**Open Geospatial Consortium**

Submission Date: 2019-09-12

Approval Date:   2019-xx-xx

Publication Date:   2019-dd-mm

External identifier of this OGC® document: http://www.opengis.net/doc/CS/i3s/1.1

Internal reference number of this OGC® document:    17-014r7

Version: 1.1

Category: OGC® Community Standard

Editor:    Carl Reed, Tamrat Belayneh

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

**Copyright notice**

Copyright © 2019 Open Geospatial Consortium  
To obtain additional rights of use, visit <http://www.opengeospatial.org/legal/>.

**Warning**

This document is an OGC Member endorsed international community standard. This document is available on a royalty free, non-discriminatory basis. Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type:    OGC® Candidate Community Standard

Document subtype:    if applicable

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]](#footnote-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](https://creativecommons.org/licenses/by-sa/4.0/), provide a link to the license, and [indicate if changes were made](https://creativecommons.org/licenses/by-sa/4.0/). 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](https://creativecommons.org/licenses/by-sa/4.0/) as the original.
* **No additional restrictions** — You may not apply legal terms or [technological measures](https://creativecommons.org/licenses/by-sa/4.0/) 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](https://creativecommons.org/licenses/by-sa/4.0/).
* 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](https://creativecommons.org/licenses/by-sa/4.0/) may limit how you use the material.

<[Skip Directly to Standard](#_Introduction)!>

Contents

[1. Introduction 12](#_Toc17566613)

[2. Conformance 12](#_Toc17566614)

[3. References 12](#_Toc17566615)

[4. Terms and Definitions 13](#_Toc17566616)

[5. Conventions 16](#_Toc17566617)

[6. Introduction to I3S and SLPK 17](#_Toc17566618)

[6.1 I3S Design Principles 17](#_Toc17566619)

[6.2 I3S – Overview 17](#_Toc17566620)

[7. I3S Specification 18](#_Toc17566621)

[7.1 Coordinate Reference Systems (CRS) 19](#_Toc17566622)

[7.1.1 A note on OGC Standards for CRS and Well Known Text. 19](#_Toc17566623)

[7.1.2 CRS use and requirements in I3S 20](#_Toc17566624)

[7.2 Height Models 21](#_Toc17566625)

[7.3 Indexed Scene Layers - Organization and Structure 23](#_Toc17566626)

[7.3.1 I3S - Indexing Model and Tree Structure 23](#_Toc17566627)

[7.3.2 Geometry Model and Storage 29](#_Toc17566628)

[7.3.3 Textures 29](#_Toc17566629)

[7.3.4 Attribute Model and Storage 30](#_Toc17566630)

[7.4 Bounding Volume Hierarchy 30](#_Toc17566631)

[7.5 Level of Detail 30](#_Toc17566632)

[7.5.1 Discrete LoDs 31](#_Toc17566633)

[7.5.2 Representation of input data that already has explicitly authored multiple representations 32](#_Toc17566634)

[7.5.3 LoD Switching Modes 32](#_Toc17566635)

[7.5.3.1 Node Switching 32](#_Toc17566636)

[7.5.4 Levels of Detail Generation 33](#_Toc17566637)

[7.5.5 LoD Selection Metrics 34](#_Toc17566638)

[7.6 JSON Resources Schema and Documentation 35](#_Toc17566639)

[7.6.1 Basic Value Types 35](#_Toc17566640)

[7.6.2 Pointers 35](#_Toc17566641)

[7.6.3 SceneServiceInfo 36](#_Toc17566642)

[7.6.3.1 Class SceneServiceInfo 37](#_Toc17566643)

[7.6.4 3dSceneLayerInfo [Common Profiles] 37](#_Toc17566644)

[7.6.4.1 Class 3dSceneLayerInfo (Common Profiles) 38](#_Toc17566645)

[7.6.4.2 Class Store 40](#_Toc17566646)

[7.6.4.3 defaultGeometrySchema 44](#_Toc17566647)

[7.6.4.4 Class\_HeaderAttribute 45](#_Toc17566648)

[7.6.4.5 field 46](#_Toc17566649)

