Open Geospatial Consortium

Submission Date: 2017-02-24

Approval Date: 2017-08-08

Publication Date: 2017-09-05

External identifier of this OGC® document: http://www.opengis.net/doc/CS/i3s/1.0

Internal reference number of this OGC® document: 17-014r5

Version: 1.0

Category: OGC® Community Standard

Editor: Carl Reed, Tamrat Belayneh

OGC Indexed 3d Scene Layer (I3S) and Scene Layer Package Format Specification

Document type:	OGC® Community Standard
Document subtype:	
Document stage:	Approved
Document language:	English

Esri (Environmental Systems Research Institute, Inc.)

The companies listed above have granted the Open Geospatial Consortium (OGC) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version under a Creative Commons ShareAlike (CC BY-SA) license (see below).

License Agreement

The standard is licensed under the Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)[1]. You can implement this standard in services, clients or processing tools without restrictions.

You are directed to the License for specific details..

This is a human-readable summary of (and not a substitute for) the license.

You are free to:

- Share — copy and redistribute the material in any medium or format
- Adapt — remix, transform, and build upon the material for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms:

- **Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
- **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
- **No additional restrictions** — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

Notices:

- You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation.
- No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material.

[1] https://creativecommons.org/licenses/by-sa/4.0/

<Skip Directly to Standard!>

Contents

FIGURES

TABLES

i. Abstract

A single I3S data set, referred to as a Scene Layer, is a container for arbitrarily large amounts of heterogeneously distributed 3D geographic data. Scene Layers are designed to be used in mobile, desktop, and server-based workflows and can be accessed over the web or as local files.

The delivery format and persistence model for Scene Layers, referred to as Indexed 3d Scene Layer (I3S) and Scene Layer Package (SLPK) respectively, are specified in detail in this OGC Community Standard. Both formats are encoded using JSON and binary ArrayBuffers (ECMAScript 2015). I3S is designed to be cloud, web and mobile friendly. I3S is based on JSON, REST and modern web standards and is easy to handle, efficiently parse and render by Web and Mobile Clients. I3S is designed to stream large 3d datasets and is designed for performance and scalability. I3S is designed to support 3D geospatial content and supports the requisite coordinate reference systems and height models in conjunction with a rich set of layer types.

The open community GitHub version of this standard is here: https://github.com/Esri/i3s-spec[2] .

ii. Source of the content for this OGC document

The majority of the content in this OGC document is a direct copy of the content contained at https://github.com/Esri/i3s-spec . No normative changes have been made to the content. This OGC document does contain content not contained at https://github.com/Esri/i3s-spec. Specifically, while derived from content on the https://github.com/Esri/i3s-spec repository, the Abstract, Keywords, Preface, Submitting Organizations, Endorsers, Terms and Definitions, and References sections in this document are not found on the https://github.com/Esri/i3s-spec website. However, there

[2] This OGC Community Standard is based on I3S version 1.6 of the Esri openly available specification. Specifically, navigate to https://github.com/Esri/i3s-spec/blob/master/README.md

is a plan to incorporate the Terms and Definitions and References sections into the community GitHub repository.

Note: Some elements (Lines, Polygons, and PointClouds) contained in https://github.com/Esri/i3s-spec have been removed from the OGC document because they are currently in Beta and not broadly implemented outside the Esri community. These elements are identified as future work in this OGC document.

iii. Validity of content

The Submission Team has reviewed and certified that the "snapshot" content in this Community Standard is true and accurate.

iv. Keywords

The following are keywords to be used by search engines and document catalogues.
ogcdoc, OGC document, i3s, 3d, visualization, scene, scene layer, slpk

v. Preface

I3S originated from investigations into technologies for rapidly streaming and distributing large volumes of 3D content across enterprise systems that may consist of server components, cloud hosted components, and a variety of client software from desktop to web and mobile applications.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

vi. Submitting organizations

The following organizations submitted this Document to the Open Geospatial Consortium (OGC):

Esri, Inc.

vii. Supporting Organizations

All questions regarding this submission should be directed to the editor or the submitters:

Name	Affiliation
Keith Ryden	Esri

Allan W. Shearer	UT Austin
Volker Coors	Hochschule für Technik Stuttgart
David Graham	CAE
Andreas Wytzisk	52North
Carl Reed	Carl Reed & Associates
Gordon Plunkett	Esri Canada
Vijay Kumar	Esri India Technologies
Clemens Portele	interactive instruments GmbH
Brian Nicholls	AAM Pty Ltd
Bomi Lee	Korea Land & Geospatial InformatiX Corporation
Isaac Zaworski	Vricon, Inc
Thorsten Rietz	Wetransform GmbH
Dale Lutz	Safe Software

viii. Future Work

The I3S community anticipates that revisions to this Community Standard will be required to prescribe content appropriate to meet new use cases. These use cases may arise from either (or both) the external user and developer community or from OGC review and comments. Further, future revisions will be driven by any submitted change requests that document community uses cases and requirements.

Currently, the following layer types are planned for future inclusion in the I3S standard (future work):

- Line Features (e.g. from a GIS data repository)
- Polygon Features (e.g. from a GIS data repository)
- Point Clouds (e.g. from LiDAR)

1. Introduction

A single I3S data set, referred to as a Scene Layer, is a container for arbitrarily large amounts of heterogeneously distributed 3D geographic data. A Scene Layer is characterized by a combination of layer type and profile to fully describe the behavior of the layer and the manner in which it is realized within the standard.

The I3S format is declarative and extendable and can be used to represent different types of 3D data. The following layer types have been specified and the standard validated via implementation and production deployments:

- 3D Objects (e.g. Building Exteriors from geospatial data and 3D models)
- Integrated Meshes (e.g. a mesh surface with high resolution imagery textures representing the skin of the Earth, typically created from satellite, aerial or drone imagery)
- Point Features (such as geolocated Hospitals or Schools, trees, street furniture, and signs)

The Indexed 3d Scene Layer (I3S) and Scene Layer Package (*.slpk) are open formats and not dependent on any vendor specific solution, technology, or products[3]. The specification for accessing I3S resources as Scene Service REST (Annex E) endpoints is also described in this standard as open formats.

2. Conformance

Not Applicable.

3. References

Normative

OGC SF [99-036/ISO 19125]: Geographic information - Simple feature access - Part 1: Common architecture. 2005. http://portal.opengeospatial.org/files/?artifact_id=13227. May17, 2017.

OGC WKT CRS [12-063r5/ISO 19162:2015]: Geographic information — Well known text representation of coordinate reference systems. 2015 http://portal.opengeospatial.org/files/?artifact_id=4700 (May 15, 2017)

[3] The specification for accessing I3S resources as Scene Service REST endpoints are described in Annex E (informative).

Informative

"Octree". *https://en.wikipedia.org/wiki/Octree*. Not Published (N.P.), 2016. Web. 20 Oct. 2016.
"Quadtree". *https://en.wikipedia.org/wiki/Quadtree*. N.P. 2017. Web. 20 Jan. 2017
"R-Trees". *https://en.wikipedia.org/wiki/R-tree*. N.P. 2017. Web. 20 Jan. 2017

4. Terms and Definitions

For the purposes of this document, the following additional terms and definitions apply.

4.1 3D Model[4]

Three-dimensional (3D) models represent a physical body using a collection of points in 3D space, connected by various geometric entities such as triangles, lines, curved surfaces, etc.

4.2 Array Buffers

In JavaScript, the `ArrayBuffer` object is used to represent a generic, fixed-length raw binary data buffer.

4.3 Face

In solid geometry, a face is a flat (planar) surface that forms part of the boundary of a solid object; a three-dimensional solid bounded exclusively by flat faces is a polyhedron.

4.4 faceRanges

Indicates the range of triangles associated with a particular object (feature).

4.5 Gravity-related height

Height dependent on the Earth's gravity field. NOTE: This refers to in particular orthometric height or normal height, which are both approximations of the distance of a point above the mean sea level. (ISO 19111)

4.6 Height

Distance of a point from a chosen reference surface measured upward along a line perpendicular to that surface. NOTE: A height below the reference surface will have a negative value.

4.7 Integrated Mesh

An Integrated Mesh is a type of I3S layer that belongs to the mesh-pyramids profile. An Integrated Mesh layer type is typically used to represent and visualize geographic data captured as '3D Image' representing the landscape in a seamless, highly scalable, textured mesh. Such '3D Image' can integrate within its content a multitude of landscape elements including terrain surface, ground imagery, vegetation, man-made objects and

[4] https://en.wikipedia.org/wiki/3D_modeling (February 7, 2017)

structures, and water surfaces. This type of data is typically produced by automated extraction solutions operating on input data from satellite, aerial and/or drone imagery.

4.8 Level of Detail (LoD)

Using different LoDs involves decreasing the complexity of a 3D model representation as it moves away from the viewer or according to other metrics such as object importance, viewpoint-relative speed or position. There are numerous approaches to defining LoDs. In GIS, LoDs typically refer to maps defined at given scales and resolutions. Typically higher levels of detail provide greater fidelity. A number of OGC standards define approaches to LoD.

4.9 Minimum Bounding Sphere[5] (MBS, mbs)

In mathematics, given a non-empty set of objects of finite extension in *n*-dimensional space, for example a set of points, a bounding sphere, enclosing sphere or enclosing ball for that set is an *n*-dimensional solid sphere containing all of these objects.

4.10 Normal or Normals[6]

The normal vector, often simply called the "normal," to a surface is a vector which is perpendicular to the surface at a given point. When normals are considered on closed surfaces, the inward-pointing normal (pointing towards the interior of the surface) and outward-pointing normal are usually distinguished.

4.11 Oriented Bounding Box (OBB)[7]

In geometry, the minimum or smallest bounding or enclosing box for a point set (S) in N dimensions is the box with the smallest measure (area, volume, or hyper-volume in higher dimensions) within which all the points lie. In many applications the bounding box is aligned with the axes of the coordinate reference system and is known as an axis-aligned bounding box (AABB). To distinguish the general case from an AABB, an arbitrary bounding box is called an oriented bounding box (OBB) when an object's local coordinate reference system is used.

4.12 Profile

In I3S, specific implementation instances for specific layer definitions (point, mesh, etc)

---

[5] https://en.wikipedia.org/wiki/Bounding_sphere (February 12, 2017)
[6] http://mathworld.wolfram.com/NormalVector.html (March 3, 2017)
[7] An Exact Algorithm for Finding Minimum Oriented Bounding Boxes. http://clb.demon.fi/minobb/minobb.html (June 1, 2015)

4.13 S3TC[8]

S3TC is a technique for compressing images for use as textures. Standard image compression techniques like JPEG and PNG can achieve greater compression ratios than S3TC. However, S3TC is designed to be implemented in high-performance hardware. JPEG and PNG decompress images all-at-once, while S3TC allows specific sections of the image to be decompressed independently.

4.14 Shader

A small program or set of algorithms that determines how 3-D surface properties of objects are rendered, and how light interacts with the object within a 3-D computer program.

4.15 Texture

In 3D graphics, the digital representation of the surface of an object. In addition to two-dimensional qualities, such as color and brightness, a texture is also encoded with three-dimensional properties, such as how transparent and reflective the object is. Once a texture has been defined, it can be wrapped around any 3-dimensional object. This is called texture mapping.

4.16 Texture Atlas[9]

A large image containing a collection, or "atlas", of sub-images, each of which is a texture map for some part of a 2D or 3D model.

4.17 Texture Mapping[10]

Texture mapping is a method for defining high frequency detail, surface texture, or color information on a computer-generated graphic or 3D model.

4.18 Texture Maps

A texture map is an image applied (mapped) to the surface of a shape or polygon. This may be a bitmap image or a procedural texture. They may be stored in common image file formats, referenced by 3d model formats or material definitions, and assembled into resource bundles.

---

[8] https://www.khronos.org/opengl/wiki/S3_Texture_Compression (February 7, 2017). Please note that S3TC is patented technology and is usage is subject to license restrictions. The patent expires October 7, 2017.

[9] https://en.wikipedia.org/wiki/Texture_atlas (February 19, 2017)

[10] https://en.wikipedia.org/wiki/Texture_mapping (February 19, 2017)

4.19 UV Coordinate[11]

UV coordinates are 2D coordinates that are mapped onto a 3D model. UV coordinates are a texture's x and y coordinates and always range from 0 to 1. Let's take for example a 800×600 image. When we use a UV coordinate with u=0.5 and v=0.5 then the pixel at x=400 and y=300 is targeted.

4.20 UV Mapping[12] (aka UV Unwrapping)

UV mapping is the 3D modeling process of projecting a 2D image to a 3D model's surface for texture mapping.

4.21 Vertex[13]

In computer graphics, a vertex is not only associated with three spatial coordinates but also with other graphical information necessary to render the object correctly, such as colors, reflectance properties, textures, and surface normals. These properties are used in rendering by a vertex shader, part of the vertex pipeline.

5. Conventions

No conventions are specified in this document.

6. Introduction to I3S and SLPK

This section provides background information on the design principals and background for I3S

6.1 I3S Design Principles

The Indexed 3d Scene layer (I3S) format and the corresponding Scene Layer Package format (*.slpk) are specified to fulfill this set of design principles:

1. **User Experience first:** Support a positive user experience - high interactivity, fast display, support rendering of visually relevant features first;
2. **Scalability:** Support very large scene layers, with global extent and large amounts of features - as well as the ability to handle highly detailed features;
3. **Reusability:** Be usable both as a service delivery format as well as a storage/exchange format;
4. **Level of Detail:** Have intrinsic support for representing level of detail;
5. **Distribution:** Allow efficient distribution of very large data sets;
6. **User-controllable symbology:** Support client-side symbology/styling and its efficient rendering;
7. **Extensibility:** Be extensible to support new layer and geometry types as well as new platforms;

[11] http://www.rozengain.com/blog/2007/08/26/uv-coordinate-basics/ (February 19, 2017)
[12] https://en.wikipedia.org/wiki/UV_mapping (February 9, 2017)
[13] https://en.wikipedia.org/wiki/Vertex_(geometry)#Vertices_in_computer_graphics (February 9, 2017)

8. **Web Friendliness:** Easy to handle and parse by web clients by using JSON and current web standards;
9. **Compatibility:** Have a single structure that is usable across a modern platform spanning web, mobile and desktop clients and cloud and on-premises servers;
10. **Declarative:** limit how much specific knowledge is needed by clients for format support;
11. **Follow REST/JSON API best practices:** "Hypertext as the Engine of Application State" - make all resources navigable using hrefs from relevant other resources.

6.2 I3S – Overview

I3S originated from investigations into technologies for rapidly streaming and distributing large volumes of 3D content across enterprise systems that may consist of server components, cloud hosted components, and a variety of client software from desktop to web and mobile applications. A single I3S data set, referred to as a Scene Layer, is a container for arbitrarily large amounts of heterogeneously distributed 3D geographic data. I3S Scene Layers are designed to provide clients access to data. Clients have the ability to then visualize the data for the layer independently according to their needs. Data here refers to vertex geometry, texture as well as any associated attributes. An I3S Layer is characterized by a combination of layer type and profile that fully describes the behavior of the layer and the manner in which it is realized within the specification.

The requirements specified below apply to the following layer types defined in this standard:

- 3D Objects (e.g., Building Exteriors from geospatial data and 3D models)
- Integrated Mesh (e.g., an integrated surface representing the skin of the earth including vegetation, buildings and roads from satellite, aerial or drone imagery via dense matching photogrammetry)
- Points (e.g. hospitals or Schools, trees, street furniture, signs, etc. from GIS data)

Layers are described using two properties: type and profile. The type of a layer describes the type of geospatial data stored within it drawing from terms including 3D Objects and Points. The profile for a layer includes additional detail on the specific I3S implementation for the layer that is exposed to clients. Each layer has a canonical profile, but in certain cases multiple layers that represent semantically different types of information can make use of the same underlying profile. In other cases the same layer type can support multiple profiles optimized for different use cases.

The following table shows the layer types and profiles. For each row the table indicates if the layer type represents features (geographic entities) with identity (as opposed to a geospatial field described by a mesh or cloud of geometry elements) and if the specific profile for the layer supports storage of attributes (either feature attributes or attributes of individual geometry elements, depending on the type of the layer).

Layer Type *(example)*	Profile	Annex	Features with Identity	Attributes
3D Object	mesh-pyramids	Annex G	Yes	Yes
Integrated Mesh	mesh-pyramids	Annex G	No	Triangle Attributes (planned)
Point	points	Annex F[14]	Yes	Yes

Table 1: 3D Layer Types Supported in I3S

7. I3S Specification

This section contains the normative clauses and requirements for implementing I3S.

7.1 Coordinate Reference Systems (CRS)[15]

7.1.1 A note on OGC Standards for CRS and Well Known Text

This document refers to two OGC standards for describing a CRS as Well Known Text. The two standards are referred to as WKT1 and WKT2

- WKT1: Refers to Well Known Text (WKT) for expressing a CRS as originally defined in clause 6.4 in OGC Simple Features [99-036/ISO 19125][16]. This original definition was extended in OGC Coordinate Transformation Service [01-009];
- WKT2: Refers to WKT as defined in OGC WKT CRS/ISO 19162:2015 Geographic information -- Well-known text representation of coordinate reference systems [12-063r5][17]. From the document, "This Standard provides an updated version of WKT representation of coordinate reference systems that follows the provisions of ISO 19111:2007 and ISO 19111-2:2009. It extends the earlier WKT to allow for the description of coordinate operations."

OGC 12-063r5[18] makes several references to backward compatibility. "Backward compatibility means that an implementation of the text strings in this International Standard would be able to read CRS WKT strings conforming to the old (ISO 19125-1:2004) syntax. It does not mean that a parser of a string compliant to ISO 19125-1:2004

[14] NOTE: The JSON/HTML examples for these profiles are posted in the OGC repository: http://schemas.opengis.net/i3s/1.0/

[15] This document refers to two OGC standards for describing a CRS as Well Known Text: 1.) WKT1: WKT as defined in OGC Coordinate Transformation Service [01-009], 2.) WKT2: WKT as defined in OGC Geographic Information – Well known text representation of coordinate reference systems [12-063r5]

[16] http://portal.opengeospatial.org/files/?artifact_id=13227

[17] http://docs.opengeospatial.org/is/12-063r5/12-063r5.html

[18] The text in this paragraph is extracted verbatim from 12-063r5.

could read WKT strings written in conformance with this International Standard. It also does not require an implementation of the text strings in this International Standard to be able to output an object according to the old syntax. Annex B.8 gives guidance on determining the version of a CRS WKT string. A mapping of older syntaxes to this International Standard is given in Annex C."

Please note that in an I3S implementation the CRS MAY be represented using either WKT1 or WKT2. While WKT1 has been in use for many years, WKT1 has been superseded by WKT2. Although implementations of OGC standards using WKT2 are not yet widely available the guidance from the OGC/ISO community is to implement WKT2. Important Note: WKT1 does not support explicit definition of axis order.

Therefore, ***I3S implementers need to note for their implementations if they support WKT1 only or both (as WKT2 requires continued support of WKT1).***

7.1.2 CRS use and requirements in I3S

Indexed 3D Scene Layers have to fulfill a number of requirements when it comes to the selection of Coordinate Reference Systems (CRS) to use:

- Minimize the need for re-projection on the client side
- Support data sets with global extent
- Render easily in coordinate reference systems for projected[19] CRSs as well as coordinate reference systems for geographic[20] CRSs
- Support local data with very high positional accuracy
- Support global data sets with high positional accuracy

These use cases lead to the following implementation requirements.

1. The location of all index-related data structures such as node bounding spheres *SHALL* be specified using a single, global geographic WGS 84 CRS. Coordinate bounds for such structures *SHALL* be in the range (-180.0000, -90.0000, 180.0000, 90.0000). Height and node minimum bounding sphere (MBS) radius *SHALL* be specified in meters. Allowed CRS specified using an EPSG code include: EPSG:4326[21]
2. All vertex positions *SHALL* be specified using a geodetic CRS (including Cartesian coordinate reference systems), where x,y,z axes are all in same unit, and

[19] A Projected CRS is defined on a flat, two-dimensional surface. Unlike a Geographic CRS, a Projected CRS has constant lengths, angles, and areas across the two dimensions. A Projected CRS is always based on a Geographic CRS that is based on an ellipse.
[20] These CRSs are based on a Geodetic datum. The EPSG dataset contains three subtypes of Geodetic CRS: Geocentric, Geographic 3D, Geographic 2D. ISO 19111 Compliance Note: In ISO19111, geog2D, geog3D and geocentric are all considered to be "geodetic CRSs".
[21] WGS 84 2d as specified here: http://www.epsg-registry.org/report.htm?type=selection&entity=urn:ogc:def:crs:EPSG::4326&reportDetail=long&style=urn:uuid:report-style:default-with-code&style_name=OGP%20Default%20With%20Code&title=

with a per-node offset (from the center point of the node's minimum bounding sphere) for all vertex positions.

3. Axis Order: Axis order explicitly defined by the CRS *SHALL* be used when present. When the axis order is not defined by the CRS, Easting, Northing, Height axis order *SHALL* be used. The Height axis *SHALL* always point upwards towards the sky (away from the center of the earth).

All I3S layers indicate the coordinate reference system via the `spatialReference` property in the 3dSceneLayerInfo resource. This property is normative.

