view" in two completely different meanings quite confusing. We stick to "position" for the time being.

In VTS, we shall try to make position specifications

- independent of surface information and thus immediately usable
- orthogonal with respect to the vertical FOV, which will become part of the position
- intuitive, and adaptable to multiple semantic contexts (situations in which positioning is performed).

In VTS, the general position format is called an *objective position*. Presuming we work with the ppspace reference frame defined in previous section, its example definition is

```
["obj",472205.0,5555763.0,"fix",229.0,-10,-57,0,75,80]
```

where

- element 0 is string "subj"
- elements 1-2 are XY components position of the *center of orbit* in navigation SRS
- element 3 is either "fix" or "float"
- element 4 is either Z component of the center of orbit in navigation SRS (if element 3 is "fix") or its AGL (if element 3 is "float")
- elements 5-7 are NED based Euler angles (yaw, pitch, and roll) of the direction of view, measured at the center of orbit
- element 8 is vertical extent of camera view, measured at the center of orbit at physical SRS units
- element 9 is vertical FOV, measured in degrees.

As a special case, the value of element 8 may be 0, indicating that the projection is orthographic.

A slightly less general position format, named *subjective position*, is defined as a similar 10-tuple:

```
["subj",472201.0,5555739.0,"fix",266.4,-10,-57,0,44.7,80]
```

where

- element 0 is string "subj"
- elements 1-2 are XY components of the *center of perspectivity* in navigation SRS
- element 3 is either "fix" or "float"
- elements 4 is either Z component of the center of perspectivity in navigation SRS (if element 3 is "fix") or its AGL (if element 3 is "float")
- elements 5-7 are NED based Euler angles of the direction of view, measured at the center of perspectivity
- element 8 is distance from the center of perspectivity to the center of orbit, and
- element 9 (optional) is vertical FOV, measured in degree

Under presumptions given, the example subjective and objective definitions are identical up to rounding errors. They define a view pointed at the base of the tower of Jenštejn castle, oriented 10 degrees east of north and tilted 57 degrees downwards in such a manner that 75 meters of terrain is observed in vertical direction and the vertical camera FOV is 80 degrees, which translates to a distance of about 44.7 meters.

Alternate definitions of the same position could be given as follows:

```
["obj",472205.0,5555763.0,"float",2.0,-10,-57,0,75,80]  
["subj",472201.0,5555739.0,"float",39.4,253078623059,-10,-57,0,44.7,80]
```

In the two definitions above, we define the position relative to the ground level, in a manner roughly corresponding to the previous two definitions. Its interpretation, however, strongly depends on what "ground level" is: not only that this depends on the surface being displayed,

and even within a single surface, this information is not fixed, as different ground level information may be used at different LODs. Floating positions are thus unstable and should not be used in context which calls for predictable, invariant positioning.

VTs API calls shall be provided for conversions between position formats.

A couple of observations should be made about positional definitions.

Fixed positions, both subjective and objective, meet the criterion of immediate usability (they directly translate to projection matrices in absence of surface information). Their intuitiveness is limited by reliance on navigation altitude (which is typically geodetic height). Floating positions rely on intuitive concept of ground level altitude, which makes them surface dependent and not usable until surface geodetic altitude at given XY coordinates is available. Both positions meet the criterion of FOV orthogonality.

Objective positions are generic in the sense that they can describe all possible projection matrix definitions, including the orthographic projection. Subjective positions lack the ability to express orthographic projections as they are based on the center of perspectivity. Any subjective position may be converted into an objective position, and any objective position with a non-zero FOV may be converted into a subjective position. In a geographic navigation SRS this conversion is non-trivial, since Euler angles are based on local tangential planes and these planes differ in the center of perspective and in the center of orbit - this phenomenon is not manifested in the trivial example above, but it will become profound at larger scale orbits.