[7.6.4.6 attributeStorageInfo 49](#_Toc17566650)

[7.6.4.7 Class IndexScheme (deprecated) 50](#_Toc17566651)

[7.6.4.8 DrawingInfo 50](#_Toc17566652)

[7.6.4.9 spatialReference 52](#_Toc17566653)

[7.6.4.10 heightModelInfo 53](#_Toc17566654)

[7.6.4.11 elevationInfo 54](#_Toc17566655)

[7.6.4.12 popupInfo 54](#_Toc17566656)

[7.6.4.13 serviceUpdateTimeStamp 55](#_Toc17566657)

[7.6.4.14 statisticsInfo 55](#_Toc17566658)

[7.6.4.15 VertexAttributes 56](#_Toc17566659)

[7.6.4.16 value 57](#_Toc17566660)

[7.6.5 3dNodeIndexDocument 57](#_Toc17566661)

[7.6.5.1 Class 3dNodeIndexDocument 58](#_Toc17566662)

[7.6.5.2 nodeReference 59](#_Toc17566663)

[7.6.5.3 Resource 60](#_Toc17566664)

[7.6.5.4 Class Feature (Deprecated) 60](#_Toc17566665)

[7.6.5.5 Class LodSelection 61](#_Toc17566666)

[7.6.5.6 obb 62](#_Toc17566667)

[7.6.6 FeatureData 63](#_Toc17566668)

[7.6.6.1 Class FeatureData 64](#_Toc17566669)

[7.6.6.2 Class featureAttribute 64](#_Toc17566670)

[7.6.6.3 Class geometry 64](#_Toc17566671)

[7.6.6.4 Class GeometryParams 65](#_Toc17566672)

[7.6.6.5 Class GeometryReferenceParams 66](#_Toc17566673)

[7.6.6.6 Class VestedGeometryParams 66](#_Toc17566674)

[7.6.6.7 Class SingleComponentParams 68](#_Toc17566675)

[7.6.6.8 Class Component 68](#_Toc17566676)

[7.6.6.9 Class geometryAttribute 69](#_Toc17566677)

[7.7 Shared Resources 69](#_Toc17566678)

[7.7.1 Class SharedResource 70](#_Toc17566679)

[7.7.1.1 Class Material 70](#_Toc17566680)

[7.7.1.2 Class Texture 71](#_Toc17566681)

[7.7.1.3 Class Image 72](#_Toc17566682)

[7.7.1.4 Class Renderer 72](#_Toc17566683)

[7.7.1.5 Class Symbol 73](#_Toc17566684)

[7.7.1.6 Class SymbolLayers 73](#_Toc17566685)

[7.7.1.7 Class ShaderDefinition 73](#_Toc17566686)

[7.7.1.8 Class Symbol 74](#_Toc17566687)

[8. I3S File Formats 74](#_Toc17566688)

[8.1 Textures 74](#_Toc17566689)

[8.1.1 Texture Recommendations and Requirements 74](#_Toc17566690)

[8.1.2 Image Formats 74](#_Toc17566691)

[8.1.3 Texture Sets 74](#_Toc17566692)

[8.1.4 Atlas usage and Regions 75](#_Toc17566693)

[8.1.5 Texture coordinates 75](#_Toc17566694)

[8.1.6 Generating Image IDs 75](#_Toc17566695)

[8.1.6.1 Function Parameters 75](#_Toc17566696)

[8.2 Geometry 76](#_Toc17566697)

[8.3 Attribute Data 77](#_Toc17566698)

[8.3.1 The content of this binary attribute resource is made up of: 78](#_Toc17566699)

[8.3.2 REST API for Accessing Attribute Resources directly from a scene service layer 81](#_Toc17566700)

[8.3.3 A typical usage pattern of the attributes REST API 81](#_Toc17566701)

[8.3.4 Attribute Resources: Details 84](#_Toc17566702)

[8.4 Accessing the Legend of a 3D Object Layer 86](#_Toc17566703)

[9. Supported I3S Layer Structures 86](#_Toc17566704)