7.2 Height Models

The I3S standard accommodates declaration of a vertical coordinate reference system that may either be ellipsoidal (height defined with respect to a reference ellipsoid) or gravity-related height (height defined with respect to a reference geoid/gravity surface). This allows the I3S approach to be applied across a diverse range of fields and applications where the particular definition of height is of importance.

The Well-known Text (WKT) string representation of the CRS now includes the vertical coordinate reference system utilized by the layer. The *spatialReference* property also includes a Well-known Id (wkid) and a Vertical Coordinate Reference System Well-known ID (*vcsWkid*) representations, which could alternatively be utilized by a client application consuming the layer instead of the WKT. In addition to the detailed spatialReference property that describes the layers horizontal and vertical CRSs, the 3dSceneLayerInfo resource also includes a coarse metadata property called *heightModelInfo*, which can be used by a client application to quickly identify if the layers' height model is either gravity-related or ellipsoidal.

WKT1 description of WGS 84, EPSG 4326

```
"spatialReference": // the horizontal and vertical coordinate reference
system of the layer
    {
        "wkid": 4326,
        "latestWkid": 4326,
        "vcsWkid": 3855,
        "latestVcsWkid": 3855,
        "wkt":
"GEOGCS[\"GCS_WGS_1984\",DATUM[\"D_WGS_1984\",SPHEROID[\"WGS_1984\",637
8137,298.257223563]],PRIMEM[\"Greenwich\",0],UNIT[\"Degree\",0.01745329
2519943295]],

VERTCS[\"EGM2008_Geoid\",VDATUM[\"EGM2008_Geoid\"],PARAMETER[\"Vertical
_Shift\",0.0],PARAMETER[\"Direction\",1.0],UNIT[\"Meter\",1.0]]}"
    }
```

WKT2 description of a compound WGS 84, EPSG 4326 and EPSG 3855

```
COMPOUNDCRS ["I3S Compund CRS",
GEODCRS["WGS 84",
  DATUM["World Geodetic System 1984",
    ELLIPSOID["WGS 84",6378137,298.257223563,LENGTHUNIT["metre",1.0]]],
  CS[ellipsoidal,2],
    AXIS["latitude",north,ORDER[1]],
    AXIS["longitude",east,ORDER[2]],
    ANGLEUNIT["degree",0.01745329252],
  ID["EPSG",4326]]
VERTCRS["EGM2008 height",
  VDATUM["EGM2008 geoid"],
  CS[vertical,1],
    AXIS["gravity-related height (H)",up],
    LENGTHUNIT["metre",1.0],
  ID["EPSG",3855]]
```

HeightModelInfo

```
    "heightModelInfo":  // a coarse metadata indicating the layers
height Model
    {
        "heightModel": "gravity_related_height", //one of {*"
gravity_related_height"*, "ellipsoidal"};
        "ellipsoid": "wgs84 (G1674)/", //datum realization
        "heightUnit": "meter" //units
        }
```

The above examples illustrate the coordinate reference system and height model of a layer in an I3S payload. The spatialReference object includes a Well-known Text (WKT) string representation of the CRS for both horizontal and vertical coordinate reference systems. The examples provided above show both WKT1 and WKT2 WKT encodings as defined in OGC 12-063r5 - either may be encoded in the spatialReference object.The heightModelInfo object is coarse metadata that could be used by client application to quickly determine if the layers' horizontal and vertical coordinate reference systems align with that of any base map data used by the application.

See Class 3dSceneLayerInfo (Clause 7.5.4) for more information.

7.3 Indexed Scene Layers - Organization and Structure

I3S organizes information using a hierarchical, node-based spatial index structure in which each node's payload may contain features with associated geometry, textures and attributes. The following sections define this structure.

7.3.1 I3S - Indexing Model and Tree Structure

The purpose of any index is to allow fast access to blocks of relevant data. In an Indexed 3D Scene layer, the spatial extent of the data is split into regions, called *nodes*, with roughly equal amounts of data, and organized into a hierarchical and navigable data structure - the index - that allows the client to quickly discover which data it actually needs and the server to quickly locate the data requested by any client. Node creation is capacity driven - the smaller the node capacity is, typically the smaller the spatial extent of each node will be.

I3S is agnostic with respect to the model used to index objects/features in 3D space. Both regular partitions of space (e.g. Quadtrees[22] and Octrees[23]) as well as density dependent partitioning of space (e.g. R-Trees[24]) are supported. The specific partitioning scheme is hidden from clients who navigate the nodes in the tree exposed as web resources. The partitioning results in a hierarchical subdivision of 3D space into regions represented by nodes, organized in a bounding volume tree hierarchy (BVH). Each node has an address and nodes may be thought of as equivalent to tiles.

[22] https://en.wikipedia.org/wiki/Quadtree
[23] https://en.wikipedia.org/wiki/Octree
[24] https://en.wikipedia.org/wiki/R-tree

All Nodes have an ID that is unique within a layer. There are two types of Node ID formats supported by I3S: As string based *treekeys* or as integers based on a fixed linearization of the nodes.

In the *treekey* format, which is loosely modeled on binary search trees, the key value is used to indicate both the level and sibling association of a given node, the key directly indicates the position of the node in the tree, allowing sorting of all resources on a single dimension. Treekeys are strings in which levels are separated by dashes: "3-1-0" has 3 numeric elements, hence the node is on level 4 ("root" node is at level 1) and the node "3-1" is its parent. The root node always gets ID `"root"`. An example of this numbering pattern is shown in Figure 1 below.

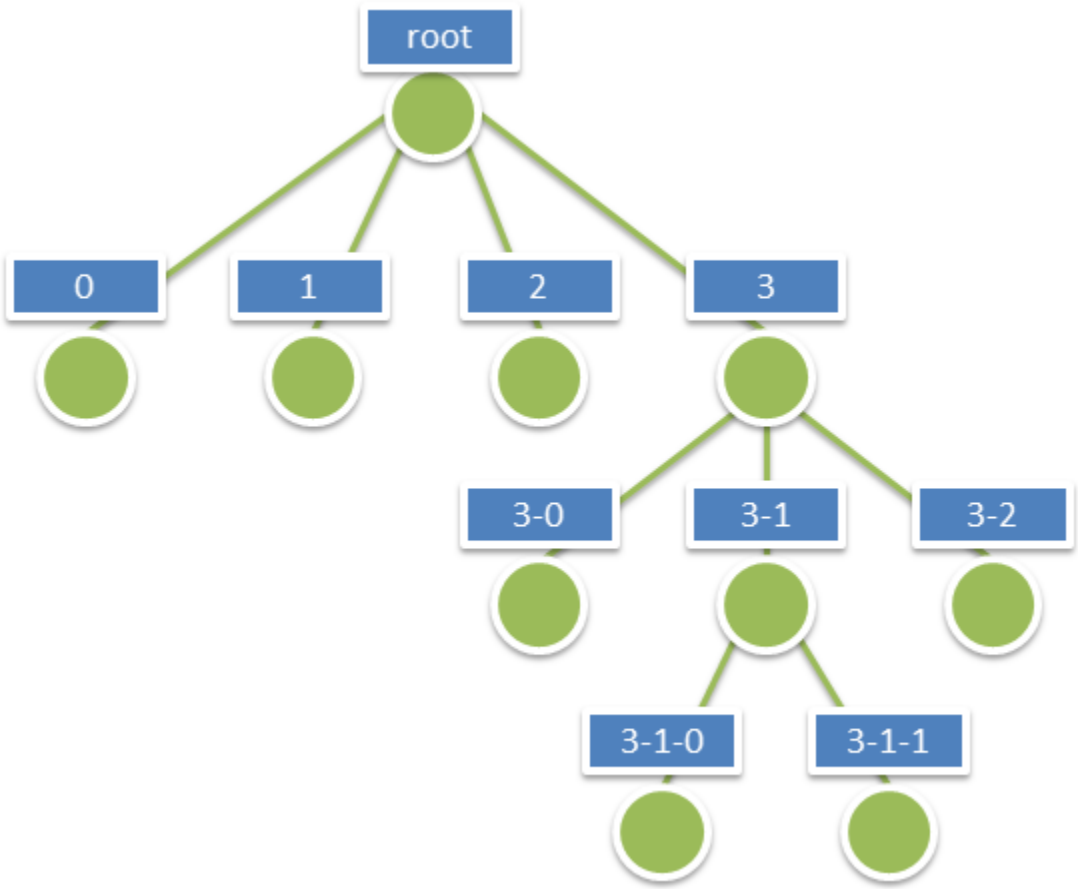

Figure 1: A Sample Index Tree with Treekeys

The information for a node is stored in multiple individually accessible resources. The node index document is a lightweight resource that captures the BVH tree topology for the node, in addition to the node's bounding volume and meta-data used for [LoD Switching] (LoD Switching Models) metrics. This resource allows for tree traversal without the need to access the more voluminous content associated with a node (geometry, texture data, attributes). The decision to render the node is based on node's bounding-volume visibility in the current 3D view and a visual quality determination made by the client using the information included in the node index document. The

node's quality is estimated as a function of current view parameters, node's bounding volume and LoD selection metric value of the node.

The standard supports both minimum bounding spheres (MBS) and oriented bounding boxes (OBB) as a node's bounding volume.

Each interior node logically contains or covers the set of information covered by the nodes below it and participates in a path to the leaf nodes below it. Interior nodes may contain generalized or reduced representation of the information contained in descendant nodes.

The I3S format models node information using a set of resources - Node Index Documents, Feature Data, Geometry, Attributes, Textures and Shared Descriptors, all of which together represent the set of features or data elements for a given node. These resources are always attached to a node.

- The Node Index Document is a lightweight resource representing a node, its topology within the tree and includes references to other sub-resources.
- The Feature Data sub-resource for a node is a text resource that contains the identifiers for the set of features within a node. It can store the geometry and attributes for all of the features in the node either by value or as references into the geometry and attribute sub-resources for the node.
- The Geometry, Attribute and Texture sub-resources describe the geometry, attribute and texture for the node. Geometry and attribute sub-resources represent the geometries and attributes of all of the features within the node and include the identifiers of the owning features within the node as well as the mapping between individual feature identifiers and their geometry segments. Vertices within the geometry contain the appropriate texture coordinates.

An I3S profile can choose between a single text-based feature-data sub-resource that contains all geometry and attribute information (e.g. *Point* profile), or separate, binary and self-contained geometry and attribute sub-resources (e.g. *mesh-pyramids* profile). Applications accessing the latter do not need to first fetch the feature-data resource in order to interpret them.

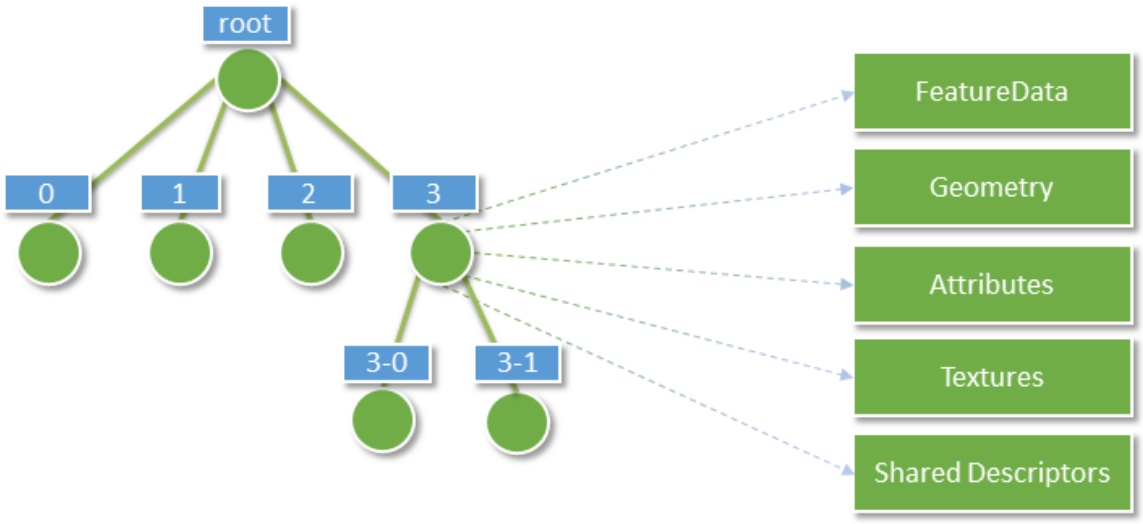

Figure 2: Nodes and their attached resources

Per node, there is exactly one Node Index Document and one Shared Descriptors resource document. FeatureData, Geometry, Texture and Attribute resources can be split into bundles for optimal network transfer and client-side reactivity. This allows balancing between index size, feature splitting (with a relatively large node capacity between 1MB and 10MB) and optimal network usage (with a smaller bundle size, usually in the range of 64kB to 512kB).

There are always an equal number *n* of FeatureData and Geometry resources, and each set contains the corresponding data elements to be able to render a complete feature. Optimal access to all required properties of the geometry data, including the feature to geometry mapping, is available directly from the binary *geometry* data resource, avoiding unnecessary dependency on the FeatureData document. All vertexAttributes (including position, normal, texture coordinates and color), vertex and feature counts, and mesh segmentation information (faceRanges) are also readily accessible from the *geometry* resource.

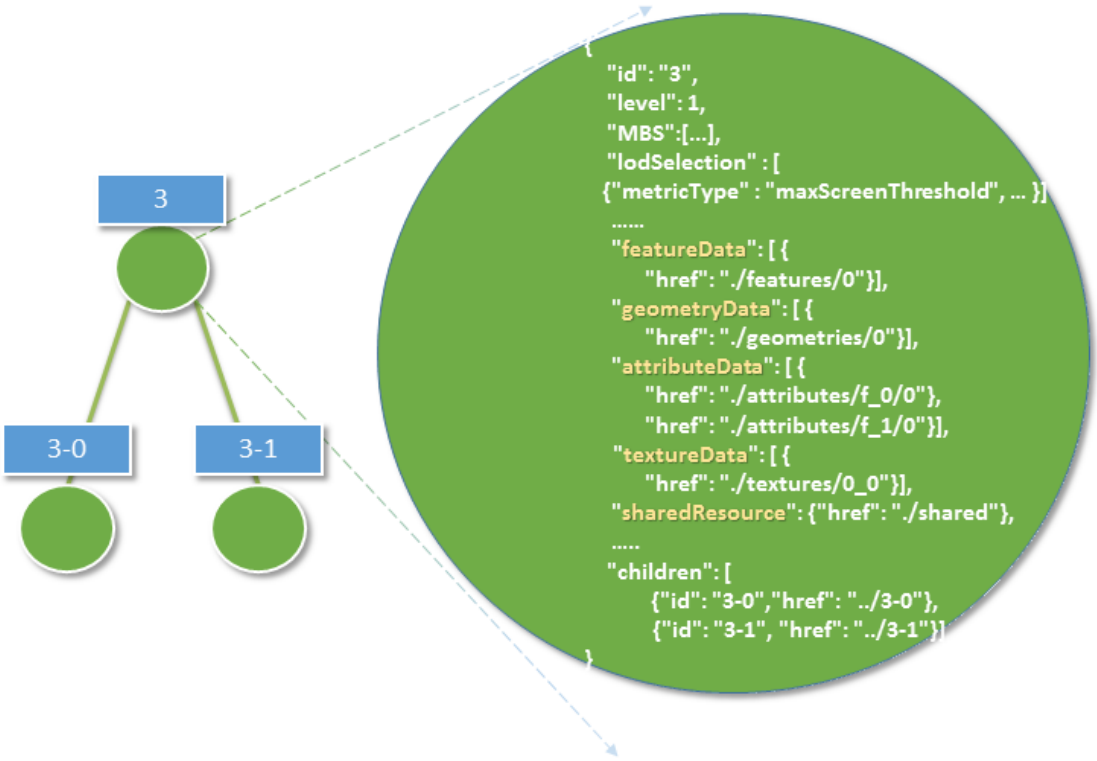

Figure 3: This diagram illustrates the content of an I3S node as stored in its node index document

Figure 4 below shows the node tree of an Indexed Scene Layer whose layer type is 3D Object and whose profile is mesh-pyramids. In the figure:

- `Nodes` are in green, where the hyphenated numbers within the blue boxes represent the identifier or address for each node.
- The orange boxes indicate the `features` explicitly represented within the node, where the numbers within the box represent feature identifiers.
- Each node has associated geometry, texture and attribute resources that compactly store the `geometries`, `attributes` and `textures` of all of the features explicitly represented by the node, as typed arrays and texture atlases.
- The turquoise boxes show the `geometry` resource associated with each node. Each geometry resource is an array of geometries. The same resource also stores the mesh-segmentation information, where each individual feature's range of triangles is stored along with the feature identifier (the values in the orange boxes) in a compact form similar to a run length encoding[25].
- Though both attribute and texture resources are omitted from the figure for clarity, it is worth noting that the attribute of all features of a given node are also stored as `attribute` resource of the node, following a similar storage model.

[25] Run-length encoding (RLE) is a very simple form of lossless data compression in which runs of data (that is, sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than as the original run.

- Each node contains explicit references (the green lines) to the child nodes below it in the bounding volume hierarchy. Each node logically covers all of the features covered by the nodes in its sub-tree, though only some of them may be explicitly represented within the node. Applications make the decision (based on the nodes LoD Selection Metrics) on using the representation within the node versus descending to more detailed nodes.
- The figure also illustrates the case where feature "6" has been generalized away at the lower level of detail node (node "3") and is intentionally no longer explicitly represented within its payload.

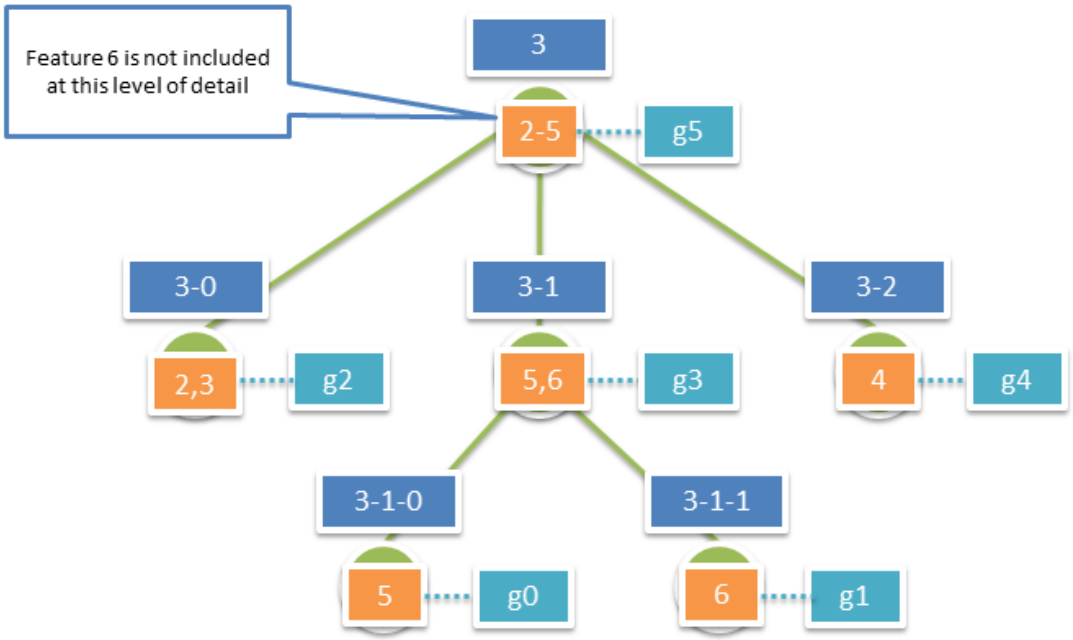

Figure 4: Example Nodes in a Mesh Pyramid

Figure detail: *Orange boxes represent features stored explicitly within the node, the numbers represent feature identifiers. Turquoise boxes represent the geometry instances associated with each node – each geometry instance is an aggregate geometry (a geometry collection) that covers all the features in the node. Blue boxes represent the node ids, the hyphenated numbers represent node ids as string based treekeys.*

7.3.2 Geometry Model and Storage

All Scene Layer types make use of the same fundamental set of geometry types:

- points
- lines
- triangles

Geometries use binary storage and consumption representation, controlled by Array Buffer View[26] geometry property declarations. I3s provides full control over those properties, such as per-vertex layout of components (e.g. position, normal and texture coordinates), in order to ensure the same pattern for face and vertex elements across the Scene Layer.

I3S supports storage of triangle meshes via *triangles* geometry type.

Both 3D Object as well as Integrated Mesh layer type model geometries as triangle meshes using the mesh-pyramids profile. The mesh-pyramids profile uses the triangles geometry type to store triangle meshes with reduced level of detail representations of the mesh, segmented by features, available in the interior nodes as described above.

See Geometry section for more discussion on the geometry format and storage models.

7.3.3 Textures

Textures are stored as a binary resource associated with a node. The texture resource for a node contains the images that are used as textures for the features stored in the node. The mesh-pyramids profile supports either a single texture or a texture atlas per node.