For completeness, let us convert the examples provided in this section to the webmerc-unprojected reference frame, as defined in the previous section:

```
[ "obj", 14.6109248, 50.1534351, "fix", 273.8, -10.3, -57, 0, 75, 80 ]  
[ "subj", 14.6108678, 50.1532190, "fix", 311.3, 253078623059, -10.3, -57, 0, 44.7, 80 ]  
[ "obj", 14.6109248, 50.1534351, "float", 2.0, -10.3, -57, 0, 75, 80 ]  
[ "subj", 14.6108678, 50.1532190, "float", 39.5, 253078623059, -10.3, -57, 0, 44.7, 80 ]
```

The difference here is that the webmerc-unprojected reference frame uses a geographic navigation SRS (wgs84-geographic). Though there is no obvious difference in the Euler angles, this is in fact a special case: things would get dramatically different if we move away from the UTM central meridian.

2.3 Attributions (credits)

Current webexport format does in no way address attributions.

The attributions in VTs shall be data driven, meaning that each resource will supply its own copyright information at data level. When browser displays a map, it will assemble all this information into a single attribution notice. All available attributions will be listed in the credits section in map configuration, in the following manner:

```
{  
  "credits" : {  
    "citationtech" : { "id": "1", "notice": "Citationtech" },  
    "usgsnasa" : { "id": "2", "notice": "USGS/NASA", "copyrighted": false },  
    "osm" : { "id": "3", "notice": "OpenStreetMap contributors" },  
    "visionmap" : { "id": "100", "notice": "VisionMap, Ltd", "url": "http://w  
    "interatlas" : { "id": "101", "notice": "INTERATLAS™: used by permission" },  
    "fotomapy" : { "id": "102", "notice": "Fotomapy sp. z o.o." },  
    "microsoft" : { "id": "103", "notice": "Microsoft Corporation" },  
    "here" : { "id": "104", "notice": "HERE, Digital Globe" }  
  }  
}
```

If a resource claims attribution, it simply refers to the numerical id from the dictionary

above. The optional attribute "copyrighted" is true by default, false value indicates that this is an attribution without a copyright claim (which is typical for US government public domain data). Optional attribute "url" allows the attribution in a form of external link.

One important notion of a copyright notice is one of *scope*, or applicability of attribution. For the time being, two scopes shall be used:

"Imagery"

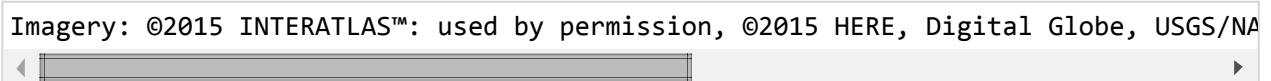
attributions from visible surfaces and bound raster layers

"Map data"

attributions from visible free geodata layers.

Originally, we wanted to include specification of year of first publication (YFP) into attribution, but this turned out to be too much of a hassle. Thus we shall always display current year in the place of YFP. Attribution 3D: MELOWN MAPS is always included.

So suppose we look at 3D map of Paris, based on InterAtlas imagery, with HERE Maps Aerial map draped over ASTER GDEM in the surrounding terrain and OSM based geodata. The complete attribution displayed will go as follows:



Imagery: ©2015 INTERATLAS™: used by permission, ©2015 HERE, Digital Globe, USGS/NA

The more specific (higher LOD) attributions always go first. If there is not enough room to include all attributions, we put hyperlink with text "and others" after the first attribution, pointing to a tooltip containing all attributions.

2.4 Bound layers

As we can recall from [section on concepts](#), bound layers are tiled datasets which may complement an actively displayed surface. The adjective bound (as opposed to free) means that for any tile used from a bound layer, there is the corresponding tile from the active surface, having the same LOD, and indices. For this reason, bound layers do not require any metatile information, as they take it from the active surface (we might be getting a bit ahead of ourselves talking about [metatiles](#), so standby for more information).