[9.1 Point Scene Layer 86](#_Toc17566705)

[9.1.1 Point Scene Layer Structure 87](#_Toc17566706)

[9.1.2 *Example of point scene layer structure* 87](#_Toc17566707)

[9.1.3 HTTP API Overview 88](#_Toc17566708)

[9.2 Integrated Mesh Scene Layer 89](#_Toc17566709)

[9.2.1 Integrated Mesh Scene Layer Structure 90](#_Toc17566710)

[9.2.2 Example of integrated mesh scene layer structure 90](#_Toc17566711)

[9.2.3 Integrated Mesh Scene Layer HTTP API Overview 91](#_Toc17566712)

[9.3 3D Object Scene Layer 92](#_Toc17566713)

[9.3.1 3D Object Scene Layer Structure 94](#_Toc17566714)

[9.3.2 Example of 3D Object layer structure 94](#_Toc17566715)

[9.3.3 HTTP API Overview 94](#_Toc17566716)

[9.4 Point Cloud Scene Layer 98](#_Toc17566717)

[9.4.1 Point Cloud Scene Layer Structure 99](#_Toc17566718)

[9.4.2 Example of point cloud layer structure 99](#_Toc17566719)

[9.4.3 HTTP API Overview 100](#_Toc17566720)

[9.5 SLPK hash table 101](#_Toc17566721)

[9.5.1 To create the SLPK hash-table 101](#_Toc17566722)

[9.5.2 To read SLPK hash-table 102](#_Toc17566723)

[10. Additional Informative Information 102](#_Toc17566724)

[10.1 Flexibility 102](#_Toc17566725)

[10.2 Summary of I3S Defining Characteristics 103](#_Toc17566726)

[11. Persistence 104](#_Toc17566727)

[11.1 Scene Layer Packages 104](#_Toc17566728)

[11.1.1 Metadata 107](#_Toc17566729)

[11.2 Key Value Stores 107](#_Toc17566730)

[Annex A: Abstract Test Suite 110](#_Toc17566731)

[Annex C: Contributor Acknowledgements 110](#_Toc17566732)

[Annex D: Revision history 110](#_Toc17566733)

[Annex E - Scene Service Access to REST Resources. Informative 111](#_Toc17566734)

[Annex G: I3S Scene Layer Profile for Points 114](#_Toc17566735)

[3DSceneLayerInfo [Point Profile] 114](#_Toc17566736)

[3DNodeIndexDocument 120](#_Toc17566737)

[Annex H: I3S Scene Layer Profile - Mesh-pyramids (MP) 123](#_Toc17566738)

[Annex I I3S Point Cloud Scene Layer Profile 128](#_Toc17566739)

[I3S point cloud Scene Layer (profile of 3dSceneLayerInfo 128](#_Toc17566740)

[I3S point cloud scene layer: attributeInfo 132](#_Toc17566741)

[I3S point cloud scene layer profile: Store 134](#_Toc17566742)

[I3S point cloud scene layer: Index 135](#_Toc17566743)

[I3S point cloud scene layer: defaultGeometrySchema 136](#_Toc17566744)

[I3S point cloud scene layer: vertexAttributes 138](#_Toc17566745)

[I3S point cloud scene layer: Attribute Statistics 138](#_Toc17566746)

[I3S point cloud scene layer: nodepage 147](#_Toc17566747)

[I3S point cloud scene layer: node 155](#_Toc17566748)

[I3S point cloud scene layer: stats 156](#_Toc17566749)

[I3S point cloud scene layer: labels 168](#_Toc17566750)

[I3S point cloud scene layer: Statistics Label 170](#_Toc17566751)

[I3S point cloud scene layer: Bit Field Label 170](#_Toc17566752)

[Annex J: Examples of 3dSceneLayerInfo 171](#_Toc17566753)

**FIGURES**

[Figure 1: A Sample Index Tree with Treekeys 24](#_Toc17567075)

[Figure 2: Nodes and their attached resources 26](#_Toc17567076)

[Figure 3: This diagram illustrates the content of an I3S node as stored in its node index document 27](#_Toc17567077)