By default, the mesh-pyramids profile allows/supports encoding the same texture resource in multiple formats, catering for bandwidth, memory consumption and optimal performance consideration on different platforms. As a result, the I3S standard supports most commonly used image formats such as JPEG/PNG as well as rendering optimized compressed texture formats such as S3TC[27]. In all cases, the standard provides flexibility by allowing authoring applications to provide additional texture formats via the `textureEncoding` declarations that use MIME types. For example, most existing I3S services provide "image/vnd-ms.dds" (for S3TC compressed texture) in addition to the default "image/jpeg" encoding.

See Textures section for more on texture format, texture coordinate, texture atlas usage and regions discussion.

7.3.4 Attribute Model and Storage

I3S supports the following two patterns of accessing the attribute data:

1. From optional paired services that expose query-able and updatable RESTful endpoints that enable direct access to dynamic source data, including attributes. The query in this case uses the unique feature-ID key – which is always

[26] JavaScript: ArrayBufferView is an abstract type that is the base for the following types: `DataView`, `Float32Array`, `Float64Array`, `Int8Array`, `Int16Array`, `Int32Array`, `Uint8Array`, `Uint8ClampedArray`, `Uint16Array`, `Uint32Array`.
[27] https://en.wikipedia.org/wiki/S3_Texture_Compression

maintained within each node and is also available as part of the descriptor for any segmented geometry.
2. From fully cached attribute information, in binary form, within I3S store. I3S clients can still choose to use both of these modes even if the attributes are fully cached within I3S store.

Cached Attributes use a binary storage representation based on Array Buffers which provide significant performance benefits relative to method 1. The attribute values are stored as a geometry aligned, per field (column), key-value pair arrays.

See Attribute Data section for more on texture format, texture coordinate, texture atlas usage and regions discussion.

7.4 Level of Detail Concept

The concept of Level of Detail (LoD) is intrinsic to the I3S standard. Scene Layers may include levels of detail that apply to the layer as whole and serve to generalize or summarize information for the layer, similar to image pyramids and also similar to raster and vector tiling schemes. A node in the I3S scene layer tree could be considered the analog of a tile in a raster or vector tiling scheme. Scene layers support levels of detail in a manner that preserves the identity of the individual features that are retained within any level of detail.

The I3S Level of Detail model covers several use cases, including, splitting up very heavy features such as detailed building or very large features (coastlines, rivers, infrastructure), thinning/clustering for optimized visualization as well as support for representing externally authored multiple LoDs.

Note that the I3S Level of Detail concept is orthogonal to the concept of consolidated storage for a set of geometries within a level of detail, based on for example, the concatenation of geometries/meshes into larger geometry collections/meshes to assist in optimal rendering. In all such cases the consolidated storage makes use of Geometry Array Buffers that provide access to individual geometries when needed, and include the preservation of feature to geometry element mapping within the consolidated geometries.

7.4.1 Discrete LoDs

I3S supports a *Discrete* LoD approach, where different Level of Details are bound to the different levels of the index tree. Typically, leaf nodes of such LoD schema contain the original (feature/object) representation with the highest detail. The closer nodes are to the root, the lower the level of detail will be. For each next lower level, the amount of data is typically reduced by employing methods such as texture down-sampling, feature reduction/generalization, mesh reduction/generalization, clustering or thinning, so that all inner nodes also have a balanced weight. Generalization applies to the Scene Layer as a whole and the number of discrete levels of detail for the layer corresponds to the number of levels in the index tree for the scene layer. Here, the level of detail concept is

analogous to the level of detail concepts for image pyramids as well as for standard raster and vector tiling schemes.

During navigation and traversal of the I3S tree nodes, clients must decide to either:

1. Discontinue traversal to node's children if the node is not visible in the current 3D view; or
2. Use/render the data within a node if its quality is appropriate to the current 3D view and discontinue further traversal to children nodes; or to
3. Continue traversal until children nodes with better quality are found.

These decisions are made using the advertised values for LoD selection metrics that are part of the information payload of the node. The I3S standard describes multiple LoD Selection Metrics and permits different LoD Switching Models. An example LoD selection metric is the maximum screen size that the node may occupy before it must be replaced with data from more detailed nodes. This model of discrete LoD rendering (LoD Switching Model) is referred to in I3S as `node-switching`.

I3S Scene Layers also include additional optional metadata on the LoD generation process (e.g. thinning, clustering and generalization) as non-actionable (to clients) information that is of interest to some service consumers.

7.4.2 Representation of input data that already has explicitly authored multiple representations

I3S Layers can be used to represent input 3D geographic data that already have multiple, semantically authored, levels of detail.

The most common method for doing so is to represent each semantically authored input level of detail as its own I3S Layer with visibility thresholds on the layer that capture the range of distances (from the 3D location of the camera) at which the layer should be used. At further or closer distances applications switch to using a different I3S layer representing a different input semantically authored level of detail. The set of such I3S Layers representing a single modeled, real world phenomena (such as buildings for a city) can be grouped within the same I3S service. For each I3S Layer within the set, the features in the leaf nodes of the index tree represent the modeled features at the level of detail presented in the input. Additional automatically generated levels of detail can optionally be generated extending the viewing range of each semantically input level of detail if so desired.

Tools can also be developed that load all of the input level of detail information for the modeled entities in the input into a single I3S layer. In this case the height of the I3S index tree is fixed to the number of levels of detail present in the input and both the feature identities and geometries in each node are set based upon the input data.

The specific approach taken is influenced by the extent of the data, the number of levels of detail actually present in the input and the need for further additional automatically generated levels of detail.

7.4.3 LoD Switching Modes

Depending on the properties of a 3D layer, a good user experience will necessitate switching out the content for a node with the content of more detailed nodes.

7.4.3.1 Node Switching

Node switching means that the content (features, geometry, attributes, textures) from child nodes is loaded to replace the content of an existing node as the user needs to be presented with more detailed information

As shown in Figure 4 above, each interior node in the I3S tree has a set of features that represent the reduced LoD representation of all of the features covered by that interior node. Not all features may be present in reduced LoD nodes. Omission of a feature at a reduced LoD node indicates that the entire feature has been intentionally generalized away at this level of detail.

The correspondence between a reduced LoD feature in an interior node and the same feature in descendant (children) nodes is based on by feature IDs which are a key part of the storage model. Applications accessing the I3S tree can display all of the features in an internal node and stop there or instead descend further and use the features found in its child nodes, based on desired quality.

The main advantage of this mechanism is that clients can focus on the display criterion associated with nodes as a whole in making the decision to switch representations. `node-switching` is the default LoD Switching model for layer types that implement the `Mesh-pyramids` profile.

7.4.4 Levels of Detail – Generation

Integrated Mesh layer types typically come with pre-authored Levels of Detail. For input data that does not come with pre-authored LoDs, different LoD generation models can be employed. For example, 3D Object layers based on the `Mesh-pyramids` profile may choose to create an LoD pyramid for all features based on generalizing, reducing and fusing the geometries (meshes) for individual features while preserving feature identity. The same approach can also be used with Integrated Mesh layers based on the `mesh-pyramid` profile - in this case there are no features and each node contains a generalized version of the mesh covered by its descendants.

The first step in the automatic LoD generation process is to build the I3S bounding volume tree hierarchy based on the spatial distribution of the 3D features. Once this has been completed generation of the reduced LoD content for interior nodes can proceed.

As shown in Table 2 below, different models of LoD generation are applicable to different 3D layers.

	3D Object	Points
Mesh-pyramids	yes	
Thinning	yes	yes
Clustering	yes	yes
Generalization	yes	

Table 2: 3D Layer Types and models of LoD generation they can employ

7.4.5 LoD Selection Metrics

A client needs information to determine whether a node's contents are "good enough" to render in the current 3D view under constraints such as resolution, screen size, bandwidth and available memory and target minimum quality goals. Multiple LoD selection metrics can be included, as in the following example:

```
"lodSelection": [
    {
        "metricType": "maxScreenThreshold",
        "maxError": 486.00
    },
    {
        "metricType": "screenSpaceRelative",
        "maxError": 0.0034
    },
    {
        "metricType": "distanceRangeFromDefaultCamera",
        "maxError": 750.00
    }
]
```

These metrics are used by clients to determine the optimal resource access patterns. Each I3S profile definition provides additional details on LoD Selection.

`maxScreenThreshold`, the default lodSelection metric used for meshpyramids profile, is a per-node value for the maximum pixel size as measured in screen pixels. This value indicates the upper limit for the screen size of the diameter of the node's minimum bounding sphere (MBS). In other words, the content referenced by this node will qualify to be rendered only when the screen size is below the maximum screen threshold value.

7.5 JSON Resources Schema and Documentation

This section provides a detailed, logical-level specification for each of the resource types.

7.5.1 Basic Value Types

Value schemas are used to ensure that the content of a JSON property follows a fixed pattern. The set of schemas that currently need to be supported iare

- **String**: An utf8 String.
- **Float**: A Float64 number with an optional fractional component, such as "1.02" or "1.0".
- **Integer**: An Int32 number without a fractional component, such as "234".
- **UUID**: A canonical hexadecimal UUID, such as "550e8400-e29b-41d4-a716-446655440000".
- **Date**: An ISO 8601 timestamp YYYY-MM-DDThh:mm:ss.sTZD, with a fixed "Z" timezone, such as "2009-01-01T12:00:00.000Z".
- **URL**: Any resolvable, relative or absolute, URL, such as "../Node/51/sharedResource".
- **Pointer**: Any resolvable reference to an object in a JSON document, consisting of a relative or absolute URL and a document path, such as [../Node/51/sharedResource]/materialDefinitions/Mat01 .
- **NodeID**: A treekey string such as “3-0-34-234-2” that is zero-based (first child is "0", root node is "root").

7.5.2 Pointers

I3S uses the following Pointer syntax whenever a specific property in the current or another document is to be referenced. The Pointer consists of two elements:

1. **mandatory in-document reference:** Relative to the currently evaluated property, or document absolute, reference to a property. References are always slash-separated paths through a document tree and can contain wildcards (*) to indicate that a set or list of properties is to be matched instead of a single property.
    - *Absolute* references start with a slash (/). Absolute references may only contain upstream path elements, i.e. they may only point to properties of objects enclosing the property that is being evaluated and indicated by a name.
        - Example: `/materialDefinitions/*/type`
    - *Relative* references start with a property key (e.g. type). Relative properties may only contain downstream path elements and are evaluated from the value being tested. They may not contain wildcards, as appropriate context is already given through the current element being evaluated. In the case of a property that has containerType set to Array or Object, the reference point for a relative path is the individual value element in the container.
        - Example: `params/ambient/0`
2. **optional URL:** The pointer may be prefixed with a URL to a different document. This URL may be relative to the document that is being evaluated or absolute. To identify the URL element of a pointer, it is given in square brackets. Examples:

- *relative URL + absolute reference:* From FeatureData to 3dSceneLayer.name: `[../../]/name`

- *absolute URL + absolute reference:* `[http://<my_server>/<my_service>/rest/services/Buildings_Portland/SceneServer/layers/0/nodes/68](http://>my_server>/tiles/P3ePLMYs2RVChkJx/<my_service>/rest/services/Buildings_Portland/SceneServer/layers/0/nodes/68)`

7.5.3 SceneServiceInfo

The SceneServiceInfo file is a JSON file that describes the capability and data sets offered by an instance of a Scene Service. A Scene Service is a web service that provides access to 3D available in some data store in which 3D content has been authored and is ready for publication (visualization). The SceneServiceInfo has the following structure:

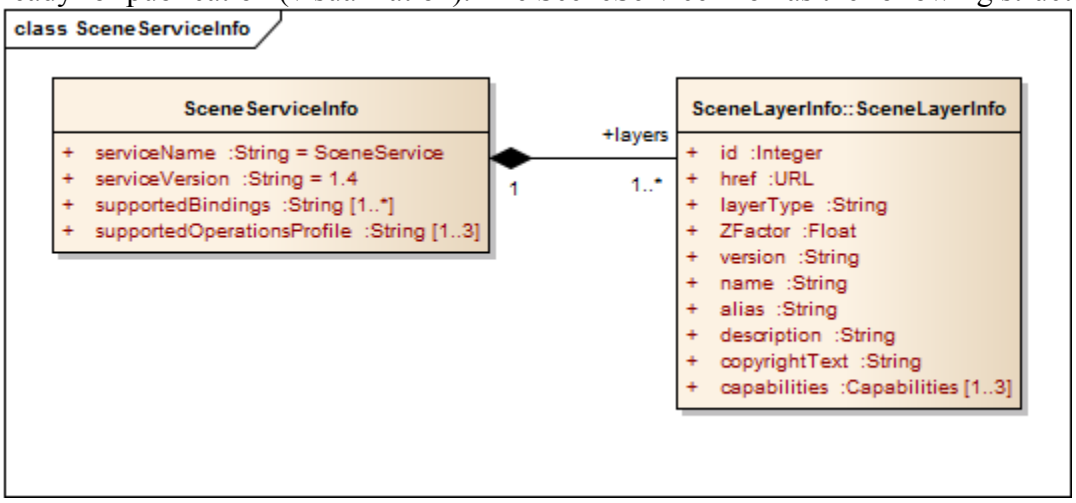

Figure 5: Logical Schema of the 3dSceneServiceInfo document

This file is automatically generated by a Scene Server for each service instance and is not part of a scene layer package file. It is included here only for reference.

7.5.3.1 Class SceneServiceInfo

SceneServiceInfo is the major object in the 3dSceneServiceInfo document. There SHALL always be exactly one SceneServiceInfo object in the document, which describes a running SceneService instance.

Name	Type	Description
serviceName	String	The type of the service; always SceneService.
serviceVersion	String	The version of the service protocol/REST endpoint.
supportedBindings	String[1..*]	the list of bindings, should we ever need to add new bindings in addition to the REST binding initially supported
supportedOperations	String[1..3]	Supported profiles of the service from the

		choice {Base, Dynamic, Editing}.
layers	3dSceneLayerInfo[1..*]	The full 3dSceneLayerInfo information.

Table 3: Attributes of Class SceneServiceInfo within *SceneServiceInfo* document

7.5.4 3dSceneLayerInfo

The Class 3dSceneLayerInfo describes the properties of a single layer in a store, including the default symbology to use. It shares the definition of this default symbology with the *drawingInfo* object, an object which contains styling information for a feature layer, and is specified as part of a web scene specification. For more information on web scene objects, including the *drawingInfo* object see Clause 7.5.4.8. The Class 3dSceneLayerInfo has the following structure:

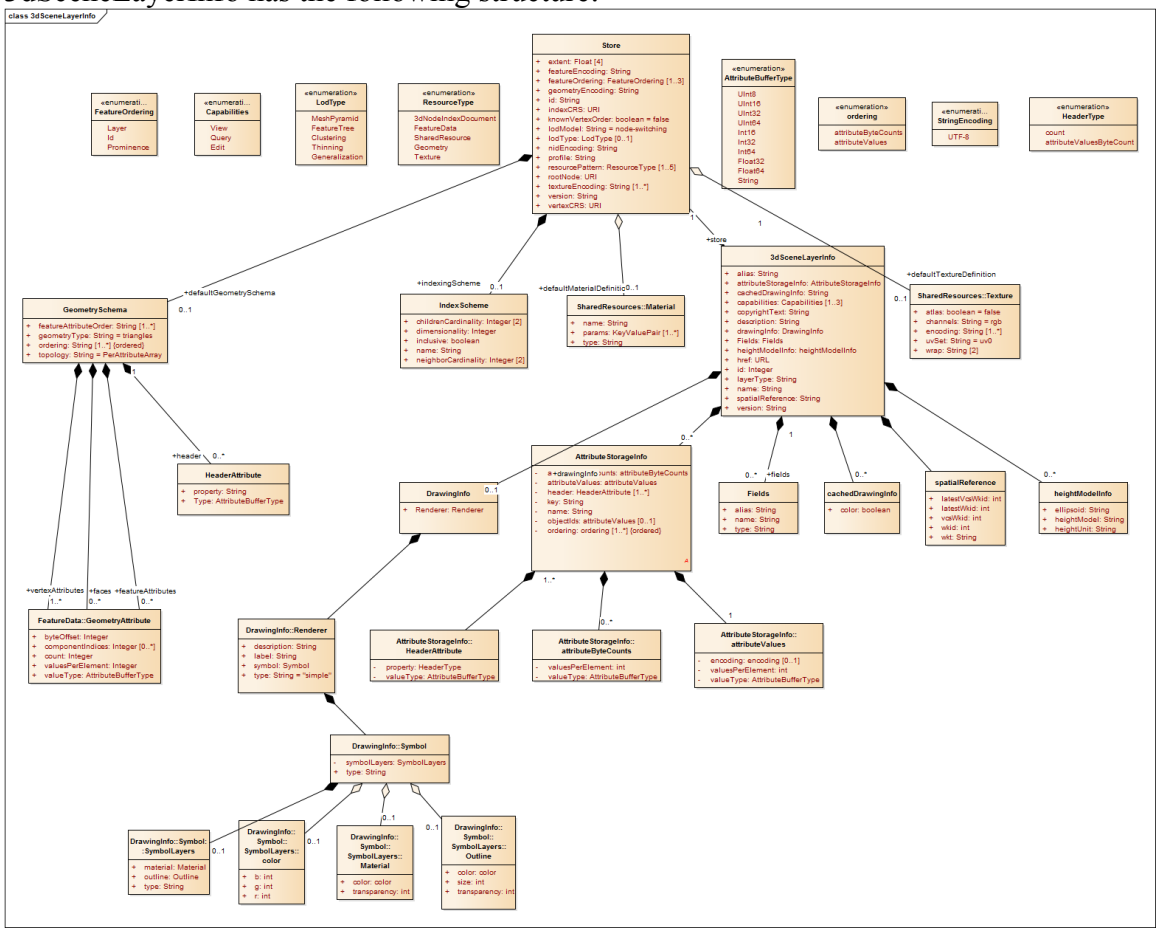

Figure 6: Logical schema of the 3dSceneLayerInfo document

7.5.4.1 Class 3dSceneLayerInfo

The 3dSceneLayerInfo is a major object in the 3dSceneLayerInfo document. A SceneServiceInfo document can contain 1...* 3dSceneLayerInfo documents. Each 3dSceneLayerInfo object describes a Layer.

Name	Type	Description
id	Integer	Unique numeric ID of the Layer.
href	URL	The relative URL to the 3dSceneLayerResource. Only present as part of the SceneServiceInfo resource.
layerType	String	The user-visible type of this layer, one of {Point, *3DObject*, IntegratedMesh}
spatialReference	spatialReference	The spatialReference of the layer including the vertical coordinate reference system. wkt is included to support custom spatial references. `{wkid, latestWkid, vcsWkid, latestVcsWkid, wkt}`
heightModelInfo	heightModelInfo	Enables consuming clients to perform quick test to determine whether this layer is compatible (with respect to its horizontal and vertical CRS) with existing content.`{heightModel, geoid, heightUnit}`
version	String	The ID of the last update session in which any resource belonging to this layer has been updated.
name	String	The name of this layer.
alias	String[0..1]	The display alias to be used for this layer.
description	String[0..1]	Description string for this layer.
copyrightText	String[0..1]	Copyright and usage information for the data in this layer.
capabilities	String[1..3]	Capabilities from the Set `{View, Query, Edit}` that are possible on this layer.
cachedDrawingInfo	cachedDrawingInfo	Indicates if any stylization information represented as *`drawingInfo`* is additionally captured as part of the binary mesh representation for optimal client side access. Currently `color` component of the *`drawingInfo`* is supported.
drawingInfo	drawingInfo	Represents the stylization information of the layer.
fields	fields	A collection of objects that describe each attribute field regarding its field name, datatype and a user friendly name {`name,type,alias`}. It includes all fields that are included as part of the I3S layer as derived from a source input feature layer.
attributeStorageInfo	attributeStorageInfo	Provides the schema and layout used for storing

		attribute content in binary format in I3S.

Table 4: Attributes of the Class 3dSceneLayerInfo within the 3dSceneLayerInfo document

7.5.4.2 Class Store

The Class Store object describes the exact physical storage of a Layer and enables the client to detect when multiple Layers are served from the same Store. Storing multiple layers in a single store - and thus having them share resources - enables efficient serving of many layers of the same content type, but with different attribute schemas or different symbology applied.