Currently, the only type of bound layer in existence is raster. Here is an example of bound layer definition.

```
[
{
  "boundLayers" : {
    "100": {
      "id" : "heresat",
      "type" : "raster",
      "url" : "https://1.aerial.maps.cit.api.here.com/maptile/maptile/newest",
      "tileSize" : [256,256],
      "lodRange" : [0,18],
      "tileRange" : [[0,0],[0,0]],
      "credits" : [ "here" ]
    },
    "110": {
      "id" : "microsoftsat",
      "type" : "raster",
      "tileSize" : [256,256],
      "url" : "http://ecn.t{ms_digit(x,y)}.tiles.virtualearth.net/tiles/a{q",
      "lodRange" : [0,21],
      "tileRange" : [[0,0],[0,0]],
      "credits" : [ "microsoft" ]
    }
  }
}
```

```

    }
  }
]

```

To put this definition into context, we rely on reference frame `webmerc-unprojected` defined in [section on reference frames](#) and use the credit definition defined above in [section on credits](#).

We define two bound layers, one for HERE Maps satellite imagery (or is it BMW these days?), the other satellite imagery from Microsoft (or is it Uber these days?). Each of these layers is assigned a unique numeric id (key to the dictionary) and human readable id (attribute `id`, for manual referencing).

The key part of the bound layer definition is the `url` template. We can see that it points to Here and Microsoft servers, this makes the layer an *external bound layer*. A url template may contain replacement strings embedded in curly braces (`{...}`). Inside these strings, the following simple macros may be used.

Macro	Arguments	Semantics	Comment
<code>x</code>	0	horizontal tile index	
<code>y</code>	0	vertical tile index	
<code>lod</code>	0	LOD level	
<code>sub</code>	0	submesh id (see section on surfaces)	
<code>alt(v1, v2, ... vn)</code>	n	function arguments represent sequence of values from which one value is selected by round-robin pattern	
<code>here_app_id</code>	0	Here Maps App ID, known to browser	
<code>here_app_code</code>	0	Here App code, known to browser	
<code>quad(lod,x,y)</code>	3	Bing Maps quadkey	
<code>ms_digit(x,y)</code>	2	$((y \& 3) \ll 1) + (x \& 1)$	
<code>ppx(lod,x)</code>	2	$\text{hex}(x \ll (28 - \text{lod}), 7)$	used in old mapy.cz URLs
<code>ppy(lod,y)</code>	2	$\text{hex}((1 \ll 28) - ((y + 1) \ll (28 - \text{lod})), 7)$	used in old mapy.cz URLs

NB: `hex(num, size)` function formats number `num` as hexadecimal of at least `size` digits; shorter numbers are zero-padded

It's possible to put simple arithmetic expressions with `+` and `-` operands in a replacement string, these need to be supported by the implementation. Hence `{quad(lod+1,x,y)}` is a legal replacement string, and indeed one we will make use of.

Bound layers definitions contain some extra attributes: `tileSize` defines size of every tile in pixels (thus implying that the size needs to be identical for every tile). `lodRange` defines a range of legal lod values, and `tileRange` defines a range of tile indices within the lowest lod (which in turn defines range of tile indices in every other lod. Finally `credits` contain a reference to a predefined attribution, which is applicable whenever a layer is used.

2.5 Surfaces

As we can recall from the introductory section on concepts, surfaces are a client side notion of tilesets. More precisely, they are

- a geometrical definition of the modeled object's surface,

- with optional textures and/or information on how to map external textures to object's surface
- with information on *terrain*, allowing to map map XY navigation SRS coordinates to their Z compoment.

Our current webexport format is, under our new terminology, a representation of a single surface. It is a surface which is *tiled* (organized in a tile hierarchy) and *sampled* (described as a set of polygonal meshes, as opposed to using analytical and implicit functions). We shall preserve this properties in the new surface format, though analytical surface definitions may be possible in future versions.