[Figure 4: Example illustrating the composition of an I3S tree for a 3D Object Indexed Scene Layer with a mesh pyramid profile. 28](#_Toc17567078)

[Figure 5: Logical Schema of the 3dSceneServiceInfo document 37](#_Toc17567079)

[Figure 6: Logical schema of the 3dSceneLayerInfo document 38](#_Toc17567080)

[Figure 7: Logical schema of the 3dNodeIndexDocument 58](#_Toc17567081)

[Figure 8: Logical schema of the FeatureData document 63](#_Toc17567082)

[Figure 9: Logical schema of the SharedResources document 70](#_Toc17567083)

[Figure 10: Geometry Buffer Layout with headers 77](#_Toc17567084)

[Figure 11: Example of the *fields* array resource 78](#_Toc17567085)

[Figure 12: A node resource document 80](#_Toc17567086)

[Figure 13: An expanded view of a scene layer resource 84](#_Toc17567087)

[Figure 14: A typical attribute (table) info of a feature class 85](#_Toc17567088)

[Figure 15: Example of a SLPK with BASIC folder layout 106](#_Toc17567089)

**TABLES**

[Table 1: 3D Layer Types Supported in I3S 18](#_Toc17567037)

[Table 2: *Example Level of Detail generation methods based on Scene Layer type* 33](#_Toc17567038)

[Table 3: Attributes of Class SceneServiceInfo within *SceneServiceInfo*  document 37](#_Toc17567039)

[Table 4: Attributes of the Class 3dSceneLayerInfo within the 3dSceneLayerInfo document 40](#_Toc17567040)

[Table 5: Attributes of the Class *Store* within the 3dSceneLayerInfo document 44](#_Toc17567041)

[Table 6: Attributes of the Class *Field* within the 3dSceneLayerInfo document 46](#_Toc17567042)

[Table 7: Attributes of the Class *attributeStorageInfo* within the 3dSceneLayerInfo document 49](#_Toc17567043)

[Table 8: Attributes of the Class *IndexScheme* within the 3dSceneLayerInfo document 50](#_Toc17567044)

[Table 9Attributes of the Class *CachedDrawingInfo* within the 3dSceneLayerInfo document 51](#_Toc17567045)

[Table 13*: Attributes of the Class Material within the 3dSceneLayerInfo document* 51](#_Toc17567046)

[Table 14*: Attributes of the Class Material within the 3dSceneLayerInfo document* 51](#_Toc17567047)

[Table 15*: Attributes of the Class Color within the 3dSceneLayerInfo document* 52](#_Toc17567048)

[Table 16*: Attributes of the Class CachedDrawingInfo within the 3dSceneLayerInfo document* 52](#_Toc17567049)

[Table 17: Attributes of the Class *Node* within the NodeIndexDocument 59](#_Toc17567050)

[Table 18: Attributes of the Class *NodeReference* within the NodeIndexDocument 59](#_Toc17567051)

[Table 19: Attributes of the Class *Resource* within the NodeIndexDocument 60](#_Toc17567052)

[Table 20: Attributes of the Class *Feature* within the NodeIndexDocument 61](#_Toc17567053)

[Table 21: Attributes of the Class *LodSelection* within the NodeIndexDocument 62](#_Toc17567054)

[Table 22: Attributes of the Class *Feature* within the FeatureData document 64](#_Toc17567055)

[Table 23: Attributes of the Class *FeatureAttribute* within the FeatureData document 64](#_Toc17567056)

[Table 24: Attributes of the Class *Geometry* within the FeatureData document 65](#_Toc17567057)

[Table 25: Attributes of the Class *GeometryReferenceParams* within the FeatureData document 66](#_Toc17567058)

[Table 26: Attributes of the Class *VestedGeometryParams* within the FeatureData document 67](#_Toc17567059)

[Table 27: Attributes of the Class *SingleComponentParams* within the FeatureData document 68](#_Toc17567060)

[Table 28: Attributes of the Class *Component* within the FeatureData document 69](#_Toc17567061)

[Table 29: Attributes of the Class *GeometryAttribute* within the FeatureData document 69](#_Toc17567062)

[Table 30: Attributes of the Class *Material* within the SharedResources document 71](#_Toc17567063)

[Table 31: Attributes of the Class *Texture* within the SharedResources document 72](#_Toc17567064)

[Table 32: Attributes of the Class Image within the SharedResources document 72](#_Toc17567065)

[Table 33: Attribute data types supported by a scene service layer. 85](#_Toc17567066)

[Table 34: example showing the layout of a SceneService 108](#_Toc17567067)



1. 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]](#footnote-2) .