Name	Type	Description
id	UUID	A Store ID, unique across a SceneServer. Enables the client to discover which layers a part of a common store, if any.
profile	String	Indicates which profile this scene store fulfills. One of `{meshes, points, analytics, meshpyramids, symbols}`.
resourcePattern	String [1..5]	Indicates the resources needed for rendering and the required order in which the client should load them. Each value is one of `{3dNodeIndexDocument, SharedResource, FeatureData, Geometry, Texture}`.
rootNode	URL	Relative URL to root node resource.
version	String	Format version of this resource; used here again if this store hasn't been served by a 3D Scene Server.
extent	Float[4]	The 2D spatial extent (x_{min}, y_{min}, x_{max}, y_{max}) of this store, in the horizontal indexCRS
indexCRS	URL	The horizontal CRS used for all minimum bounding spheres (mbs) in this store, identified by an OGC URL.
vertexCRS	URL	The horizontal CRS used for all "vertex positions" in this store, identified by an OGC URL.
normalReferenceFrame	String	Describes the coordinate reference frame used for storing normals. One of `{east-north-up, *earth-centered*, vertex-reference-frame}`. A value of *east-north-up* indicates that normals are stored in a node local

		reference frame defined by the easting, northing and up directions at the MBS center, and is only valid for geographic (WGS84) vertexCRS. A value of *earth-centered* indicates that normals are stored in a global earth-centered, earth-fixed (ECEF) reference frame where the x-axis points towards Prime meridian (lon = 0°) and Equator (lat = 0°), the y-axis points East towards lon = +90 and lat = 0 and the z-axis points North. It is only valid for geographic vertexCRS. A value of *vertex-reference-frame* indicates that normals are stored in the same reference frame as vertices and is only valid for projected vertexCRS.
nidEncoding	MIMEType	MIME type for the encoding used for the Node Index Documents; format: `application/vnd.ogc.I3S.json+gzip; version=1.6`
featureEncoding	MIMEType	MIME type for the encoding used for the Feature Data Resources; format: `application/vnd.ogc.I3S.json+gzip; version=1.6`
geometryEncoding	MIMEType	MIME type for the encoding used for the Geometry Resources; format: `application/octet-stream; version=1.6`
textureEncoding	MIMEType[1..*]	MIME type(s) for the encoding used for the Texture Resources
lodType	String	optional field to indicate which LoD Generation Scheme is used in this store. One of `{*MeshPyramid*, Thinning, Clustering, Generalizing}`.
lodModel	String	optional field to indicate which LoD Switching mode clients have to use. One of `{*node-switching*, none}`.
indexingScheme	IndexScheme	Information on the Indexing Scheme (QuadTree, R-Tree, Octree, ...) used.
defaultGeometrySchema	GeometrySchema[0..1]	A common, global ArrayBufferView definition that can be used if the schema of vertex attributes and face attributes is consistent in an entire cache; this is a requirement for meshpyramids caches.
defaultTextureDefinition	TextureDefinition[0..1]	A common, global TextureDefinition (see

		SharedResources) to be used for all textures in this store. The default texture definition uses a reduced profile of the full TextureDefinition, with the following attributes being mandatory: `encoding`, `uvSet`, `wrap` and `channels`.
defaultMaterialDefinition	MaterialDefinition[0..1]	If a store uses only one material, it can be defined here entirely as a MaterialDefinition (see SharedResources).

Table 5: Attributes of the Class *Store* within the 3dSceneLayerInfo document

7.5.4.3 Class GeometryStore

This class is used in stores where all ArrayBufferView geometry declarations use the same pattern for face and vertex elements. This effectively reduces redundancies of ArrayBufferView geometry declarations in a store. Reuses the GeometryAttribute type from FeatureData. However, valueType and valuesPerElement are mandatory and SHALL be implemented.

Name	Type	Description
geometryType	String	Low-level default geometry type, one of `{triangles, lines, points}`; if defined, all geometries in the store are expected to have this type.
topology	String[0..1]	one of `{*PerAttributeArray*, Indexed}`. When "Indexed", the indices must also be declared in the geometry schema ("faces") and precede the vertexAttribute data.
header	HeaderAttribute[0..*]	Defines header fields in the Geometry resources of this store that precede the vertex (and index) data
ordering	String[1..*]	Provides the order of the keys in vertexAttributes and faceAttributes, if present.

vertexAttributes	FeatureData::GeometryAttribute[1..*]	Declaration of the attributes per vertex in the geometry, such as position, normals or texture coordinates
faces	FeatureData::GeometryAttribute[0..*]	Declaration of the indices into vertex attributes that define faces in the geometry, such as position, normals or texture coordinates
featureAttributeOrder	String[1..*]	Provides the order of the keys in featureAttributes, if present.
featureAttributes	FeatureData::GeometryAttribute[0..*]	Declaration of the attributes per feature in the geometry, such as feature ID or face range

Table 6: Attributes of the Class GeometrySchema within the 3dSceneLayerInfo document

7.5.4.4 Class HeaderAttribute

Headers to Geometry resources SHALL be uniform across a cache and may only contain fixed-width, single element fields. The HeaderDefinition provides the name of each field for later access and the valueType of that header field.

Name	Type	Description
property	String	The name of the property in the header
type	String	The element type of the header property, from `{UInt8, UInt16, UInt32, UInt64, Int16, Int32, Int64 or Float32, Float64}`

Table 7: Attributes of the Class HeaderAttribute **within the 3dSceneLayerInfo document**

7.5.4.5 Class Field

The Field class is used to provide schema information for this 3dSceneLayer.

Name	Type	Description
name	String	The name of the field.
type	String	The type of the field, from this enum: `{FieldTypeBlob, FieldTypeGeometry, FieldTypeDate, FieldTypeFloat, FieldTypeDouble, FieldTypeGeometry, FieldTypeGlobalID, FieldTypeGUID, FieldTypeInteger, FieldTypeOID, FieldTypeSmallInteger, FieldTypeString, FieldTypeGroup}`
alias	String[0..1]	The display alias to be used for this field.

Table 8: Attributes of the Class *Field* within the 3dSceneLayerInfo document

7.5.4.6 Class attributeStorageInfo

The attributeStorageInfo is another major object in the 3dSceneLayerInfo document. This is an object that describes the structure of the binary attributeData resource of a node.

Name	Type	Description
key	string	The unique field identifier key.
name	string	The name of the field.
header	String[1..*]	Declares the headers of the binary attribute data. One of {`count`, `attributeValuesByteCount`}. `count`, should always be present and indicates the count of features in the attribute storage. `attributeValuesByteCount` will only be present for strings data type and indicates the total byte count of the string data for all features in the binary attribute buffer.
ordering	String[1..*\]	Declares the ordering indicating the order in which the array of attribute byte counts and the array of attribute values are stored in the binary attribute data. One of {`attributeByteCounts`, `attributeValues`}. `attributeValues`, should always be present. `attributeByteCounts` should only be present when working with string data types.
attributeByteCounts	String	The element type of the attributeByteCounts property, from `{UInt32}`.
attributeValues	String	The element type of the attributeValues property, from `{UInt8, UInt16, UInt32, UInt64, Int16, Int32, Int64 or Float32, Float64}`

Table 9: Attributes of the Class *attributeStorageInfo* within the 3dSceneLayerInfo document

7.5.4.7 Class IndexScheme

The IndexScheme class declaratively describes computational and structural properties of the index used within an I3S store. This information can be used by clients to better understand how to work with the index.

Name	Type	Description
name	String	Name of the scheme, selected from `{RTree, QuadTree, AGOLTilingScheme}`.
inclusive	Boolean	true indicates that the extent and mbs of all children nodes is fully within their parent nodes' extent/mbs
dimensionality	Integer	The number of dimensions in which this index

		differentiates.
childrenCardinality	Integer[2]	min/max number of children per node.
neighborCardinality	Integer[2]	min/max number of neighbors per node.

Table 10: Attributes of the Class *IndexScheme* within the 3dSceneLayerInfo document

7.5.4.8 Class DrawingInfo

DrawingInfo and the associated classes contain the default symbology (drawing information) of an Indexed 3D Scene Layer. When the DrawingInfo object is present in the `3dSceneLayerInfo` Class, a client application may symbolize an I3S layer by utilizing the *Renderer* information. Indexed 3d Scene Layers also supports capturing the DrawingInfo object as part of the binary I3S representation This is to support applications that may not be able to dynamically symbolize/override a given I3S layer based on its drawing information. Such a behavior, when present, is indicated by the `CachedDrawingInfo` Class, indicating the component of the DrawingInfo object that's captured as part of the binary I3S representation. The Class DrawingInfo has the following structure:

Name	Type	Description
renderer	DrawingInfo::Renderer	The renderer object encapsulates the drawing information of the layer.

Table 11 Attributes of the Class *CachedDrawingInfo* within the 3dSceneLayerInfo document

7.5.4.8.1 Class Renderer

The Renderer class contains properties that define the drawing symbology of an Indexed 3D Scene Layer, including its type, symbol and any label or descriptions associated with it.

The Class Renderer has the following structure:

Name	Type	Description
type	String	The renderer type. One of `{\*Simple\*, UniqueValue, ClassBreaks}`. The default, `simple` renderer is a renderer that uses one symbol only.
symbol	Renderer::Symbol	An object that represents how all features of this I3S layer will be drawn.
label	String	The text string that may be used to label a symbol when displayed in a table of content of an application.
description	String	The text string that does not appear in the table of contents but may appear in the legend.

Table 12*: Attributes of the Class Renderer within the 3dSceneLayerInfo document*

7.5.4.8.2 Class Symbol

The Class Symbol represents the render primitive used to symbolize an Indexed 3D Scene Layer. MeshSymbol3D is the only supported type of Symbol.

The Class Symbol has the following structure:

Name	Type	Description
type	String	Specifies the type of symbol used. Value of this property must be `{\*MeshSymbol3D\*}`.
symbolLayers	Renderer::SymbolLayers	An object that represents how all features of this I3S layer will be drawn.

Table 13*: Attributes of the Class Symbol within the 3dSceneLayerInfo document*

7.5.4.8.3 Class SymbolLayers

A Collection of symbol objects used to visualize the feature.

The Class SymbolLayers has the following structure:

Name	Type	Description
type	String	Specifies the type of symbol used. Value of this property must be `{\*Fill\*}`.
material	SymbolLayers::Material	The material used to shade the geometry.
outline	SymbolLayers::Outline	The outline of the mesh fill symbol.

Table 14*: Attributes of the Class SymbolLayers within the 3dSceneLayerInfo document*

7.5.4.8.4 Class Material

The material used to shade the geometry.

The Class Material has the following structure:

Name	Type	Description
color	Material::Color	Color is represented as a three-element array (RGB).
transparency	Integer	Indicates the transparency value associated with the symbol. The value has to lie between 100 (full transparency) and 0 (full opacity).

Table 15*: Attributes of the Class Material within the 3dSceneLayerInfo document*

7.5.4.8.5 Class Outline

The Class Outline defines the outline of the mesh fill symbol. It has properties such as color, size and transparency.

The Class Outline has the following structure:

Name	Type	Description
color	Material::Color	Color is represented as a three-element array. The three elements represent values for red, green and blue in that order.
size	Integer	Outline size in points, positive only.
transparency	Integer	Indicates the transparency value associated with the outline of the symbol. The value has to lie between 100 (full transparency) and 0 (full opacity).

Table 16*: Attributes of the Class Material within the 3dSceneLayerInfo document*

7.5.4.8.6 Class Color

The Color class defines the color of a symbol or the outline. Color is represented as a three-element array. The three elements represent values for red, green and blue in that order. Values range from 0 through 255. If color is undefined for a symbol or an outline, the color value is null.

The Class Color has the following structure:

Name	Type	Description
color	String	The renderer type. One of `{\*Simple\*, UniqueValue, ClassBreaks}`. The default, `simple` renderer is a renderer that uses one symbol only.
symbolLayers	Renderer::Symbol	An object that represents how all features of this I3S layer will be drawn.

Table 17*: Attributes of the Class Color within the 3dSceneLayerInfo document*

7.5.4.8.7 Class CachedDrawingInfo

The Class CachedDrawingInfo is used to indicate if the DrawingInfo object is captured as part of the binary I3S representation.

The Class CachedDrawingInfo has the following structure:

Name	Type	Description
color	Boolean	Indicates if the color component of the `drawingInfo` object is captured as part of the binary I3S representation.

Table 18*: Attributes of the Class CachedDrawingInfo within the 3dSceneLayerInfo document*

7.5.5 3dNodeIndexDocument

The 3dNodeIndexDocument JSON file describes a single index node within a store, with links to other nodes (children, sibling, and parent), links to feature data, geometry data and texture data resources, metadata such as metrics used for LoD selection, and its spatial extent.

Depending on the geometry and `lodModel` used, a node document can be tuned towards being light-weight or more heavy-weight. This is the means by which clients have to further decide which data to retrieve. The bounding volume information provided for the node, its parent, any neighbors and children present, already provides sufficient data for simple visualization by rendering the centroids as point features for example.

The 3dNodeIndexDocument has the following structure:

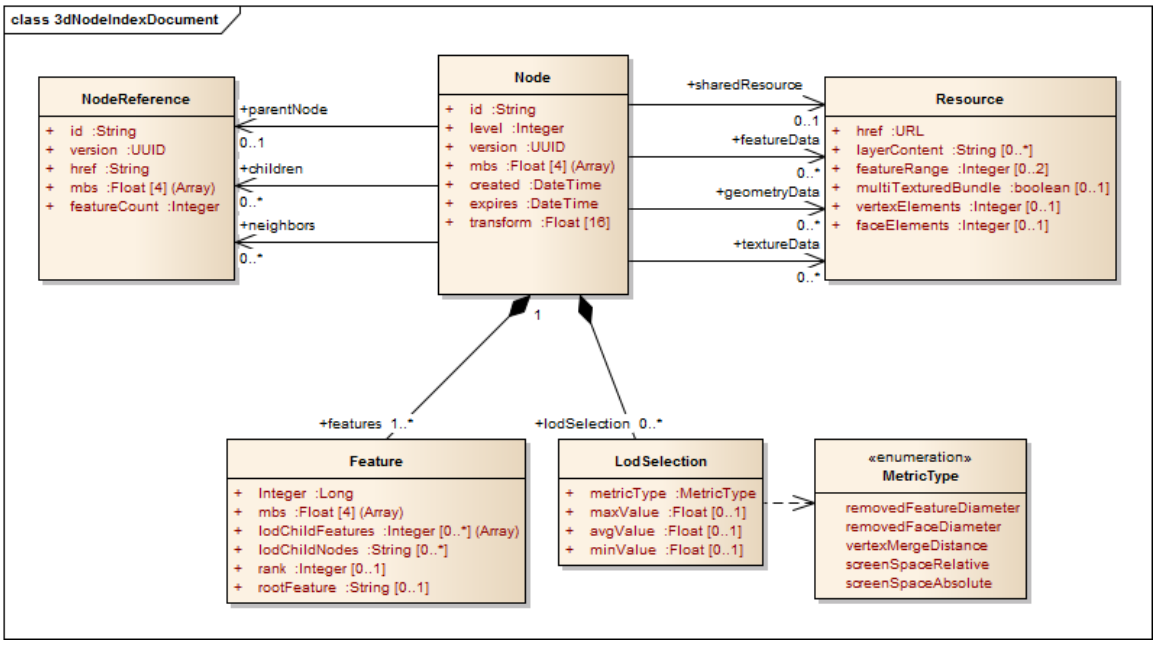

Figure 7: Logical schema of the 3dNodeIndexDocument

7.5.5.1 Class Node

The Node is the root object in the 3dNodeIndexDocument. There *SHALL* always be exactly one Node object in a 3dNodeIndexDocument.

Name	Type	Description
id	String (TreeKey)	Tree Key ID, unique within the store. The root node is always "root", all others follow the pattern "2-4-0-15-2". At each level in a subtree, numbering starts at 0.

level	Integer	Explicit level of this node within the index tree. The lowest level is 1.
version	UUID	The version (store update session ID) of this node.
mbs	Float[4]	An array of four doubles, corresponding to x, y, z and radius of the minimum bounding sphere of a node.
created	Date[0..1]	Creation date of this node in UTC, presented as a string in the format YYYY-MM-DDThh:mm:ss.sTZD, with a fixed "Z" timezone (see http://www.w3.org/TR/NOTE-datetime).
expires	Date[0..1]	Expiration date of this node in UTC, presented as a string in the format YYYY-MM-DDThh:mm:ss.sTZD, with a fixed "Z" timezone (see http://www.w3.org/TR/NOTE-datetime).
transform	Float[16]	Optional, 3D (4x4) transformation matrix expressed as a linear array of 16 values.
parentNode	NodeReference[0..1]	Reference to the parent Node of a Node.
children	NodeReference[0..*]	Reference to the child Nodes of a Node.
neighbors	NodeReference[0..*]	Reference to the neighbor (same level, spatial proximity) Nodes of a Node.
sharedResource	Resource[0..1]	Resource reference describing a shared resource document.
featureData	Resource[0..*]	Resource reference describing a FeatureData document.
geometryData	Resource[0..*]	Resource reference describing a geometry resource.
textureData	Resource[0..*]	Resource reference describing a texture resource.
lodSelection	LodSelection[0..*]	Metrics for LoD Selection, to be evaluated by the client.
features	Feature[1..*]	A list of summary information on the features present in this Node, used for pre-visualisation and LoD switching in featureTree LoD stores.

Table 19: Attributes of the Class *Node* within the NodeIndexDocument

7.5.5.2 Class NodeReference

A NodeReference is a pointer to another node - the parent, a child or a neighbor. NodeReferences contain a relative URL pointing to the referenced NID, as well as a set of meta information that can be used by the client to determine whether to load that node or not, as well as maintaining store consistency.

Name	Type	Description

id	String	Tree Key ID (e.g. "1-3-0-5") of the referenced node.
mbs	Float[4]	An array of four doubles, corresponding to x, y, z and radius of the minimum bounding sphere of the referenced node.
href	URL	The relative URL to the referenced node resource.
version	UUID	Version (store update session ID) of the referenced node.
featureCount	Integer	Number of features in the referenced node and its descendants, down to the leaf nodes.

Table 20: Attributes of the Class *NodeReference* within the NodeIndexDocument

7.5.5.3 Class Resource

Resource objects are pointers to different types of resources related to a node, such as the feature data, the geometry attributes and indices, textures and shared resources.

Name	Type	Description
href	String	The relative URL to the referenced resource.
layerContent	String[1..*]	The list of layer names that indicates which layer features in the bundle belongs to. The client can use this information to selectively download bundles.
featureRange	Integer[2]	Only applicable for featureData resources. Provides inclusive indices of the features list in this node that indicate which features of the node are located in this bundle.
multiTextureBundle	Boolean	Only applicable for textureData resources. `true` if the bundle contains multiple textures. If `false` or not set, clients can interpret the entire bundle as a single image.
vertexElements	Integer[0..1]	Only applicable for geometryData resources. Represents the Count of elements in vertexAttributes; multiply by the sum of bytes required for each element as defined in the defaultGeometrySchema.
faceElements	Integer[0..1]	Only applicable for geometryData resources. Represents the Count of elements in faceAttributes; multiply by the sum of bytes required for each element as defined in the defaultGeometrySchema.

Table 21: Attributes of the Class *Resource* within the NodeIndexDocument

7.5.5.4 Class Feature

Features are representations of the geographic objects stored in a layer. In the 3dNodeIndexDocument, these objects define relationships, e.g. for linking feature representations of multiple LoDs.

Name	Type	Description

id	Integer	An ID of the Feature object, unique within the store (important to note when using Features from multiple stores!). The ID SHALL not be re-used e.g. for multiple representation of an input feature that are present in different nodes.
mbs	Float[4]	An array of four doubles, corresponding to x, y, z and radius of the minimum bounding sphere of the referenced node.
lodChildFeatures	Integer[0..*]	IDs of features in a higher LoD level which together make up this feature.
lodChildNodes	String[0..*]	Tree Key IDs of the nodes in which the lodChildFeatures are found
rank	Integer[0..1]	The LoD level of this feature. Only required for features that participate in a LoD tree. The lowest rank SHALL be 1.
rootFeature	String	The Tree Key ID of the root node of a feature LoD tree that this feature participates in. Only required if the feature participates in a LoD tree and if it is not the rootFeature itself.

Table 22: Attributes of the Class *Feature* within the NodeIndexDocument

7.5.5.5 Class LodSelection

A LodSelection object provides information on a given metric determined during the cooking process of an I3S store. This metric can be used by the client to determine whether a representation is of the right quality level for rendering or whether a different representation is needed.

Publishers (aka “cookers”) can add as many `LodSelection` objects as desired but must provide one as soon as the layer's `lodType` is not null. Of the three min/avg/max values, typically only one or two are used.