A surface definition may look like this:

```
{
  "surfaces" : [
    {
      "id" : "grand",
      "metaUrl" : "/surfaces/grand/{lod}-{x}-{y}.meta",
      "navUrl" : "/surfaces/grand/{lod}-{x}-{y}.nav",
      "meshUrl" : "/surfaces/grand/{lod}-{x}-{y}.bin",
      "textureUrl" : "/surfaces/grand/{lod}-{x}-{y}-{sub}.jpg",
      "lodRange" : [3,22],
      "tileRange" : [[1,1],[6,6]]
    }
  ],
}
```

The elements of the definition are:

Attribute	Description
id	A descriptive id, used to identify surface in client-side view defintions or in API menus.
metaUrl	URL template for metatiles, allows replacement strings .
navUrl	URL template for navigation (terrain) tiles, allows replacement strings .
meshUrl	URL template for meshes, allows replacement strings .
textureUrl	Optional URL template for textures, allows replacement strings .
lodRange	Maximum range of LODs in a surface. This is provided as an optimization in a multi-surface configuration.
tileRange	Maximum range of tile indices at minimum LOD from lodRange. This provided as an optimization in a multi-surface configuration.

As noted above, textureURL template is optional, since a surface does no need to define internal textures. If the textureURL is omitted, the surface meshes should not define internal UVs, since there is no way to use them.

A different approach to texturing might look like this:

```
{
  "surfaces" : [
    {
      "id" : "earth",
      "metaUrl" : "/surfaces/earth/{lod}-{x}-{y}.meta",
      "navUrl" : "/surfaces/grand/{lod}-{x}-{y}.nav",
      "meshUrl" : "/surfaces/earth/{lod}-{x}-{y}.bin",
      "textureLayer" : "microsoftsat",
      "lodRange" : [0,21],
      "tileRange" : [[0,0],[0,0]]
    }
  ]
}
```

```
    ]  
}
```

Here we make use of the following attribute:

Attribute	Description
textureLayer	reference to a bound layer definition used for texturing

The attribute is used to make all tiles textured by an external bound layer. It is one of the three places where a reference to the layer may be used: the other two are mesh files and **view definitions** (higher priority). If textureLayer attribute is defined, all mesh files have to contain external UVs. If textureLayer is omitted, the surface should either define internal textures, or meshes have to contain external UVs to make the surface usable with bound layers defined in view definitions.

As demonstrated in the examples above, a surface is composed of multiple *resources* (metatiles, navtiles, meshes and textures), split across different network URLs. In the remainder of this sections, we will describe the individual resources and their formats.

As we can recall from the section on reference frames, the space the surface tiles occupy is defined either by bisection of a parent tile, or in some exceptional cases, by manual partitioning. In the former case, subtiles are always numbered from the upper left corner and their numbering is zero based (hence if a tile has indices [x,y], its upper left, upper right, lower left and lower right subtiles will have indices [2x,2y],[2x+1,2y],[2x,2y+1],[2x+1,2y+1], respectively). Every tile in every surface is uniquely identified by its id, which is a triple [lod,x,y], where lod is level of detail, and x and y are the tile indices within the given LOD.

The surface definition in map configuration is an *array*, and the order of the surfaces within defines a fixed stacking order. This means that if two or more surfaces are active in the current view, they always show up in the order defined in the surface definition (the last one comes on top, see sections on **glue** and **rendering pipeline** for details).

There is one more trait of surfaces worth mentioning: their similarity to **free layers**. Similarly to surfaces, free layers contain 3D geometries described by metatiles. The two concepts are closely related to the point where we considered their representation by a single data entity. Tempting as this abstraction was, we shunned it for the time being, though both the server and client side implementation will no doubt make heavy use of their similarities.

2.5.1 Metatiles

Metatiles contain horizontally aggregated **metanodes**.

2.5.1.1 Metatnodes

Metanodes contain information about visibility, credits, ...

2.5.2 Navigation tiles

A navigation tile is actually a heightfield, providing a mapping (presumably related to the tile geometry) from horizontal components of a coordinate system to vertical component of a coordinate system. It serves two purposes:

1. **pan motion navigation**, or the use case which gave the navtiles their slightly misleading name, and
2. **heightcoding**, or turning 2D layer information into 3D using a surface geometry, this operation is of little analytical importance but of huge significance in synthesis.

Both of these purposes are unique to surfaces, thus there are no navtiles in free layers.