1. 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 and Annex B (Bibliography) in this document are not found on the <https://github.com/Esri/i3s-spec> website. The Terms and Definitions and References sections may be added into the community GitHub repository in the future.

Please note that the Esri I3S Github repository includes a “Building Scene Layer (e.g. comprehensive building model including building components)”. This layer type is not included in this Community Standard but may be proposed for inclusion in a future version.

1. Validity of content

The Submission Team has reviewed and certified that the “snapshot” content in this Community Standard is true and accurate. The snapshot date is June 2019 in time for Esri User Conference. This version is also known as Esri version 1.6.

1. Keywords

The following are keywords to be used by search engines and document catalogues.

ogcdoc, OGC document, i3s, 3d, visualization, scene, scene layer, slpk

1. 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.*

1. Submitting organizations

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

Esri, Inc

1. Supporting Organizations

The following organizations support the submission of version 1.1 of I3S as an OGC Community Standard:

|  |  |
| --- | --- |
| Name | Affiliation |
| Keith Ryden | Esri |
| Allan W. Shearer | UT Austin |
| Volker Coors | Hochschule für Technik Stuttgart |
| 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 |

Note on supporting organizations. As per the OGC Technical Committee policies and Procedures:

Any <Community Standard> submission requires that three or more distinct Member organizations support the submission. In addition to the submission team lead, each other organization supporting the submission shall provide the TCC with an email stating their organization’s support of the submission.

Please note that all questions and/or comments regarding this OGC Community Standard should be submitted to the OGC via the comment submission form. [[insert link here when the OGC actually has one!]](https://github.com/cnreediii/ogc-i3s-community-1.1/blob/master/clause_0_front_material.adoc#insert link here when the OGC actually has one!)

1. 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 type may be considered for future inclusion in the I3S standard (future work):

* Building Scene Layer (e.g. comprehensive building model including building components)

# 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. Scene Layers provide clients access to data and allow them to visualize it according to their needs. The definition of "data" in this case includes the geometry, attributes, and vertex geometry.

A Scene Layer is characterized by a combination of layer type and profile. The layer type describes the kind of geospatial data stored within it. The layer profile includes additional details on the specific I3S implementation.

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](#_3D_Object_Scene) such as Building Exteriors from geospatial data and 3D models.
* [Integrated Meshes](#_Integrated_Scene_Mesh) such as a mesh surface with high resolution imagery textures representing the skin of the Earth, typically created from satellite, aerial or drone imagery.
* [Point Features](#_Point_Scene_Layer) such as geolocated hospitals or schools, trees, street furniture, and signs.
* [Point Clouds](#_Point_Cloud_Scene) such as large point data from LiDAR.

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]](#footnote-3). The specification for accessing I3S resources as Scene Service REST (Annex E) endpoints is also described in this standard as open formats.

# Conformance

Not required in an OGC Community Standard.

# References

**Normative**

OGC SF [99-036/ISO 19125]: Geographic information - Simple feature access - Part 1: Common architecture. 2005. https://www.iso.org/obp/ui#iso:std:iso:19162:ed-2:v1:en August 17, 2019.

OGC WKT CRS [12-063r5/ISO 19162:2019]: Geographic information — Well-known text representation of coordinate reference systems. 2019. <http://portal.opengeospatial.org/files/?artifact_id=4700> (May 15, 2017)

**Informative**

"Octree". *https://en.wikipedia.org/wiki/Octree*. Not Published (N.P.), 2016. Web. 20 Oct. 2016.