Name	Type	Description
metricType	String	The name of the error metric, one of `{maxScreenThreshold, screenSpaceRelative, ...}`
maxValue	Float[0..1]	Maximum metric value, expressed in the CRS of the vertex coordinates or in reference to other constants such as screen size
avgValue	Float[0..1]	Average metric value, expressed in the CRS of the vertex coordinates or in reference to other constants such as screen size
minValue	Float[0..1]	Minimum metric value, expressed in the CRS of the vertex coordinates or in reference to other constants such as screen size

Table 23: Attributes of the Class *LodSelection* within the NodeIndexDocument

7.5.6 FeatureData

The FeatureData JSON file(s) contain geographical features with a set of attributes, accessors to geometry attributes and other references to styling or materials. Features have the following structure:

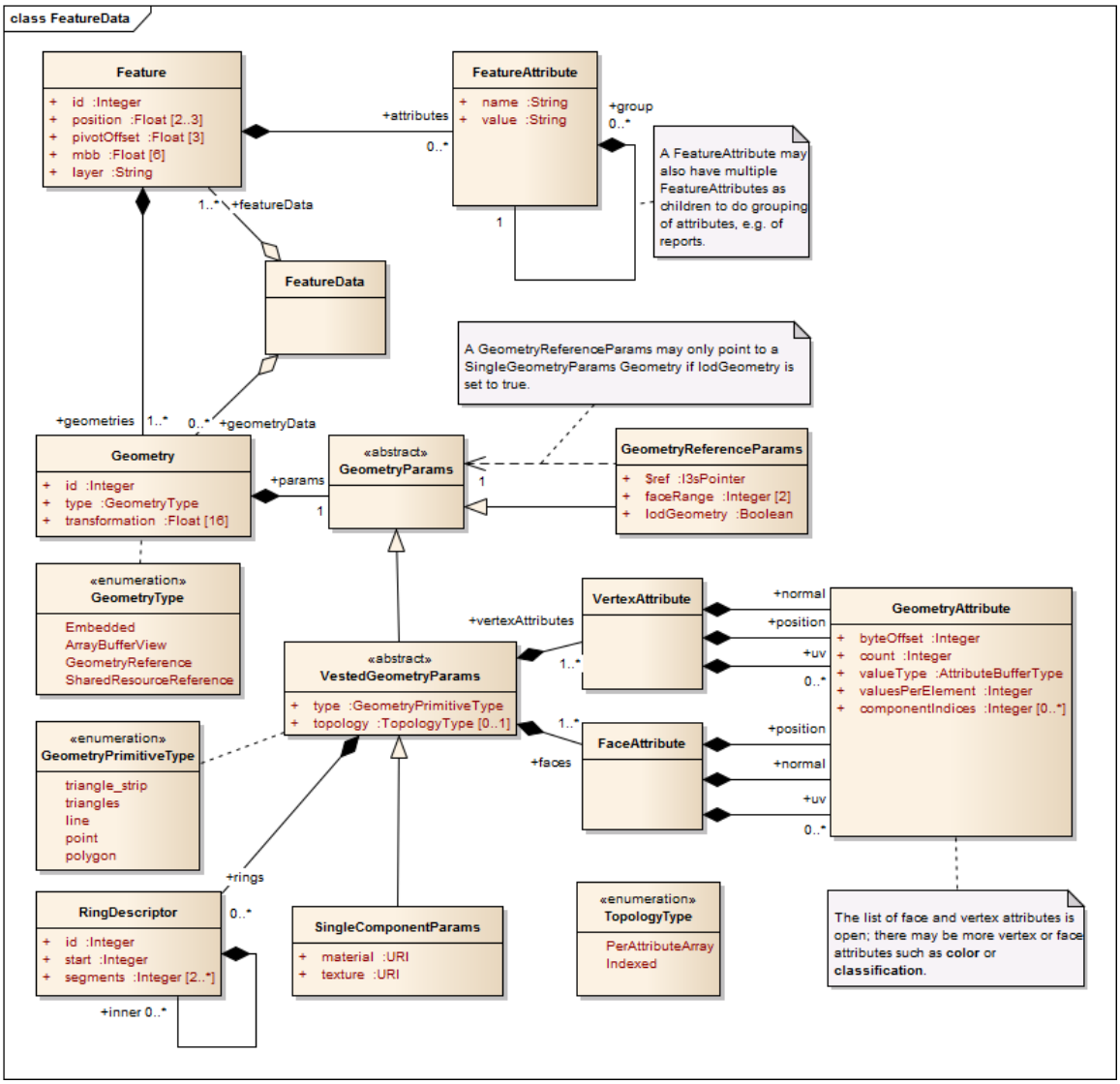

Figure 8: Logical schema of the FeatureData document

7.5.6.1 Class Feature

A Feature is a single object within a geospatial data set, usually representative of a feature present in the real, geographic world.

Name	Type	Description

id	Integer	Feature ID, unique within the Node. If `lodType` is `FeatureTree`, the ID SHALL be unique in the store.
position	Float[2..3]	An array of two or three doubles, giving the x,y(,z) (easting/northing/height) position of this feature's minimum bounding sphere center, in the vertexCRS.
pivotOffset	Float[3]	An array of three doubles, providing an optional, "semantic" pivot offset that can be used to, for example, correctly drape tree symbols.
mbb	Float[6]	An array of six doubles, corresponding to x_{min}, y_{min}, z_{min}, x_{max}, y_{max} and z_{max} of the minimum bounding box of the feature, expressed in the vertexCRS, without offset. The mbb can be used with the Feature's Transform to provide a LOD0 representation without loading the GeometryAttributes.
layer	String	The name of the Feature Class this feature belongs to.
attributes	FeatureAttribute[0..*]	The list of attributes for this feature.
geometries	Geometry[1..*]	The list of geometries the feature has. A feature always SHALL have at least one Geometry.

Table 24: Attributes of the Class *Feature* within the FeatureData document

7.5.6.2 Class FeatureAttribute

A FeatureAttribute is a field carrying a value. This value may also be a list of complete attributes, to be used with reports or metadata.

Name	Type	Description
name	String	The name of the attribute.
value	String	The value of the attribute. If group is set and the type of this attribute is set to FieldTypeGroup, the value may be used as a label.
group	FeatureAttribute[0..*]	A list of FeatureAttributes belonging to an attribute value group.

Table 25: Attributes of the Class *FeatureAttribute* within the FeatureData document

7.5.6.3 Class Geometry

This is the common container class for all types of I3S geometry definitions.

Name	Type	Description
id	Integer	Reference-able, unique ID of the Geometry in this store.
type	String	The type denotes whether the following geometry is

		defined by using array buffer views (ArrayBufferView), as an internal reference (GeometryReference), as a reference to a shared Resource (SharedResourceReference) or embedded (Embedded).
transformation	Float[16]	3D (4x4) transformation matrix expressed as a linear array of 16 values.
params	GeometryParams	The parameters for a geometry, as an Embedded GeometryParams object, an ArrayBufferView, a GeometryReference object, or a SharedResourceReference object.

Table 26: Attributes of the Class *Geometry* within the FeatureData document

7.5.6.4 Class GeometryParams

This is the abstract parent class for all GeometryParams classes (GeometryReferenceParams, VestedGeometryParamas, SingleComponentParams). It does not have properties of its own.

7.5.6.5 Class GeometryReferenceParams

Instead of owning a Geometry exclusively, a Feature can also reference a (or part of a) Geometry defined for the node. This allows to pre-aggregate Geometries for many features. In this case, a GeometryReferenceParams has to be used.

Name	Type	Description
$ref	Pointer	In-document absolute reference to full geometry definition (Embedded or ArrayBufferView) using the I3S json pointer syntax.
faceRange	Integer[2]	Inclusive range of faces in this geometry that belongs to this feature.
lodGeometry	Boolean	True if this geometry participates in a LoD tree. This value SHALL always be true for the mesh-pyramids profile.

Table 27: Attributes of the Class *GeometryReferenceParams* within the FeatureData document

7.5.6.6 Class VestedGeometryParams

This Class extends GeometryParams and is the abstract parent class for all concrete ("vested") GeometryParams classes that directly contain a Geometry definition, either as an ArrayBufferView or as an Embedded Geometry.

Name	Type	Description
type	String	The primitive type of the geometry defined through a VestedGeometryParams object. One of

		{*triangles*, lines, points}
topology	TopologyType	Declares the typology of embedded geometry attributes or those in a geometry resources. One of {"PerAttributeArray", "InterleavedArray", "Indexed"}. When "Indexed", the indices (faces) SHALL be declared.
vertexAttributes	VertexAttribute[1..*]	A list of Vertex Attributes, such as Position, Normals, UV coordinates, and their definitions. While there are standard keywords such as `position`, `uv0..uv9`, `normal` and `color`, this is an open, extendable list.
faces	FaceAttribute[0..*]	A list of Face Attributes, such as indices to build faces, and their definitions. While there are standard keywords such as `position`, `uv0..uv9`, `normal` and `color`, this is an open, extendable list.

Table 28: Attributes of the Class *VestedGeometryParams* within the FeatureData document

7.5.6.7 Class SingleComponentParams

Objects of this type extend VestedGeometryParams and use one texture and one material. They can be used with aggregated LoD geometries.

Name	Type	Description
material	URI	I3S Pointer reference to the material definition in this node's shared resource, from its root element. If present, used for the entire geometry.
texture	URI	I3S Pointer reference to the material definition in this node's shared resource, from its root element. If present, used for the entire geometry.

Table 29: Attributes of the Class *SingleComponentParams* within the FeatureData document

Component objects provide information on parts of the geometry they belong to, specifically with which material and texture to render them.

Name	Type	Description
id	Integer	The ID of the component, only unique within the Geometry
materialID	UUID	ID of the material, as defined in the shared resources bundle, to use for rendering this component
textureID	Long[0..1]	Optional ID of the texture, as defined in shared resources, to use with the material to render this component
regionID	Long[0..1]	Optional ID of a texture atlas region which to use with the texture to render this component

Table 30: Attributes of the Class *Component* within the FeatureData document

7.5.6.8 Class GeometryAttribute

Each GeometryAttribute object is an accessor, (a view) into an arraybuffer. There are two types of GeometryAttributes - VertexAttributes and FaceAttributes. While the first describes properties that are valid for a single vertex, the second is used to describe faces and other structures by providing a set of indices. As an example, the `faces.position` index attribute is used to define which vertex positions make up a face.

Name	Type	Description
byteOffset	Integer	The starting byte position where the required bytes begin. Only used with the Geometry `"type": "ArrayBufferView"`.
count	Integer	The number of elements. Multiply by number of bytes used for valueType to know how many bytes need to be read. Only used with the Geometry `"type": "ArrayBufferView"`.
valueType	String	The element type, from `{UInt8, UInt16, Int16, Int32, Int64 or Float32, Float64}`
valuesPerElement	short	The number of values need to make a valid element (such as 3 for a xyz position)
values	Float[*]	The actual values. Only used with the Geometry `"type": "Embedded"`
componentIndices	Integer[0...*]	An optional array that indicates how many of the elements in this view belong to the first, second and consecutive components of the geometry. The number of entries in this array, when present, has to be equal to the number of entries in the components List of the enclosing Geometry object. The entire field is optional when no components have been declared for this Geometry.

Table 31: Attributes of the Class *GeometryAttribute* within the FeatureData document

7.6 Shared Resources

Shared resources are models or textures that can be shared among features within the same layer. They are stored entirely as a JSON file. Each node has a shared resource which contains materials and symbols used by more than a single feature in that node or in features which are stored in the subtree of the current node. This approach ensures an optimal distribution of shared resources across nodes, while maintaining the node-based updating process.

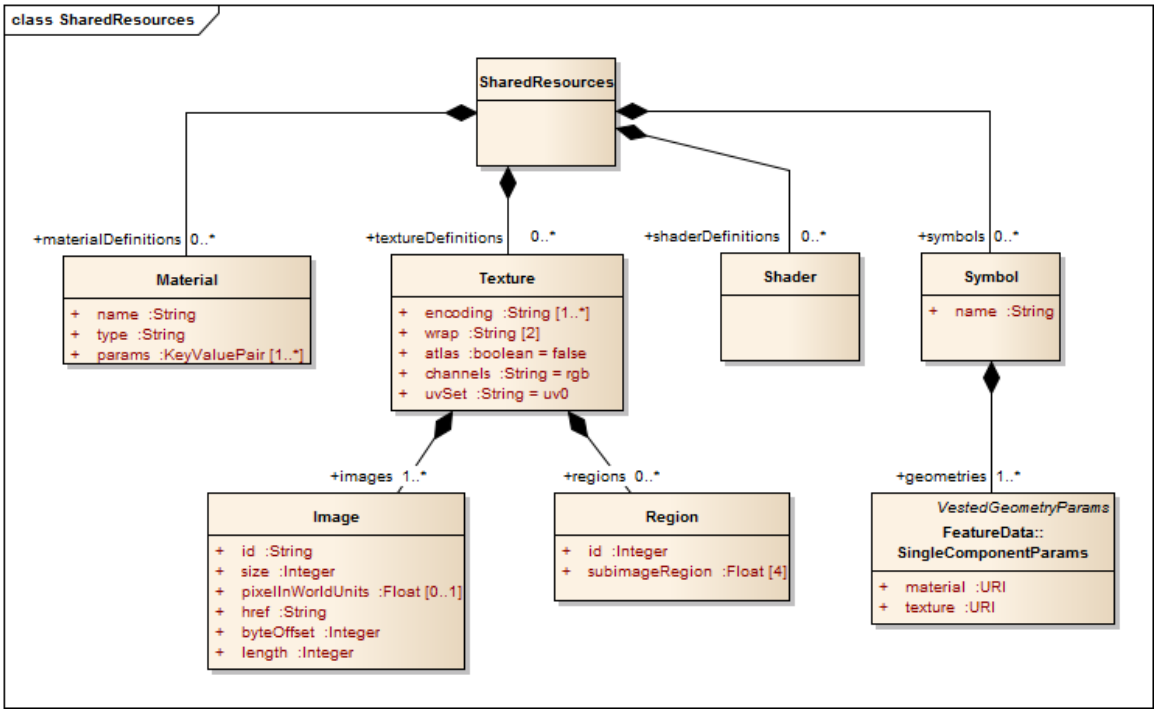

Figure 9: Logical schema of the SharedResources document

7.6.1 Class SharedResource

The `SharedResource` class collects Material definitions, Texture definitions, Shader definitions and geometry symbols that need to be instanced.

7.6.2 Class Material

Materials describe how a Feature or a set of Features is to be rendered. This includes which shading and which colors to use. The following table provides the set of attributes and params for the `"type": "standard"` material.

Name	Type	Description
name	String	A name for the material as assigned in the creating application.
type	String	Indicates the material type, chosen from the supported values `{standard, water, billboard, leafcard, reference}`
$ref	JSONPointer	The href that resolves to the shared resource bundle in which the material definition is contained.
params.vertexRegions	Boolean[0..1]	Indicates whether this Material uses per-vertex regions. Defaults to `false`.
params.vertexColors	Boolean[0..1]	Indicates whether this Material use Vertex

		Colors. Defaults to `false`.
params.useVertexColorAlpha	Boolean[0..1]	Indicates whether Vertex Colors also contain a transparency channel. Defaults to `false`.
params.transparency	Float	Indicates whether the transparency of this material; 0 = opaque, 1 = fully transparent.
params.reflectivity	Float	Indicates reflectivity of this Material.
params.shininess	Float	Indicates shininess of this Material.
params.ambient	Float[3]	Ambient color of this Material.
params.diffuse	Float[3]	Diffuse color of this Material.
params.specular	Float[3]	Specular color of this Material.
params.renderMode	String	Rendering mode, any one of `{textured, solid, untextured, wireframe}`
params.castShadows	Boolean	`true` if features with this material should cast shadows
params.receiveShadows	Boolean	`true` if features with this material should receive shadows
params.cullFace	String	Indicates the material culling options {back, front, *none*}. Default being `none`.

Table 32: Attributes of the Class *Material* within the SharedResources document

7.6.3 Class Texture

A Texture is a set of images, with some parameters specific to the texture/uv mapping[28] to geometries.

Name	Type	Description
encoding	MIMEtype[1..*]	The encoding/content type that is used by all images in this map
wrap	String[2]	UV wrapping modes, from `{none, repeat, mirror}`
atlas	Boolean	true if the Map represents a texture atlas.
uvSet	String	The name of the UV set to be used as texture coordinates.
channels	String[1..*]	indicates which channels are stored in which channel of this map. Possible values: h=brightness, r=red, g=green, b=blue, a=alpha, n=bump, d=displacement, ...

Table 33: Attributes of the Class *Texture* within the SharedResources document

[28] https://en.wikipedia.org/wiki/UV_mapping for more information.

7.6.4 Class Image

An image is a binary resource, containing a single raster that can be used to texture a feature or symbol. It represents one specific texture LoD. For details on texture organization, please refer to the section on Texture resources.

Name	Type	Description
id	String	A unique ID for each image. Generated using the BuildID function.
size	Integer	x size of this image.
pixelInWorldUnits	Float	maximum size of a single pixel in world units (used by the renderer to pick the image to load/map)
href	URL[1..*]	The href to the image(s), one per encoding, in the same order as the encodings.
byteOffset	Integer[0..*]	The byte offset of this image's encodings (one per encoding, in the same order as the encodings.) in the block in which this texture image resides.
length	Integer[0..*]	The length in bytes of this image's encodings (one per encoding, in the same order as the encodings).

Table 34: Attributes of the Class Image within the SharedResources document

7.6.5 Class ShaderDefinition

ShaderDefinitions are, in this version of the I3S standard, an optional feature to provide API-dependent shader programs with a layer.

7.6.6 Class Symbol

For Symbols, the same model is used as in the FeatureData Geometry.

8. I3S File Formats

8.1 Textures.bin

The Textures file is a binary resource that contains one or multiple images that are used as textures of features in the store. A single Texture.bin file contains 1...n textures for a single specific texture LoD. It can contain a single texture atlas or multiple individual textures; the decision how this is bundled is left to the authoring application so that specific aspects of the materials and textures used can be taken into account, such as tiling.

8.1.1 Texture Recommendations and Requirements

Especially for Web and Mobile clients, the number of textures and their volume is the limiting factor in how much data can be displayed at any given time, Thus, this standard

provides several recommendations and requirements on texture resources that are delivered as part of an Indexed 3D Scene.

8.1.2 Image Formats

I3S supports multiple texture formats which are suitable for different scenarios. For example, a client application might prefer consuming the more compact JPEG (and/or PNG) formats over low bandwidth conditions since they are very efficient to transmit and have a widespread adoption. However, client applications that might be constrained for memory or computing resource might prefer to directly consume compressed textures such as S3TC for scalability and performance reasons. As a result, the I3S standard supports most commonly used image formats such as JPEG/PNG as well as rendering optimized compressed texture formats such as S3TC. The only requirement is the authoring application needs to provide the appropriate `textureEncoding` declaration by using MIME types such as, "image/jpeg" (for Jpeg) and "image/vnd-ms.dds" (for S3TC). With more wide-spread client support for next-generation texture compression formats such as Adaptive Scalable Texture Compression (ASTC), Ericsson Texture Compression v2 (ETC2), and PVRTC[29], I3S will include support for more compressed texture formats in the future to enable specific platforms.

8.1.3 Texture Sets

While this standard allows the combination of multiple textures into a single resource by using array buffer views, we generally recommend using large atlases (e.g. 2048x2048px) and then to use exactly one texture per bundle.

8.1.4 Atlas usage and Regions

Individual textures should be aggregated into texture atlases, where they become subtextures. Just as all texture resources, the atlas has to be 2^n-sized on both dimensions, with n being in the range [3,12]. Width and height dimensions do not have to be equal, e.g. 512px x 256px. Subtextures contained within an atlas also need to be 2^n-sized, with n being in the range [3,12]. Otherwise if their width or height dimension is not 2^n, border artifacts are likely to appear when filtering or MIP-mapping. If source subtexture dimensions do not match this requirement, they need to be padded (with nearest/interpolated pixels) or scaled to the nearest lower 2^n size. An image that is 140px x 90px would thus be rescaled to 128px x 64px before being inserted into the atlas or padded to 256px x 128px.

The pixels belonging to a subtexture are identified by the `subimageRegion: [umin, vmin, umax, vmax]` attribute. Region information is passed on to the shader using a separate vertex attribute so that every vertex UV coordinate becomes a UVR coordinate, with the R encoding the `[umin, vmin, umax, vmax]` of the region in 4 `UInt16` values.

[29] https://en.wikipedia.org/wiki/PVRTC

8.1.5 Texture coordinates

Texture coordinates do not directly take atlas regions into account. They always range from `0...1` in U and V, except when using the "repeat" wrapping mode, where they may range from `0...n` (n being the number of repeats). The client is expected to use the `subimageRegion` values and the texture coordinates to best handle repeating textures in atlases. This approach has been selected since client capabilities in dealing with more complex UV cases vary greatly.

8.1.6 Generating Image IDs

The Id of an image is generated using the following method:

```
UInt64 BuildID(LONG id, int w, int h , int l, int al)
{
    UInt64 l_al = ((UInt64)al)<<60;
    UInt64 l_l = ((UInt64)l)<<56;
    UInt64 l_w = ((UInt64)(w - 1))<<44;
    UInt64 l_h = ((UInt64)(h - 1))<<32;
    UInt64 id64 = l_al + l_l + l_w + l_h + (UInt64)id;
    return id64;
}
```

Usage syntax:

```
UInt64 image_id = BuildID(id, w, h, l, al);
```

8.1.6.1 Function Parameters

id	Index of the texture in the store, start from 1
w	Width of the texture
h	Height of the texture
l	Index of the level that the texture belong to, start from 0
al	Level count of the texture

8.2 Geometry.bin

The binary geometry attribute file follows the Khronos[30] Typed Array specification[31] in the ECMAScript® 2015 Language Specification. Citing the overview of that specification:

[30] Khronos is a not for profit, member-funded consortium dedicated to the creation of royalty-free open standards for graphics, parallel computing, vision processing, and dynamic media on a wide variety of platforms from the desktop to embedded and safety critical devices.

"This specification defines an ArrayBuffer type, representing a generic fixed-length binary buffer. The contents of an ArrayBuffer cannot be directly manipulated. Instead, a group of types are used to create views of the ArrayBuffer. For example, to access the buffer as an array of 32-bit signed integers, an Int32Array would be created that refers to the ArrayBuffer.