To define the navtile mapping mentioned above more precisely, it is a mapping *from the local spatial division SRS XY coordinates to the navigation SRS Z coordinate*. The local spatial division SRS is defined by the reference frames ["division"]["nodes"] value applicable to the

node in question. Though it might seem confusing to combine an XY domain from one SRS with a Z range from a different SRS, it is actually the simplest scheme we could figure out. The range is dictated by the purpose of the value and the domain by the way used to access it.

2.6 Glue

Glue is a whole new concept in VTS, specific to multi-surface configurations. As we may recall from the introductory section, a glue is a derived tileset containing tiles composed from different tilesets: in this section, we shall give this somewhat vague synthetic definition a more precise meaning.

First of all, a glue is a *surface*, meaning that it is comprised of the very same resources (metatiles, navigation tiles, meshes and textures) as a **regular surface**. These resources have the very same format and semantics, making glue capable of completely independent rendering. Glue, however, differs from a regular surface in its origin, definition, and most importantly, usage.

So where does glue come from? It is a derived surface created by a specific combination of surfaces. Within any combination, there is a highest order surface (surfaces have a fixed order, as you no doubt remember from the preceding section). Let's call the highest order (topmost) surface *alpha* surface. The glue comes to existence when one of the following conditions are met for any given tile:

1. the tile needs all surfaces in the combination to define its geometry,
2. the tile is fully derived (created from ancestor tiles in tile hierarchy for all surfaces in question) and needs all surfaces except alpha to define its geometry.
3. the tile is fully derived and only alpha is needed to define its geometry.

Only tiles meeting one of these conditions form part of the glue, though metanodes exist all the way to the root tile, as for any other surface.

The situations described by one of the three conditions above is well known to Vastena operators and developers: they occur in *tileset merging* whenever tile geometries need to be physically merged to form the resultant tile. Actually, glue does just that in synthesis, with the important difference that no changes are inflicted on the underlying surfaces: instead, glue is created as a completely new entity.

Let us consider the following surface definition:

```
{
  "surfaces": [
    {
      "id": "ev1",
      "metaUrl": "/surfaces/ev1/{lod}-{x}-{y}.meta",
      "navUrl": "/surfaces/ev1/{lod}-{x}-{y}.nav",
      "meshUrl": "/surfaces/ev1/{lod}-{x}-{y}.bin",
      "textureUrl": "/surfaces/ev1/{lod}-{x}-{y}-{sub}.jpg",
      "lodRange": [7,18],
      "tileRange": [[60,61],[68,65]]
    },
    {
      "id": "krnov",
      "metaUrl": "/surfaces/krnov/{lod}-{x}-{y}.meta",
      "navUrl": "/surfaces/krnov/{lod}-{x}-{y}.nav",
      "meshUrl": "/surfaces/krnov/{lod}-{x}-{y}.bin",
      "textureUrl": "/surfaces/krnov/{lod}-{x}-{y}-{sub}.jpg",
      "lodRange": [13,18],
      "tileRange": [[4286,4036],[4293,4042]]
    }
  ]
}
```

And accompanying glue definition:

```
{
  "glue": [
    {
      "id": ["ev1", "krnov"],
      "metaUrl": "/glue/ev1-krnov/{lod}-{x}-{y}.meta",
      "navUrl": "/glue/ev1-krnov/{lod}-{x}-{y}.nav",
      "meshUrl": "/glue/ev1-krnov/{lod}-{x}-{y}.bin",
      "textureUrl": "/glue/ev1-krnov/{lod}-{x}-{y}-{sub}.jpg",
      "lodRange": [13, 18],
      "tileRange": [[4286, 4036], [4293, 4042]]
    }
  ]
}
```

This is actually a real-life example, with the city of Krnov sitting atop the orthophoto map draped over a DEM of Czech Republic. Since Krnov contains tiles with partial coverage, a glue has been generated for these partially covered tiles, merging geometry (and texture) from both surfaces. Glue definition differs little from a surface definition, with the only subtle difference in the value of the `id` attribute: the value of this attribute is a combination of regular surfaces, which identifies the particular glue.