Multiple typed array views can refer to the same ArrayBuffer, of different types, lengths, and offsets. This allows for complex data structures to be built up in the ArrayBuffer. As an example, given the following code:

```
// create an 8-byte ArrayBuffer
var b = new ArrayBuffer(8);

// create a view v1 referring to b, of type Int32, starting at
// the default byte index (0) and extending until the end of the buffer
var v1 = new Int32Array(b);

// create a view v2 referring to b, of type Uint8, starting at
// byte index 2 and extending until the end of the buffer
var v2 = new Uint8Array(b, 2);

// create a view v3 referring to b, of type Int16, starting at
// byte index 2 and having a length of 2
var v3 = new Int16Array(b, 2, 2);
```

This defines an 8-byte buffer b, and three views of that buffer, v1, v2, and v3. Each of the views refers to the same buffer -- so v1[0] refers to bytes 0..3 as a signed 32-bit integer, v2[0] refers to byte 2 as a unsigned 8-bit integer, and v3[0] refers to bytes 2..3 as a signed 16-bit integer. Any modification to one view is immediately visible in the other: for example, after v2[0] = 0xff; v2[1] = 0xff; then v3[0] == -1 (where -1 is represented as 0xffff)."

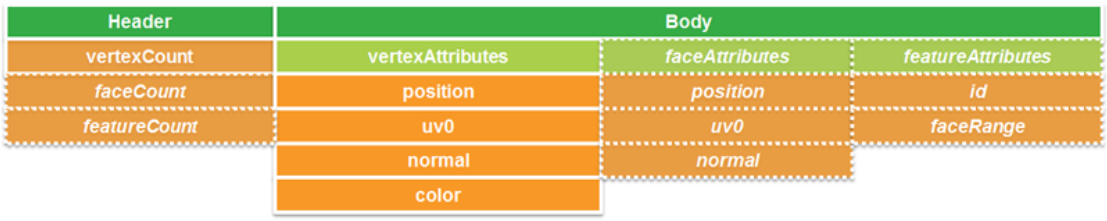

Figure 10: Geometry Buffer Layout with headers

Note: The expected triangle/face winding order in all geometry resources is counterclockwise (CCW).

[31] http://www.ecma-international.org/ecma-262/6.0/#sec-typedarray-objects and http://www.ecma-international.org/ecma-262/6.0/#sec-arraybuffer-objects

Note: If normal vectors are present in a geometry, they need to be calculated based on uniform axis units. They are always given as if x, y and z axes all had metric units, as a unit vector. This means that if WGS84 is used as a horizontal CRS, the normal calculation cannot directly use the face's WGS84 coordinates, but needs to convert them to a local Cartesian CRS first.

8.3 Attribute Data

This section describes the format for storing attribute data within I3S layers as part of the scene service cache along with geometry, texture and material resources, in an optimized renderer friendly format.

By attribute data we mean the tabular information stored as an attribute of a feature class, which is the primary input source of scene services.

Attribute data for all features in a node is stored and made available as discrete, per field resource called ***attribute***. The number of attribute resources corresponds to the number of fields the service publisher opted to include in the scene cache.

A key concept of this storage model is that the order in which attribute values are stored within any *attribute* resource SHALL be the same as the order in which the feature geometries are stored within the geometry resource of that node. This allows clients who fetch these resources to render each node efficiently - using direct array access to retrieve feature attribute(s) without the need for object-id based attribute lookups.

For cases where object-id based access to attributes is needed, the *attribute* resource representing the *object-id* field stores the object-id values of each feature within the node and SHALL use the same storage order as the geometry resource. This facilitates object-id based access. Clients can also build an object-id to array-index dictionary for cases where large numbers of object-id based attribute or geometry look ups within a node are needed. (Note: the following ways of referring to the ObjectId of a feature are equivalent in these and previous versions of the I3S specification: ObjectId, object-id, OID, FID).

When the same feature is included in multiple nodes at different levels of detail, the corresponding attributes for the feature are also included as *attribute* resource/s of each node it is present in. This redundancy in attribute storage allows each node to be rendered independently of any other node.

Metadata on each *attribute* resource is made available to clients via the scene service layer. When attributes are present within the scene cache, the *resourcePattern* array in the layers store (layers[id].store.resourcePattern) will include a value called *Attributes*, indicating attributes are a required resource, utilized for attribute driven symbolization and rendering. In addition to the *resourcePattern*, additional metadata present in the *fields* array (*layers[id].fields[id]*) and *attributeStorageInfo* array (*layers[id].attributeStorageInfo[id]*), further describe each attribute resource.

These metadata allow clients to initialize and allocate any required client side resources prior to accessing any attributes of interest.


```
fields: [
 - {
        name: "OID",
        type: "FieldTypeOID",
        alias: "OBJCTID"
   },
 - {
        name: "NEAR_FID",
        type: "FieldTypeInteger",
        alias: "NearestFeature"
   },
 - {
        name: "NEAR_DIST",
        type: "FieldDoubleType",
        alias: "NearestFeatureDistance"
   },
 - {
        name: "NAME",
        type: "FieldTypeString",
        alias: "Name"
   },
 - {
        name: "Building_ID",
        type: "FieldTypeInteger",
        alias: "BuildingID"
   }
],
```


Figure 11: Example of the *fields* array resource

Detail: The above is an example of the fields array (layers[id].fields[id]) resource of a scene service layer illustrating different supported types of feature attribute fields. The fields array describes an attribute field with respect to its name, type and alias.

Once a client application makes a decision regarding the field it is interested in accessing, it can use the *key* property (*layers[id].attributeStorageInfo[].key*) of the *attributeStorageInfo* metadata to uniquely identify and request the *attribute* resource thru an API, called **attributes.** The *attributeStorageInfo* metadata in addition contains all the information that a client application requires to decode the retrieved *attribute* binary content.t

8.3.1 The content of this binary attribute resource is made up of:

- A header section of 4 bytes which indicates the count of features. The count value SHALL be present in all *attribute* resources. For an *attribute* resource of a string data type, the header has an additional 4 bytes indicating the total byte count of the string attribute values.
- For all numerical field types, the header section SHALL be followed by the attribute values array record. The attribute values SHALL always begin at an

offset that is divisible by the byte length of a single value. If the header does not end at such an offset, the necessary amount of padding SHALL be inserted between the header and the attribute values.

- For string field types, the header section SHALL be followed by a fixed length array whose values are the byte counts of each string data, inclusive of the null termination character. This array SHALL then followed by an array of actual string data. The strings SHALL be stored null terminated.

An example JSON encoding for a 3dSceneLayer mesh pyramid can be found at Annex B. *This is a scene layer resource illustrating the metadata information found in the fields (layers[id].fields[id]) and attributeStorageInfo arrays (layers[id].attributeStorageInfo[id]).*

A client application will be able to find the URI of any attribute resource through its href reference from the **attributeData** array of the **Node Index Document** (similar access patterns exist for resources such as 'features', 'geometries', etc …). See Figure 12 below:

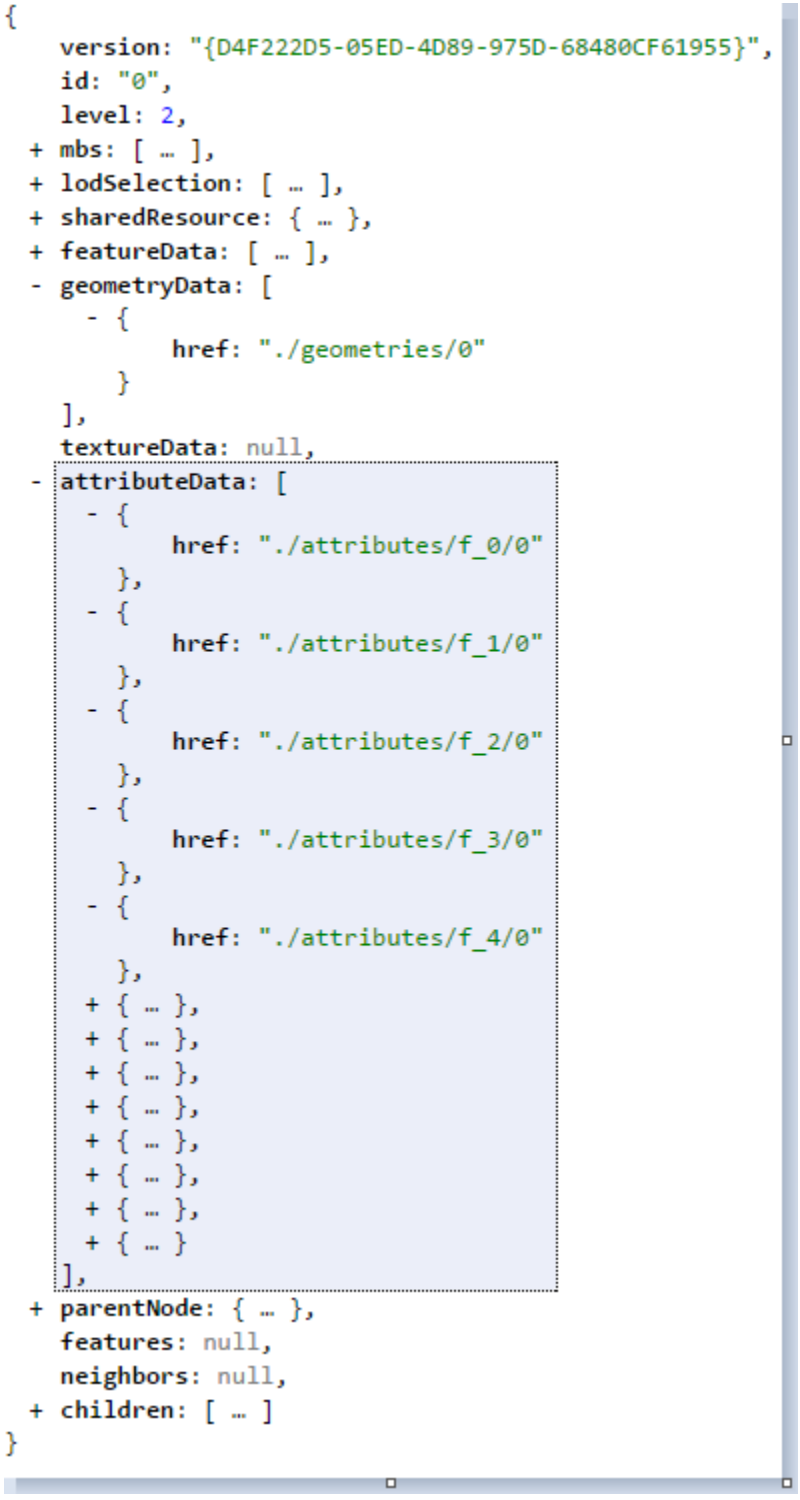

Figure 12: A node resource document

Detail: A node resource document illustrating attribute data content access URLs (href).

8.3.2 REST API for Accessing Attribute Resources directly from a scene service layer

This section describes a REST API for accessing attribute resources. The **attributes** REST API allows client apps to fetch the attribute records of a given field as identified by its *Key* property. As a result, every scene node (with the exception of 'root' node), will expose available attribute fields as discrete *attribute* resources. These resources are accessible thru a relative URL to any Node Index Document.

The *attributes* REST api syntax is as follows:

URL: **http://<sceneservrice-url>/attributes/<field_key>/<id>**

- *attributes* - is the RESTful resource responsible for fetching the binary attribute resource. A client application will be able to decode the content of this *attribute* resource solely based on the metadata information found in the scene layer *attributeStorageInfo* array (which adequately describes the content of the binary data).
- *field_key* - is the field key value that will be used to request the desired feature attribute content.
- *id* - is the bundle id of the *attribute* binary resource, corresponding to the geometry bundle id. By default this value is 0 (same as the geometry bundle id). If a node has more than 1 geometry resource, then the id of the *attribute* resource SHALL match the geometry bundle id.

8.3.3 A typical pattern of usage of the attributes REST API includes

1. Prior to symbolizing a given node based on attribute information, a client application should get attribute field metadata information by fetching the scene server *layers* REST resource. The *layers* resource contain the *fields* (*layers[Id].Fields[id]*) array, which lists all available attribute fields and types and the *attributeStorageInfo* (layers[id].attributeStorageInfo[id]) array, which describes the content of each binary *attribute* resource.

   The *fields* array object contains a collection of objects that describe each attribute field regarding its field name ('name'), datatype ('type') and a user friendly name ('alias'). The *fields* array includes all fields that are present in the source feature layer of the scene service layer.

   The *attributeStorageInfo* array contains a collection of objects that describes all *attribute* binary resources. It includes only fields the publisher/author chose to include as part of the scene cache during publishing time. The *attributeStorageInfo*, which is metadata information about the binary *attribute* resources, is made up of:

i. name (attributeStorageInfo[id].name) and key (attributeStorageInfo[id].key) properties that identify each resource.

ii. A *header* (*attributeStorageInfo[id].header*) object, consisting of a *count* and *valueType* properties indicating the count of the attributeValue objects. In case of string atttibute values the *header* consists an additional object, *attributeByteCounts* property, which indicates the total byte count of the string values.

iii. An *ordering* (*attributeStorageInfo[id].ordering*) object that indicates the object storage layout.

iv. For string attribute values, an *attributeByteCounts* object describing each of the string attribute values byte count.

v. The *attributeValues* object describing the attribute value array, which contains member properties such as *valueType* and *valuesPerElement*. For string attribute values in addition to its *valueType* ('String'), there is an additional property *encoding* ('UTF-8') that indicates the unicode enconding type. A String-Array is capable of supporting null attribute values (a 0 byte count value indicates a null string).

Note that the *key* property (with values such as *f_0*, *f_1*, etc...) is automatically computed and that there shouldn't be any relationship *assumed* to the field index of the source feature class (especially important when a user adds or deletes fields during the lifetime of a layer).

```
attributeStorageInfo: [
  - {
        name: "OID",
        key: "f_0",
      - header: [
          - {
                property: "count",
                valueType: "UInt32"
            }
        ],
      - ordering: [
            "ObjectIds"
        ],
      - objectIds: {
            valueType: "UInt32",
            valuesPerElement: 1
        }
    },
  - {
        name: "NEAR_FID",
        key: "f_1",
      - header: [
          - {
                property: "count",
                valueType: "UInt32"
            }
        ],
      - ordering: [
            "attributeValues"
        ],
      - attributeValues: [
          - {
                valueType: "Int32",
                valuesPerElement: 1
            }
        ]
    },
  - {
        name: "NEAR_DIST",
        key: "f_2",
      - header: [
          - {
                property: "count",
                valueType: "UInt32"
            }
        ],
      - ordering: [
            "attributeValues"
        ],
      - attributeValues: {
            valueType: "Float64",
            valuesPerElement: 1
        }
    },
  - {
        name: "Name",
        key: "f_3",
      - header: [
          - {
                property: "count",
                valueType: "UInt32"
            },
          - {
                property: "attributeValuesByteCount",
                valueType: "UInt32"
            }
        ],
      - ordering: [
            "attributeOffsets",
            "attributeValues"
        ],
      - attributeOffsets: {
            valueType: "UInt32",
            valuesPerElement: 1
        },
      - attributeValues: {
            valueType: "string",
            encoding: "UTF-8",
            valuesPerElement: 1
        }
    },
  - {
        name: "Building_ID",
        key: "f_4",
      - header: [
          - {
                property: "count",
                valueType: "UInt32"
            }
        ],
      - ordering: [
            "attributeValues"
        ],
      - attributesValues: {
            valueType: "Int32",
            valuesPerElement: 1
        }
    }
]
```

Figure 13: An expanded view of a scene layer resource

More detail: The above is an expanded view of a scene layer resource showing the content of an attributeStorageInfo resource. The example shows 5 objects each corresponding to the 5 objects of the fields *resource (as matched by the 'key' and 'name' properties present in both arrays).The JSON representation of the example is located in Annex B.*

2. A client application equipped with the list of available fields and the corresponding attribute-value-array metadata, can then fetch the attribute values of interest just by supplying the desired field *Key* as part of the **attributes** REST request. Furthermore, the client will also be capable of decoding the fetched *attribute* resource based on the metadata as retrieved in step 1.

   Note: The geometry buffer contains the *objectIDs* array as the last section of the geometry layout (layers[id].store.defaultGeometrySchema.featureAttributes). A client application that has a need to access the *ObjectIDs* array, should first check in the geometry buffer before requesting it from the *attributes* REST resource.

   The following example below shows the *attributes* REST request signature:

   a. http://<myserver>/<my product>/rest/services/Hosted/SanFran/SceneServer/layers/0/nodes/0-0-0-0/**attributes/0/f_1**
   b. http://<myserver>/<my product>/rest/services/Hosted/SanFran/SceneServer/layers/0/nodes/0-0-0-/**attributes/0/f_2**

   In *Example 1.a* we are requesting the attributes of all features for a *field* named 'NEAR_FID', as identified by its field key (*f_1*) in Figure 11. This field resource contains the attribute values of all *features* that are present in node 0-0-0-0. Similarly, *Example 1.b* will fetch the attributes of all features associated with the field called ('NEAR_DIST') using its key (*f_2*).

8.3.4 Attribute Resources: Details

An *attribute* resource consists of either a single one dimensional array in the case of numeric fields (including the object-id field) or two one dimensional arrays that sequentially follow each other in the case of variable length string fields.

The structure of each *attribute* resource is declared upfront in the scene layer resource thru the *attributeStorageInfo* object. The client application (as stated above in the typical usage pattern) is expected to read the *attributeStorageInfo* metadata to get the header information, the ordering of the stored records (arrays) as well as their value types before consuming the binary attribute resource.

Consider a sample scene service layer and its field types (see Figure 14). This layer has 6 fields named 'OID', 'Shape', 'NEAR_FID', 'NEAR_DIST', 'Name' and 'Building_ID'.

OID	Shape	NEAR_FID	NEAR_DIST	Name	Building_ID
1	MultiPatch	0	-1	Front	1044
3	MultiPatch	1	0.786088	back	1045
5	MultiPatch	2	0.786416	side	1046

Figure 14: A typical attribute (table) info of a feature class

More detail: A typical attribute (table) info of a feature class. The fields array that's shown as an example in Figure 11 and the attributeStorageInfo array in Figure 13 is derived from the attribute value of the above feature class.

As it could be inferred from Figure 11 and Figure 13, a scene service layer exposes/includes **only** supported attribute field value types of a feature class. As a result, the 'Shape' field (see Figure 14), which is of *FieldTypeGeometry* type, will not be included in the attribute cache of a scene layer.

Table 35 below lists a feature layer's field data types (including its values and description). The I3S valueTypes column indicates the value types of the fields that are supported for attribute based mapping/symbology.

Feature Data Field Type Constants	Value	Description	I3S ValueTypes
FieldTypeSmallInteger	0	Short Integer	Int16
FieldTypeInteger	1	Long Integer	Int32
FieldTypeSingle	2	Single Precision floating point number	Float32
FieldTypeDouble	3	Double Precision floating point number	Float64
FieldTypeString	4	Character String	String*
FieldTypeDate	5	Date	string
FieldTypeOID	6	Long Integer representing object ID	UInt32
FieldTypeGUID	10	Globally Unique Identifier	string
FieldTypeGlobalID	11	Global ID	string
FieldTypeXML	12	XML Document	string

Table 35: Attribute data types supported by a scene service layer.

* String – using UTF-8 Unicode character encoding scheme.

The following types of attribute value arrays are supported : *Int32-Array*, *UInt32-Array*, *UInt64-Array*, *Float64-Array*, *Float32-Array*, *String-Array*

Using our example feature class shown in Figure 14 let's see how it maps to the different types of *attribute* resources.

The *'OID'* field, whose field type is 'FieldTypeOID' is by default represented as an *UInt32-Array*. This is a simple 1-d array of *UInt32* value type.

The next attribute field type in the above example, 'NEAR-FID' which is of field type 'FieldTypeInteger' is represented as an *Int32-Array*. This again is also a simple 1-d array of *Int32* value type.

The 'NEAR_DIST' field is of field type 'FieldTypeDouble' field type and is represented as a *Double-Array*, represented as 1-d array of *Float64* value type.

The 'Name' field is of 'FieldTypeString' and is represented as a *String-Array*. A String-Array supports storage of variable length strings and is stored as two arrays in sequence where the first fixed length array has the byte counts of each string (null terminated) in the second array and the second array stores the actual string values as UTF8 encoded strings. The value type of the first array is (*UInt32*) whereas the value type of the second array is *String*.

The *attributes* REST API of a scene layer gives access to all scene cache operations supported feature attribute data as attribute value arrays that are stored in binary format. As a result, the scene cache of the example feature class in Figure 14 will have 5 binary resources, as identified by keys *f_0_, f_1_, f_2_, f_3_* and *f_4* and accessible by their respective rest resource URLs (_.../nodes/<nodeID>/attributes/0/f_0, .../nodes/<nodeID>/attributes/0/f_1, etc..).

8.4 Accessing the Legend of a 3D Object Layer

Legends are essential for proper display (complete communication of represented information) of 3D Object Layer (also equally applicable for other layer types).

Clients are responsible for building legend information from the drawingInfo resource for the scene layer. In this scene layers and scene services behave identically to feature layers and feature services.

9. Additional Informative Information

9.1 Flexibility

I3S is flexible and allows for different implementation choices for different types of 3D data or even for the same type of 3D data. The profile for a layer indicates the set of choices made. Choices supported by I3S and made use of by different profiles are described below. In each case the profile listed is the canonical profile for the corresponding layer-type. Consider the following:

1. The Minimum Bounding Volume (MBV) property can be represented as:
   a. Minimum Bounding Sphere (MBS)
   b. Oriented Bounding Box (OBB)
2. Node structure
   a. Expanded – in support of clients that want to gain more complete meta-

information about node's position within BVH topology and its immediate neighborhood

- Each index node provides pointers to its parent, all its children, and sibling neighbors (including their MBVs)
  Used by: mesh-pyramids and points profiles

b. fixed-size in support of paged access pattern

- A minimal structure – just the essentials: bounding volume; first-child reference; child-count; LoD selection data; etc..