Note that there is no glue with `id` of `["krnov", "ev1"]`. This is because surfaces have a fixed stacking order and there is no way to move `ev1` atop of `krnov` (actually, there would likely be no glue even if this was possible, since `ev1` completely covers `krnov` area). Hence a glue may exist for each combination, but not for each permutation of regular surfaces; fixed stacking order is a deliberate limitation in VTS.

So what is this good for? When an active view contains both the `ev1` and `krnov` surfaces, Vadtana renderer needs to render resources from both surfaces. For each tile in the reference frame's spatial division, only one surface may provide the resources. If a tile exists in both surfaces, it seems natural to prefer top (`krnov`) to bottom (`ev1`). But if the top surface contains tiles with incomplete coverage, we may not use them, or we will end up with hole. Additionally, the tile trees defining both surfaces may not simply add to a valid surface definition (union of the trees may be disjoint, fail the ternary condition, provide false geometry extents, etc. This is where glue comes into play. Whenever a tile in glue exists, we prefer it over *both* surfaces, hence the final stacking order is `"ev1"` (bottom), `"krnov"` (middle) and `["ev1", "krnov"]` (top).

Things get more complex if more than two active surfaces are involved. For a detailed description please refer to the section on [rendering pipeline](#).

2.7 Free layers

types: geodata, tiled mesh

2.9 Client side views

As obvious from the contents of previous few sections, many a resource may exist in a typical map configuration. A *client side view* defines a set of resources visible at any given moment to the user. An example, in which we follow up on another example provided in [section on glue](#) above may look as follows:

```
{
  "view": {
    "surfaces": {"ev1": [], "krnov": []}, "freeLayers": []
  }
}
```

The above is an example of an initial view definition, forming part of a standard [map](#)

configuration. It tells us that the initial map displayed to the user shall sport two surfaces and that no extra bound layers should be rendered on any of these surfaces - hence the empty value arrays in the surfaces dictionary. The view does not define their stacking order, which is fixed and given by their order in map configuration.

The initial view should always be provided as a good practice, though a *default initial view* exists, and incidentally, it is the same as above: all surfaces and no extra layers. Note that no extra bound layers in the definition above does not necessarily mean that bound layers will not play any part, they may be used in **surface definitions**, or at **meshes themselves**. In fact, a raster bound layer in a view definition serves as a top level overlay, overriding texturing information provided for the surface at another places.

A map configuration may define *named views*. This, as the name implies, is a set of views with descriptive ids and descriptions. Named views may be defined as an API convenience, an application programmer may display a named views menu through a simple API function (namedViewsMenu), thus creating a declarative way to allow the user to switch between different resources at a higher level.

Here is an example of a named view definition, building up on the second, webmerc reference frame:

```
{
  "namedViews": {
    "HereSatEarth": {
      "description": "Here Satellite overlay",
      "surfaces": {"earth": ["HereSat"] },
      "freelayers": []
    },
    "MicrosoftSatEarth": {
      "description": "Microsoft Satellite overlay",
      "surfaces": {"earth": ["MicrosoftSat"]},
      "freelayers": []
    },
    "Mishmash": {
      "description": "Combined overlay, for testing purposes",
      "surfaces": {
        "earth": [
          {"id": "HereSat", "alpha": 1.0},
          {"id": "MicrosoftSat", "alpha": 0.5}
        ]
      },
      "freelayers": []
    }
  }
}
```

The two named views above actually make sense, providing a 3D earth model based on two popular web services. The third one is provided as an example of *overlays with alpha channels*. Semitransparent Bing satellite imagery is overlayed over Here satellite imagery (it's probably not legal to do so for license reasons, so do not use this as a live example).

2.10 A complete map configuration (client side)

A map configuration integrates definitions of all resources forming a map: surfaces, bound and free layers and roi providers. It also defines a common reference frame, initial position and named views. From analytical (client side) standpoint, a map configuration is *always a single file*. Things get very different on server side, but this is not our concern here.

Let's take a look at a very simple map configuration, which, in substance, corresponds to the contents of Melown Maps website at the time of writing (Aug 2015):

```
{
  "srses" : {
```

```

    "utm33n": {
      "comment": "Projected, UTM 33N",
      "srsDef": "+proj=utm +zone=33 +datum=WGS84 +vunits=m +no_defs",
      "type": "projected"
    },
    "utm33n-va": {
      "comment": "Projected, UTM 33N, orthometric, vertical adjusted",
      "srsDef": "+proj=utm +zone=33 +datum=WGS84 +geoidgrids=egm96_15.gtx +",
      "srsModifiers": [ "adjustVertical" ],
      "type": "projected",
      "geoidGrid": {
        "extents": { "ll": [-2009979, 3000861], "ur": [2999421, 8260731] },
        "valueRange": [-17.6, 67.3],
        "definition": "srs/utm33n-va/geoidgrid.jpg",
        "srsDefEllps": "+proj=utm +zone=33 +datum=WGS84 +no_defs"
      }
    },
    "geographic-wgs84": {
      "comment": "Geographic, WGS84 (epsg:4326)",
      "srsDef": "+proj=longlat +datum=WGS84 +no_defs",
      "type": "geographic"
    },
    "geographic-wgs84-egm96": {
      "comment": "Geodetic, WGS84 with EGM96 vertical (epsg:4326+5773)",
      "srsDef": "+proj=longlat +datum=WGS84 +geoidgrids=egm96_15.gtx +vunits:",
      "type": "geographic",
      "geoidGrid": {
        "extents": { "ll": [-180,-90], "ur": [180,90] },
        "valueRange": [-107, 85.4]
        "definition": "srs/geographic-wgs84-egm96/geoidgrid.jpg",
        "srsDefEllps": "+proj=longlat +datum=WGS84 +no_defs",
      }
    }
  },
  "referenceFrame": {
    "id": "ppspace",
    "description": "Europe, legacy UTM33N planar reference frame (Seznam.cz 0)",
    "model": {
      "physicalSrs": "utm33n-va",
      "navigationSrs": "utm33n-va",
      "publicSrs": "geographic-wgs84-egm96"
    },
    "division": {
      "extents": { "ll": [-3700000,1300000,-500], "ur": [4688608,9688608,7000] },
      "heightRange": [-500,7000],
      "nodes": [
        {
          "id": { "lod": 0, "position": [0,0] },
          "srs": "utm33n",
          "extents": { "ll": [-3700000,1300000], "ur": [4688608,9688608] },
          "partitioning": "bisection",
        }
      ]
    },
    "params": {
      "metaBinaryOrder": 5,
    }
  },

```

```

"credits": {
  "1": { "id": "citationtech", "notice": "Citationtech" },
  "2": { "id": "usgs", "notice": "USGS/NASA", "copyrighted": false },
  "100": { "id": "visionmap", "notice": "VisionMap, Ltd", "url": "http://www",
  "101": { "id": "interatlas", "notice": "INTERATLAS™, used by permission."
  "102": { "id": "fotomapy", "notice": "Fotomapy sp. z o.o." },
  "103": { "id": "topgis", "notice": "TopGis, s.r.o." }
},

"surfaces" : [
  {
    "id": "grand",
    "metaUrl": "/surfaces/grand/{lod}-{x}-{y}.meta",
    "navUrl": "/surfaces/grand/{lod}-{x}-{y}.nav",
    "meshUrl": "/surfaces/grand/{lod}-{x}-{y}.bin",
    "textureUrl": "/surfaces/grand/{lod}-{x}-{y}.jpg",
    "lodRange": [3,22],
    "tileRange": [[1,1],[6,6]]
  }
],

"glue": [],

"boundLayers": [],
"freeLayers": [],
"rois": [],
"namedViews": [],

"view": {
  "surfaces": ["grand": []], "freeLayers": []
},

"position": ["obj",472205.0,5555763.0,"fix",229.0,-10,-57,0,75,80],

"version": 4,

"params" : {}
}

```

In the example above, we explore the definitions introduced throughout this chapter. There is a single surface, hence no need for glue, no bound or free layers, and no ROI providers. The initial view is in fact the only view that makes sense, with the only existing surface visible.

Complete map configuration will reside at the traditional path, <map-url>/mapConfig.json.