3. Embedded versus Binary geometry content format
   a. Embedded geometry: as text (JSON) in-lined with other metadata within featureData resource. This supports profiles where run-length encoding of feature-IDs along the vertex data is suboptimal Used by: the canonical points profile.

   b. Binary format: for voluminous, ready to render/use geometries and cached attributes. Both typed array buffer views as well as fixed format binary buffers are supported.

   - The mesh-pyramids profile uses 'array buffer views' (ArrayBufferView follows the Khronos Typed Array specification)

4. LoD Selection based on different metricTypes:

   - maxScreenThreshold – LoD switching based on screen 'size' of the node's MBV. Used by: mesh-pyramids profile
   - screenSpaceRelative – LoD switching based on screen 'scale' of the node's MBV. Used by: points profile
   - distancRangeFromDefaultCamera – LoD switching based on normalized distance of the node's MBV from the camera. Used by: points profile

9.2 Summary of I3S Defining Characteristics

In summary, here are other characteristics, including content data formats, which the scene layer may include:

• Attributes may be included on individual entities, points, or on partial segments of meshes
• Attribute-based stylization may be modified by client software
• Multiple, alternative textures may be provided to optimize for per-platform performance and display
• JSON format for index and metadata, binary for more voluminous geometry, texture and attribute data
• A Scene Layer Package format for distribution, or direct use, of the scene layer as a single file (see SLPK section)
• Optional paired services that expose query-able and updatable RESTful

endpoints that enable direct access to dynamic source data
• Explicit control over bounding index shape and per-node switching behavior to provide for optimized display and query
• BVH based on bounding spheres (MBS) as well as oriented bounding boxes (OBB) (planned)
• Scene layers may be created in Cartesian 3D or in global 3D world coordinate reference systems

10. Persistence

I3S scene layers can be delivered to web, mobile and desktop clients using a number of different patterns. Most users will interact with scene layers using applications that access cloud or server based information via RESTful interfaces/services. In these cases the cache (the I3S nodes and their payloads) for the scene layer resides on the server and is returned to clients via a RESTful interface that exposes the scene layer, its nodes and their associated resources (geometries, attributes, textures) as web addressable resources. The I3S standard contains a complete description of the web addressable resources and their URL scheme. Some users will also interact with a scene layer delivered to them as a single large Scene Layer Package – this is a single file that packages the complete node tree and its resources into an archive that supports direct access to the individual nodes and resources within it. Scene Layer Packages (SLPK files) are part of the current I3S implementation with multiple generators and the ability by clients to consume packages containing hundreds of Giga-Bytes of content.

All storage methods store the Indexed 3D Scene Layers in a simple key-value structure, with the key representing the access URL and the value being the JSON document or other resource type.

10.1 Scene Layer Packages

Scene Layer Packages (SLPK) serve two purposes: First, they allow a complete I3S layer, with all resources, to be transported or exchanged as a single file. Second they optionally also allow direct consumed by applications such as clients or services.

The format of the package itself is defined as follows:

- The Archive type is always Zip64[32].
- On this Archive, an overall compression scheme may be applied. This compression scheme SHALL be either STORE or DEFLATE64. Standard DEFLATE is acceptable as a fallback if DEFLATE64 is not available, but will only work with smaller SLPKs.

[3232] https://en.wikipedia.org/wiki/Zip_(file_format)

- STORE is the preferred compression schema for an SLPK intended for direct consumption by client application, especially if a resource compression is already applied on the individual resources (as shown in the figure 15 below).
- Every resource except textures may also be individually compressed. Compressed textures (such as S3TC) can additionally have GZIP[33] compression applied to them.
- For resource compression, only the GZIP scheme is supported, as DEFLATE support is not universally available in all browsers.

The layout show in Figure 15 below is referred to as the BASIC folder pattern. The I3S standard allows also for an EXTENDED folder pattern that uses subtree partitions to avoid problems with very large packages.

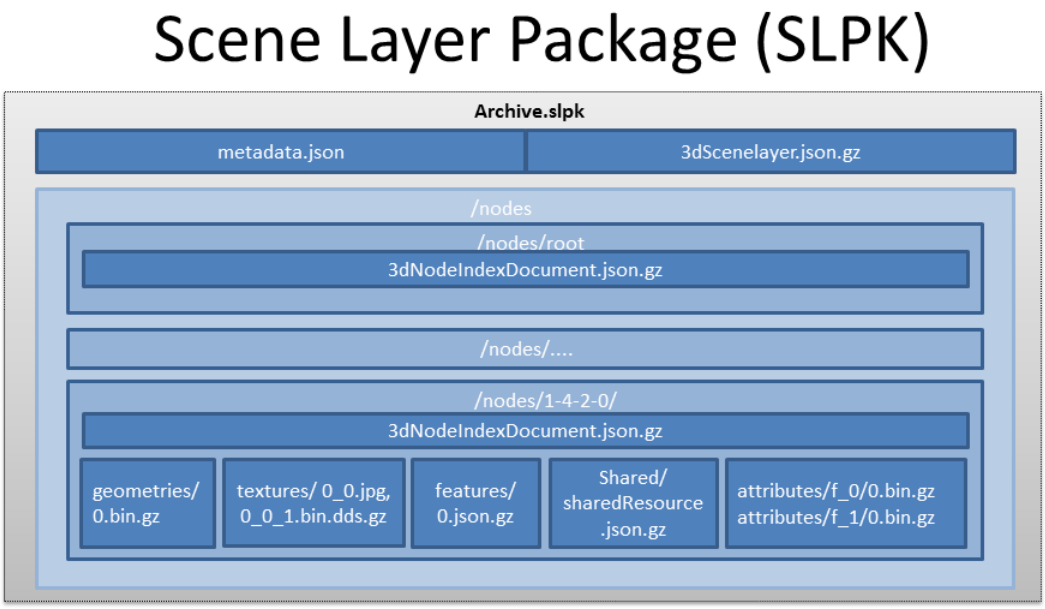

Figure 15: Example of a SLPK with BASIC folder layout

The contents of the archive depicted in Figure 15 shows an SLPK with the BASIC folder pattern. At the top level, it has a *nodes* subfolder containing all node resources, a *metadata.json* file that describes the content of the SLPK and a *3dSceneLayer.json.gz* file that defines the Scene Layer. In the example, the *nodes* subfolder contains, nodes named *root*, *1-4-2-0*, etc. Drilling further into one of the nodes, *1-4-2-0*, notice that all file

[33] https://en.wikipedia.org/wiki/Gzip

resources are individually compressed with GZIP compression (indicated by the file extension .gz), with the exception of the texture resource that is in JPEG format (*textures/0_0.bin*). The resources under the sub folders *geometries* (*geometries/0.bin.gz*) and *attributes* (*attributes/f_0/bin.gz*, *attributes/f_1/bin.gz*, ...), serialized as binary, correspond to the geometryData, and attributeData resources of a scene layer, respectively. Similarly, *3dNodeIndexDocument.json.gz*, *features/0.json.gz* and *SharedResource.json.gz* correspond to 3dNodeIndexDocument, featureData and SharedResource documents of the Scene Layer, respectively, and are encoded in JSON and are also stored with a GZIP compression.

For the above mentioned two use cases, an SLPK file is employed as follows:

1. SLPK as a transfer format:
    1. ArchiveCompressionType: DEFLATE64
    2. ResourceCompressionType: NONE
2. SLPK as a serving format:
    1. ArchiveCompressionType: STORE
    2. ResourceCompressionType: GZIP

10.1.1 Metadata

The following entries are permitted in the Metadata.json file that is part of every SLPK archive:

Property	Required	Notes
folderPattern	True	One of {*BASIC*, EXTENDED}
ArchiveCompressionType	True	One of {*STORE*, DEFLATE64[,DEFLATE]}
ResourceCompressionType	True	One of {NONE, *GZIP*}
I3SVersion	True	One of {1.2, 1.3, 1.4, *1.6*}
nodeCount	True	Total number of nodes stored in this SLPK.

10.2 Key Value Stores

In this persistence schema, all scene layer resources are stored within either key value based cloud blob stores such as Windows Azure Blob Storage or Amazon Simple Storage (S3) or within more general key value stores. In the case of cloud blob stores, layer resources are stored as either simple objects within containing buckets (S3) or blobs within blob containers (Azure). In all cases each resource within a scene layer is identified by a unique key.

I3S Resources	Optional	Notes
/SceneServer	False	The SceneServiceInfo JSON that defines the service name

		and list the layers offered by this Scene Service {content type: text/plain, content encoding {NONE, *GZIP*}}
/SceneServer/layers/0	False	The 3dSceneLayer JSON resource. The layer id (e.g. `0`) is used as the key of the document {content type: text/plain, content encoding {NONE, *GZIP*}}
/SceneServer/layers/0/nodes/root	False	The 3dNodeIndexDocument of the layer as a JSON resource. The node id (e.g. `root`) is used as the key of the document {content type: text/plain, content encoding: {NONE, *GZIP*}}
/SceneServer/layers/0/nodes/0	False	The 3dNodeIndexDocument of the layer as a JSON resource. The node id (e.g. `0`) is used as the key of the document {content type: text/plain, content encoding: {NONE, *GZIP*}}
/SceneServer/layers/0/nodes/0/shared	False	The SharedResource of the node as a JSON resource. The keyword `shared` is used as the key of the document {content type: text/plain, content encoding {NONE, *GZIP*}}
/SceneServer/layers/0/nodes/0/features/0	True	The FeatureData document of the node as a JSON resource. The resource array id (e.g.`0`) is used as the key of the document {content type: text/plain, content encoding: {NONE, *GZIP*}}
/SceneServer/layers/0/nodes/0/geometries/0	False	The GeometryData of the node as a binary resource. The resource array id (e.g.`0`) is used as the key of the resource {content type: application/octet-stream, content encoding {NONE, *GZIP*}}

/SceneServer/layers/0/nodes/0/textures/0_0	True	The Texture of the node as a binary resource. The resource id (e.g.`0_0`) is used as the key of the resource {content type: image/jpeg, content encoding {*NONE*}}
/SceneServer/layers/0/nodes/0/textures/0_0_1	True	The compressed texture of the node as a binary resource. The resource id (e.g.`0_0_1`) is used as the key of the resource {content type: image/vnd-ms.dds, content encoding {NONE, *GZIP*}}
/SceneServer/layers/0/nodes/0/attributes/f_0/0	True	The AttributeData as a binary resource. The resource id (e.g.`0`) is used as the key of the resource {content type: application/octet-stream, content encoding: {NONE, *GZIP*}}
/SceneServer/layers/0/nodes/0/attributes/f_1/0	True	same as the attributeData resource `f_0/0` above
....	....	....
/SceneServer/layers/0/nodes/1-4-2-0	False	same as node resource `root` and `0`

Table 36: example showing the layout of a SceneService

Table 36 Detail: A typical example showing the layout of a SceneService in a key value store environment. The example illustrates the structure of the service using a 3D Object scene layer containing textured geometries as well as attribute data.

Annex A: Abstract Test Suite

An Abstract Test Suite is not required for a Community Standard

Annex B: Example JSON encoding for a 3dSceneLayer mesh pyramid

The below example code is also available online at:
http://schemas.opengis.net/i3s/1.0/profiles/meshpyramids/3dSceneLayer.js


```
/**
        Example I3S 1.6 3d Scene Layer Resource for the Meshpyramids profile.
*/
{
        "id": 0, // the ID of this layer, unique within a 3dSceneService.
        "layerType": "3DObject", // the type of this layer, one of {Point, Line, Polygon, *3DObject*, Pointcloud}
        "version": "ee4fbf04-e882-444e-854d-cd519b68594a", // the newest version (store update session ID) of this
layer.
        "ZFactor": 1.0, // Multiplier for z ordinate to arrive at meters.
        "name": "PublicBuildings", // the name of this layer.
        "spatialReference": // The spatialReference of the layer including the vertical coordinate system. wkt is
included to support custom spatial references
        {
        "wkid": 4326,
        "latestWkid": 4326,
        "vcsWkid": 3855,
        "latestVcsWkid": 3855,
        "wkt":
"GEOGCS[\"GCS_WGS_1984\",DATUM[\"D_WGS_1984\",SPHEROID[\"WGS_1984\",6378137,298.257223563]],
PRIMEM[\"Greenwich\",0],UNIT[\"Degree\",0.0174532925199433295]],VERTCS[\"EGM2008_Geoid\",VDATUM[\"
EGM2008_Geoid\"],PARAMETER[\"Vertical_Shift\",0.0],PARAMETER[\"Direction\",1.0],UNIT[\"Meter\",1.0]]}"
  },
        "heightModelInfo": { //enables consuming clients to perform quick test whether this layer is mashable or not
with exisitng content they have.
                        "heightModel": "orthometric", //one of {*"orthometric"*, "ellipsoidal"};
                        "ellipsoid": "wgs84 (G1674)/", //datum realization
                        "heightUnit": "meter" //units
        },
        "alias": "Public Buildings", // the display alias to be used for this layer.
        "description" : "This layer contains textured Building features for the City of Zurich.\n", // Cache description
        "copyrightText" : "Vermessungsamt der Stadt Z□rich", // copyright/usage information
        "capabilities" : ["View","Query"], // capabilities possible on this layer.,
        "cachedDrawingInfo": {"color":  //cachedDrawingInfo.color - a true value indicates that the drawingInfo
color is captured/cached as vertex colors. The drawingInfo used to generate the color cache is saved and present in
scene service layer cache.
        false}, //A false value (or abscene of the object) indicates that the scene cache for the layer does not include a
cached representation of the symbology for color.The client applies standard behavior where material/meshcolors are
interpreted and any drawinginfo present at the WebScene Layer or Map is rendered over it.
        "drawingInfo":
{"renderer":{"type":"simple","symbol":{"type":"MeshSymbol3D","symbolLayers":[{"type":"Fill","material":{"color":
[255,255,255],"transparency":0}}]},"label":"","description":""}} // the layer drawingInfo. Refrer to
https://developers.arcgis.com/web-map-specification/objects/renderer/
        "store": { // information on the store
                "id" : "9f62cd8f-0ab7-451e-917a-65ec8e10a432", // store ID - unique across a SceneServer.
Enables the client to discover which layers a part of a common store, if any.
                "profile": "meshpyramids", // Indicates which profile this scene store fulfills. One of {polygons,
points, lines, meshpyramids, pointclouds}.
                "resourcePattern": ["3dNodeIndexDocument", "Geometry", "Texture"], // The resources need for
rendering and the required order in which the client should load them.
                "rootNode": "./nodes/root", // relative URL to root node resource.
                "version": "1.6", // format version of this resource; used here again if this store hasn't been served
by a 3D Scene Server.
                "extent": [8.54, 47.385, 8.72, 47.455], // the spatial extent of this store, in the horizontal indexCRS
(xmin, ymin, xmax, ymax)
```

```
                "indexCRS": "http://www.opengis.net/def/crs/EPSG/0/4326", // the horizontal CRS used for all minimum bounding spheres (mbs) in this cache, identified by a OGC URL.
                "vertexCRS": "http://www.opengis.net/def/crs/EPSG/0/4326", // the horizontal CRS used for all "vertex positions" in this cache, identified by a OGC URL.
                "normalReferenceFrame": "earth-centered", // One of {east-north-up, *earth-centered*, vertex-reference-frame}.*east-north-up* indicates normals are stored with easting, northing and up directions; *earth-centered* indicates normals are stored in earth-centered, earth-fixed (ECEF) reference frame.
                //*vertex-reference-frame* indicates that normals are stored in the same reference frame as vertices and is only valid for projected vertexCRS.
                "nidEncoding": "application/vnd.esri.i3s.json+gzip; version=1.6", // MIME type for the encoding used for the Node Index Documents
                "featureEncoding": "application/vnd.esri.i3s.json+gzip; version=1.6",  // MIME type for the encoding used for the Feature Data Resources
                "geometryEncoding": "application/octet-stream+gzip; version=1.6", // MIME type for the encoding used for the Geometry Resources
                "textureEncoding": ["image/jpeg", "image/vnd-ms.dds"], // MIME types for the encoding used for the Texture Resources
                "lodType": "MeshPyramid", // optional field to indicate which LoD Generation Scheme is used in this store. Selected from {*MeshPyramid*, FeatureTree, Thinning, Clustering, Generalizing}.
                "lodModel": "node-switching", // optional field to indicate which LoD Switching mode clients have to use. One of {*node-switching*, feature-switching, none}.
                "defaultGeometrySchema": { // geometry resource layout for nodes that declare the use of defaultGeometrySchema in the node index.
                        "geometryType": "triangles", // Low-level default geometry type, one of {triangle_strip, triangles, lines, points}; if defined, all geometries in the store are expected to have this type.
                        "header": [             // header fields that precede the vertex data
                                {
                                        "property": "vertexCount", // vertex count
                                        "type": "UInt32" // the element type, from {UInt8, UInt16, UInt32, UInt64, Int16, Int32, Int64, *Float32*, Float64}
                                },
                                {
                                        "property": "faceCount", // face count
                                        "type": "UInt32" // the element type, from {UInt8, UInt16, UInt32, UInt64, Int16, Int32, Int64, *Float32*, Float64}
                                },
                                {
                                        "property": "featureCount", // feature count
                                        "type": "UInt32" // the element type, from {UInt8, UInt16, UInt32, UInt64, Int16, Int32, Int64, *Float32*, Float64}
                                }
                        ],
                        "topology": "PerAttributeArray", // one of ["PerAttributeArray", "Indexed"]. When "Indexed", the indices must also be declared in the geometry schema ("faces") and precede the vertexAttribute data.
                        "ordering": ["position", "normal", "uv0", "color", "region"], // provides the order of the keys in vertexAttributes and faceAttributes, if present.
                        "vertexAttributes": { // the vertex attributes must appear in the order that they are declared here.
                                "position": { // the name of the vertex attribute; here: vertex positions
                                        "valueType": "Float32", // the element type, from {UInt8, UInt16, UInt32, UInt64, Int16, Int32, Int64, *Float32*, Float64}
                                        "valuesPerElement": 3  // number of (Float32) values need to make a valid element (here a xyz position)
                                },
                                "normal": { // the name of the vertex attribute; here: vertex normals
                                        "valueType": "Float32", // the element type, from {UInt8, UInt16, UInt32, UInt64, Int16, Int32, Int64, *Float32*, Float64}
                                        "valuesPerElement": 3 // number of (Float32) values need to make a valid element (here a normal vector)
                                },
                                "uv0": { // the name of the vertex attribute; here: 1st texture coordinates, must be present if a textureID is referenced
```

```
                                                "valueType": "Float32", // the element type, from {UInt8, UInt16,
UInt32, UInt64, Int16, Int32, Int64, *Float32*, Float64}
                                                "valuesPerElement": 2 // number of (Float32) values need to make a
valid element (here a texture coordinate that will be normalized)
                                        },
                                        "color": { // the name of the vertex attribute; here: color as RGBA
                                                "valueType": "UInt8", //  the element type, always UInt8 for color
value
                                                "valuesPerElement": 4 //number of (UInt8) values need to make a
valid element (here a color in RGBA format)
                                        },
                                        "region": { // per-vertex region info. analogous to textureDefinitions.regions in
sharedResource. Values define uv-coordinates of region borders: [umin, vmin, umax, vmax]
                                                "valueType": "UInt16", // the element type, always UInt16 for region
info
                                                "valuesPerElement": 4 // number of (UInt16) values need to make a
valid element (here a region info)
                                        }
                                },
                                "featureAttributeOrder": ["id", "faceRange"], // provides the order of the keys in
featureAttributes object, if present.
                                "featureAttributes": {
                                        "id": {
                                                "valueType": "UInt64", // the element type, from {UInt8, UInt16,
UInt32, UInt64, Int16, Int32, Int64, *Float32*, Float64}
                                                "valuesPerElement": 1 // number of (UInt64) values need to make a
valid element (here a feature ID)
                                        },
                                        "faceRange": {
                                                "valueType": "UInt32", // the element type, from {UInt8, UInt16,
UInt32, UInt64, Int16, Int32, Int64, *Float32*, Float64}
                                                "valuesPerElement": 2 // number of (UInt32) values need to make a
valid element (here a faceRange with minIndex/length)
                                        }
                                }
                        }
                        }
                },
                "fields" :  [ // schema definition for this layer, as with 2D Layers.
                        {
                          "name" : "ObjectID",
                          "type" : "esriFieldTypeOID",
                          "alias" : "ObjectID"
                        },
                        {
                          "name" : "type",
                          "type" : "esriFieldTypeString",
                          "alias" : "Building Type"
                        },
                        {
                          "name" : "totalHeight",
                          "type" : "esriFieldTypeFloat",
                          "alias" : "Total Height"
                        },
                        {
                          "name" : "eaveHeight",
                          "type" : "esriFieldTypeDouble",
                          "alias" : "Eave Height"
                        }
                ],
                "attributeStorageInfo":[ // see AttributeData section in spec
                {
```

```
            "key": "f_0",
            "name": "objectid",
            "header": [{
                        "property": "count",
                        "valueType": "UInt32"
            }],
            "ordering": ["ObjectIds"],
            "objectIds": {
                        "valueType": "UInt32",
                        "valuesPerElement": 1
             }
  },
  {
            "key": "f_1",
            "name": "type",
            "header": [{
                        "property": "count",
                        "valueType": "UInt32"
            }, {
                        "property": "attributeValuesByteCount",
                        "valueType": "UInt32"
            }],
            "ordering": ["attributeByteCounts", "attributeValues"],
            "attributeByteCounts": {
                        "valueType": "UInt32",
                        "valuesPerElement": 1
            },
            "attributeValues": {
                        "valueType": "String",
                        "encoding": "UTF-8",
                        "valuesPerElement": 1
            }
  },
            {
            "key": "f_2",
            "name": "totalHeight",
            "header": [{
                        "property": "count",
                        "valueType": "UInt32"
            }],
            "ordering": ["attributeValues"],
            "attributeValues": {
                        "valueType": "Float32",
                        "valuesPerElement": 1
             }
  },
            {
            "key": "f_3",
            "name": "totalHeight",
            "header": [{
                        "property": "count",
                        "valueType": "UInt32"
            }],
            "ordering": ["attributeValues"],
            "attributeValues": {
                        "valueType": "Float64",
                        "valuesPerElement": 1
             }
   }
 ]
}
```

Annex C: Contributor Acknowledgements

Contributors: Tamrat Belayneh, Javier Gutierrez, Markus Lipp, Johannes Schmid, Simon Reinhard, Thorsten Reitz, Chengliang Shan, Ben Tan, Moxie Zhang, Pascal Müller, Dragan Petrovic, Sud Menon

Acknowledgements: Bart van Andel, Fabien Dachicourt, Carl Reed

Annex D: Revision history

Date	Release	Editor	Paragraph modified	Description
2/16/2017	1	Carl Reed	Various	Put I3S into the OGC document template
2/27/2017	1	Carl Reed	New clauses	Added based on OAB concerns with Community Standards
5/10/2017	1	Carl Reed	Various	More Terms and definitions, more changes to CRS clauses. Finish processing public comment suggestions.

Annex E - Scene Service Access to REST Resources. Informative

This is the description of a REST API for a Scene Service. Please note that the examples use the Esri ArcGIS REST implementation. Simply change the base URL pattern (see below) for access to the I3S services available on your system.

There is a set of REST resources also defined in the I3S format specification that are served out via different endpoints.

There is a base URL that needs to be defined and used in all I3S scene service access REST resource instances. This base URL points to where your I3S host services and content are located. In addition there are 6 mandatory resource instances and 2 optional resource instances.

Mandatory:

- 3dSceneServiceInfo (JSON)
- 3dSceneLayerInfo (JSON)
- 3dNodeIndexDocument (JSON)
- SharedResources (JSON)
- TextureData (Binary)
- GeometryData (Binary)

Optional:

- FeatureData JSON (optional for Mesh-Pyramids profile)
- AttributeData (Binary)

This is the REST API for retrieval of these resources:

3dSceneServiceInfo

- *URL Pattern*: `<base-url>/<server-name>/SceneServer`
- *Method*: `GET`
- *Example Service*:http://3dcities.maps.arcgis.com/arcgis/rest/services/New_York_LoD2_3D_Buildings/SceneServer
- *Returns*: Scene Service metadata and list of available layers.

3dSceneLayerInfo

- *URL Pattern*: `<scene-server-url>/layers/<layer-id>`
- *Method*: `GET`

- *Example Service*:http://3dcities.maps.arcgis.com/arcgis/rest/services/New_York_LoD2_3D_Buildings/SceneServer/layers/0
- *Returns*: Detailed information about single layer, including symbology, field schema, and profile/store metadata, with a link to the root 3dNodeIndexDocument

3dNodeIndexDocument

- *URL Pattern*: `<layer-url>/nodes/<node-id>`
- *Method*: `GET`
- *Example Service*:http://3dcities.maps.arcgis.com/arcgis/rest/services/New_York_LoD2_3D_Buildings/SceneServer/layers/0/nodes/5-1-0-0-0
- *Returns*: A file describing a single node in the spatial index, with links to all associated resources such as FeatureData, textures, Geometry and SharedResources

SharedResources

- *URL Pattern*: `<node-url>/shared/`
- *Method*: `GET`
- *Example Service*:http://3dcities.maps.arcgis.com/arcgis/rest/services/New_York_LoD2_3D_Buildings/SceneServer/layers/0/nodes/5-1-0-0-0/features/0
- *Returns*: A feature data resource (bundle)

FeatureData

- *URL Pattern*: `<node-url>/features/<feature-data-bundle-id>`
- *Method*: `GET`
- *Example Service*:http://3dcities.maps.arcgis.com/arcgis/rest/services/New_York_LoD2_3D_Buildings/SceneServer/layers/0/nodes/5-1-0-0-0/features/0
- *Returns*: A feature data resource (bundle). In case of Points layer type, the feature data document contains positional information and could also include by value any attribute information associated with each feature. For layer types belonging to Mesh-Pyramids profile, this resource is optional as all required information to identify feature to geometry mapping is compactly stored with the binary geometry data. In addition, attribute data in Mesh-Pyramids profiles are stored as AttributeData resource as described in the I3S format specification.

GeometryData

- *URL Pattern*: `<node-url>/geometries/<geometry-data-bundle-id>`
- *Method*: `GET`

- *Example Service*: http://3dcities.maps.arcgis.com/arcgis/rest/services/New_York_LoD2_3D_Buildings/SceneServer/layers/0/nodes/5-1-0-0-0/geometries/0
- *Returns*: A geometry data resource (bundle)

TextureData

- *URL Pattern*: `<node-url>/textures/<texture-data-bundle-id>`
- *Method*: `GET`
- *Example Service*:http://scene.arcgis.com/arcgis/rest/services/Hosted/Buildings_San_Francisco/SceneServer/layers/0/nodes/1-3-0-0-0-0-0-0-0/textures/0_0
- *Returns*: A texture data resource (bundle). Refer to the I3S format specification for details on how different encodings and resolutions are encoded.

AttributeData

- *URL Pattern*: `<node-url>/attributes/<attribute-data-bundle-id>`
- *Method*: `GET`
- *Example Service*: http://3dcities.maps.arcgis.com/arcgis/rest/services/New_York_LoD2_3D_Buildings/SceneServer/layers/0/nodes/5-1-0-0-0/attributes/f_0/0
- *Returns*: An attribute data resource (bundle). Refer to the I3S format specification for details on how different types of attributes are encoded.

Annex F: I3S profile: Points

Summary

What this profile is for: Support for points and multi-points with symbolization. This profile does not use external ArrayBufferGeometries.

Access Pattern

The access pattern is identical to that of the mesh-pyramids profile. The profile utilizes different LoD selection metrics (`screenSpaceRelative`, `distanceRangeFromDefaultCamera`).

Schema

The points profile makes use of 5 main resource types and allows a restricted set of properties.

SceneServiceInfo

No specific profile.

3dSceneLayer

Property	Required	Min.	Max.	Container	Value Rules	Conditions
`id`	True			None	number/Integer	
`href`				None	string/URL	
`layerType`	True			None	string/None	
`ZFactor`				None	number/Float	
`version`	True			None	string/UUID	
`name`	True			None	string/None	
`alias`				None	string/None	
`description`				None	string/None	
`copyrightText`				None	string/None	
`capabilities`	True	1	3		object/None	
`capabilities.*`	True			undefined	string/None	
`store`	True			None	object/None	
`store.id`	True			None	string/UUID	
`store.profile`	True			None	string/None	
`store.rootNode`	True			None	string/URL	
`store.version`	True			None	string/None	
`store.extent`	True	4	4		object/None	
`store.extent.*`	True			undefined	number/None	
`store.indexCRS`	True			None	string/URL	

store.vertexCRS	True			None	string/URL	
store.nidEncoding				None	string/None	
store.featureEncoding				None	string/None	
store.geometryEncoding				None	string/None	
store.textureEncoding	True	1	3		object/None	
store.textureEncoding.*				None	string/None	
store.lodType	True			None	string/None	
store.lodModel	True			None	string/None	
store.featureOrdering		0	*		object/None	
store.featureOrdering.*				undefined	string/None	
store.defaultGeometrySchema				None	object/NamedRuleset	
store.indexingScheme				None	object/NamedRuleset	
fields	True	0	*		object/None	
fields.*	True			undefined	object/NamedRuleset	
drawingInfo				None	object/None	

3dNodeIndexDocument

Property	Required	Min.	Max.	Container	Value Rules	Conditions
id	True			None	string/NodeID	/level NOT 1
id	True			None	string/None	/level IS 1
level	True			None	number/Integer	
version	True			None	string/UUID	
created				None	string/Date	
expires				None	string/Date	
mbs	True	4	4		object/None	
mbs.*	True			undefined	number/None	
lodSelection	True	0	*		object/None	[../../]/store/lodType IS FeatureTree
lodSelection.*	True			undefined	object/None	
lodSelection.*.metricType	True			None	string/None	
lodSelection.*.maxError				None	number/None	
lodSelection.*.avgError				None	number/None	
transform	True	16	16		object/None	
transform.*				undefined	number/None	
sharedResource	True			None	object/None	
sharedResource.href	True			None	string/URL	
featureData	True	0	*		object/None	
featureData.*	True			undefined	object/None	
featureData.*.href	True			undefined	string/URL	
featureData.*.featureRange		2	2		object/None	
featureData.*.featureRange.*	True			undefined	number/Integer	
featureData.*.layerContent		1	*		object/None	
featureData.*.layerContent.*				undefined	string/None	
geometryData	True	0	1		object/None	

geometryData.*	True			undefined	object/None	
geometryData.*.href	True			undefined	string/URL	
parentNode	True			None	object/None	/level NOT 1
parentNode.id	True			None	string/NodeID	
parentNode.version	True			None	string/UUID	
parentNode.href	True			None	string/URL	
parentNode.mbs	True	4	4		object/None	
parentNode.mbs.*	True			undefined	number/None	
parentNode.featureCount				None	number/Integer	
children	True	0	99		object/None	
children.*	True			undefined	object/None	
children.*.id	True			None	string/NodeID	
children.*.version	True			None	string/UUID	
children.*.href	True			None	string/URL	
children.*.mbs	True	4	4		object/None	
children.*.mbs.*	True			undefined	number/None	
children.*.featureCount				None	number/Integer	
neighbors	True	0	*		object/None	/level NOT 1
neighbors.*	True			undefined	object/None	
neighbors.*.id	True			None	string/NodeID	
neighbors.*.version	True			None	string/UUID	
neighbors.*.href	True			None	string/URL	
neighbors.*.mbs	True	4	4		object/None	
neighbors.*.mbs.*	True			undefined	number/None	
neighbors.*.featureCount				None	number/Integer	
features	True	0	*		object/None	
features.*	True			undefined	object/None	
features.*.id	True			None	number/Integer	
features.*.mbs	True	4	4		number/None	
features.*.mbs.*	True			undefined	number/None	
features.*.lodChildFeatures	True	0	*		object/None	[../../]/store/lodType IS FeatureTree
features.*.lodChildFeatures.*	True			undefined	number/Integer	
features.*.lodChildNodes	True	0	9		object/None	[../../]/store/lodType IS FeatureTree ../lodChildFeatures NOT null
features.*.lodChildNodes.*	True			undefined	string/NodeID	
features.*.rank	True			None	number/Integer	[../../]/store/lodType IS FeatureTree
features.*.rootFeature	True			None	number/Integer	[../../]/store/lodType IS FeatureTree /level NOT 1 ../rank NOT 1

FeatureData

Property	Required	Min.	Max.	Container	Value Rules	Conditions
featureData	True	1	*		object/None	
featureData.*	True			undefined	object/None	
featureData.*.id	True			None	number/Integer	
featureData.*.position	True	2	3		object/None	
featureData.*.position.*	True			undefined	number/None	
featureData.*.mbb		6	6		object/None	
featureData.*.mbb.*	True			undefined	number/None	
featureData.*.layer	True			None	string/None	
featureData.*.geometries		1	*		object/None	
featureData.*.geometries.*	True			None	object/NamedRuleset	type IS Embedded
featureData.*.attributes	True	0	*		object/None	
featureData.*.attributes.*				undefined	object/NamedRuleset	

Annex G: I3S profile - Mesh-pyramids (MP)

Summary

What this profile is for: This profile is implemented by the 3D Object and Integrated Mesh layer types.

Access Pattern

This section describes how a client is expected to load and handle resources from an Indexed 3D Scene Layer using the Mesh-pyramids profile. The general pattern consists of these phases:

1. Handshake & capabilities negotiation: The client ensures that the service has the expected resources and that a client and a server have a common set of capabilities. Within this phase, the client utilizes the following resources:
    1. Retrieve SceneServiceInfo: General service information
    2. Retrieve 3dSceneLayer: Information on available layers, including symbology and encoding
2. Index exploration: The client retrieves Node Index Documents and decides – based on lodSelection properties – whether it wants to download and render their attached resources. Within this phase, the client utilizes the following resource:
    1. NodeIndexDocument: Summary of the content of a single node of the index, references children, parent and neighbor nodes, indicating what can be found there
3. Rendering: When a client has decided that it wants to render the content of a node, it retrieves the attached resources:
    1. SharedData: Material definitions, shared geometries for instancing
    2. GeometryData: Geometry attributes such as positions and indices
    3. TextureData: Images used as texture maps
    4. AttributeData: Attribute data of features used for attribute-based symbolization (as indicated by the DrawingInfo object in the 3dSceneLayer resource)
4. Identify: Additional resources belonging to a node are accessed only if needed, e.g. for an Identify operation.
    1. AttributeData: If the AttributeData resources of the node have not already been fetched (in step 3 above) client application can request the desired attribute data.

A familiar access pattern based on a single tree data structure is proposed for view frustum culling, level-of-detail selection, and rendering. The following pseudo code illustrates the recommended pattern when navigating an index tree using Mesh Pyramids.

Node traversal starts at the root node and recursively calls TraverseNodeTree(node):

```
TraverseNodeTree(node)
{
```

```
    if (node's mbs is not visible) // see 1)
        // do nothing
    else if (node has no children or ScreenSize(mbs) <
maxScreenThreshold) //see 2)
        // render the node // see 3)
    else
        for each child in children(node) // see 4)
            TraverseNodeTree(child);
}
```

Additional notes:

1. view frustum culling:
    1. visibility test can include the 'entirely inside the viewing frustum' result which can be used to optimize away all further frustum culling tests on the children of the node
    2. this step can also optionally incorporate a cutoff distance threshold test if desired.
2. level-of-detail selection:
    1. test used to decide how deep to recurse is based on mbs' projected size (diameter) on the screen vs the per node provided 'maxScreenThreshold'.
3. Rendering:
    1. "render the node" potentially includes some, or all, of the following steps:
        1. Requesting the corresponding geometry and texture data if not already requested
        2. (asynchronously) accessing the corresponding geometry and texture data and loading it into GPU memory if not already loaded
        3. Binding, if loaded, the geometry VBO
        4. Binding, if loaded, the texture
        5. Making a draw() call if, at least, the geometry is loaded
4. optimized user experience:
    1. children should be sorted by the ascending distance from the observer…

Schema

The mesh pyramids profile makes use of all 7 main resource types and allows for a restricted set of properties. Note that the FeatureData resource is optional for this profile. Hence the 3dSceneLayer resource must contain a DefaultGeometrySchema.

SceneServiceInfo

No specific profile.

3dSceneLayer

Note that in this profile, the DefaultGeometrySchema is mandatory.

Property	Required	Min.	Max.	Container	Value Rules	Conditions
id	True			None	number/Integer	
href				None	string/URL	
layerType	True			None	string/None	
ZFactor				None	number/Float	
version	True			None	string/UUID	
name	True			None	string/None	
alias				None	string/None	
description				None	string/None	
copyrightText				None	string/None	
capabilities	True	1	3		object/None	
capabilities.*	True			undefined	string/None	
store	True			None	object/None	
store.id	True			None	string/UUID	
store.profile	True			None	string/None	
store.resourcePattern	True	1	5		object/None	
store.resourcePattern.*	True			None	string/None	
store.rootNode	True			None	string/URL	
store.version	True			None	string/None	
store.extent	True	4	4		object/None	
store.extent.*	True			undefined	number/None	
store.indexCRS	True			None	string/URL	
store.vertexCRS	True			None	string/URL	
store.nidEncoding				None	string/None	
store.featureEncoding				None	string/None	
store.geometryEncoding				None	string/None	
store.textureEncoding	True	1	3		object/None	
store.textureEncoding.*				None	string/None	
store.lodType	True			None	string/None	
store.lodModel	True			None	string/None	
store.defaultGeometrySchema				None	object/NamedRuleset	
store.indexingScheme				None	object/NamedRuleset	
store.defaultTextureDefinition				undefined	object/NamedRuleset	
store.defaultMaterialDefinition				undefined	object/NamedRuleset	
fields	True	0	*		object/None	
fields.*	True			undefined	object/NamedRuleset	
drawingInfo				None	object/None	

3dNodeIndexDocument

There is always exactly 1 geometry and texture resource per node.

Property	Required	Min.	Max.	Container	Value Rules	Conditions
id	True			None	string/NodeID	/level NOT 1
id	True			None	string/None	/level IS 1
level	True			None	number/Integer	
version	True			None	string/UUID	
created				None	string/Date	
expires				None	string/Date	
mbs	True	4	4		object/None	
mbs.*	True			undefined	number/None	
lodSelection	True	1	1		object/None	
lodSelection.*	True			undefined	object/None	
lodSelection.*.metricType	True			None	string/None	
lodSelection.*.maxError				None	number/None	
lodSelection.*.avgError				None	number/None	
transform		16	16		object/None	
transform.*	True			undefined	number/None	
sharedResource	True			None	object/None	
sharedResource.href	True			None	string/URL	
featureData	True	0	1		object/None	
featureData.*	True			undefined	object/None	
featureData.*.href	True			undefined	string/URL	
featureData.*.featureRange		2	2		object/None	
featureData.*.featureRange.*	True			undefined	number/Integer	
featureData.*.layerContent		1	1		object/None	
featureData.*.layerContent.*				undefined	string/None	
geometryData	True	0	1		object/None	
geometryData.*	True			undefined	object/None	
geometryData.*.href	True			undefined	string/URL	
textureData	True	0	*		object/None	
textureData.*	True			undefined	object/None	
textureData.*.href	True			undefined	string/URL	
textureData.*.multiTextureBundle				undefined	boolean/None	
parentNode	True			None	object/None	/level NOT 1
parentNode.id	True			None	string/NodeID	/level NOT 2
parentNode.id	True			None	string/None	/level IS 2
parentNode.version				None	string/UUID	
parentNode.href	True			None	string/URL	
parentNode.mbs	True	4	4		object/None	
parentNode.mbs.*	True			undefined	number/None	
parentNode.featureCount				None	number/Integer	
children	True	0	99		object/None	

children.*	True			undefined	object/None	
children.*.id	True			None	string/NodeID	
children.*.version				None	string/UUID	
children.*.href	True			None	string/URL	
children.*.mbs	True	4	4		object/None	
children.*.mbs.*	True			undefined	number/None	
children.*.featureCount				None	number/Integer	
neighbors	True	0	*		object/None	/level NOT 1
neighbors.*	True			undefined	object/None	
neighbors.*.id	True			None	string/NodeID	
neighbors.*.version	True			None	string/UUID	
neighbors.*.href	True			None	string/URL	
neighbors.*.mbs	True	4	4		object/None	
neighbors.*.mbs.*	True			undefined	number/None	
neighbors.*.featureCount				None	number/Integer	

AttributeData

Attribute data for all features in a node is stored and made available as discrete, per field resource called ***attribute***. The number of attribute resources corresponds to the number of feature data *fields* that are chosen to be included along with the 3d Scene Layer cache.

FeatureData

The FeatureData is optional with this profile.

Property	Required	Min.	Max.	Container	Value Rules	Conditions
featureData	True	0	*		object/None	
featureData.*	True			undefined	object/None	
featureData.*.id	True			None	number/Integer	
featureData.*.position	True	2	3		object/None	
featureData.*.position.*	True			undefined	number/None	
featureData.*.pivotOffset		3	3		object/None	
featureData.*.pivotOffset.*				undefined	number/None	
featureData.*.mbb	True	6	6		object/None	
featureData.*.mbb.*	True			undefined	number/None	
featureData.*.layer	True			None	string/None	
featureData.*.geometries	True	1	1		object/None	
featureData.*.geometries.*	True			undefined	object/NamedRuleset	
featureData.*.attributes		0	*		object/None	
featureData.*.attributes.*				undefined	object/NamedRuleset	
geometryData	True	1	*		object/None	
geometryData.*	True			undefined	object/NamedRuleset	type IS ArrayBufferView

SharedResources

Property	Required	Min.	Max.	Container	Value Rules	Conditions
`materialDefinitions`	True	0	*		object/None	
`materialDefinitions.*`	True			undefined	object/NamedRuleset	
`textureDefinitions`	True	0	*		object/None	
`textureDefinitions.*`	True			undefined	object/NamedRuleset	
`shaderDefinitions`				None	object/None	