“Quadtree”. [*https://en.wikipedia.org/wiki/Quadtree*](https://en.wikipedia.org/wiki/Quadtree). N.P. 2017. Web. 20 Jan. 2017

“R-Trees”. [*https://en.wikipedia.org/wiki/R-tree*](https://en.wikipedia.org/wiki/R-tree)*.* N.P. 2017. Web. 20 Jan. 2017

# Terms and Definitions

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

1. 3D Model[[4]](#footnote-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.

1. Array Buffers

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

1. Community Standard

An OGC [Community Standard](http://www.opengeospatial.org/standards/community) is an official position of the OGC endorsing a specification or standard developed external to the OGC.

1. 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.

1. faceRanges

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

1. 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)

1. 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.

1. 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 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.

1. 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.

1. LiDAR

Light Detection and Ranging, is a remote sensing method that uses light in the form of a pulsed laser to measure ranges (variable distances) to the Earth.

1. Minimum Bounding Sphere[[5]](#footnote-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](https://en.wikipedia.org/wiki/Solid_sphere) containing all of these objects.

1. Near Infrared Light

Generally refers to light within the wavenumber range of 12,500 to 4,000 cm-1 (wavelengths from 800 to 2,500 nm) (<https://www.shimadzu.com/an/ftir/support/tips/letter9/nir1.html>, March 2019)

1. Normal or Normals[[6]](#footnote-6)

The normal vector, often simply called the "normal," to a surface is a [vector](http://mathworld.wolfram.com/Vector.html) which is [perpendicular](http://mathworld.wolfram.com/Perpendicular.html) 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.

1. Oriented Bounding Box (OBB)[[7]](#footnote-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. An oriented bounding box is simply a bounding parallelepiped whose faces and edges are not parallel to the basis vectors of the frame in which they're defined. 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.

1. Point Cloud

A point cloud is a set of data points in space. Point clouds are generally produced by 3D scanners, which measure a large number of points on the external surfaces of objects around them. (Wikipedia, <https://en.wikipedia.org/wiki/Point_cloud>, March 2019)

1. Profile

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

1. S3TC[[8]](#footnote-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.

1. 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.

1. 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.

1. Texture Atlas[[9]](#footnote-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.

1. Texture Mapping[[10]](#footnote-10)

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

1. 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.

1. UV Coordinate[[11]](#footnote-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.

1. UV Mapping[[12]](#footnote-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.

1. Vertex[[13]](#footnote-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.

# Conventions

No conventions are specified in this document.

# Introduction to I3S and SLPK

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

## 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: Provide a positive user experience, including high interactivity and fast display;
2. Scalability: Support very large scene layers, including scenes with a global extent and many detailed features;
3. Reusability: Use as a service delivery format, storage format, and exchange format;
4. Level of Detail: Have support for multiple levels of detail;
5. Distribution: Allow efficient distribution of very large data sets;
6. User-controllable symbology: Support efficient rendering of client-side symbology/styling;
7. Extensibility: Support new layer, geometry types and new platforms;
8. Web Friendliness: Provide easy to handle data using JSON and current web standards;
9. Compatibility: Provide a single structure that is compatible across web, mobile, and desktop clients. Support is also included for cloud and servers;
10. Declarative: Communicate clearly to minimize the amount of required domain knowledge to support the format;
11. Follow REST/JSON API best practices: Provide navigable links to all resources.

## I3S – Overview

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 provide clients access to data and allow them to visualize it 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. The layer type describes the kind of geospatial data stored within it. The layer profile includes additional details on the specific I3S implementation.

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

* [3D Object Scene Layer](#_3D_Object_Scene) (e.g., Building Exteriors from geospatial data and 3D models)
* [Integrated Mesh](#_Integrated_Scene_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](#_Point_Scene_Layer) (e.g. hospitals or Schools, trees, street furniture, signs, etc. from GIS data)
* [Point Clouds](#_Point_Cloud_Scene) (e.g. LiDAR 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, Points, Lines, Polygons, and Point Clouds. 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 Objects | mesh-pyramids | [Annex H](#AnnexH) | Yes | Yes |
| Integrated Mesh | mesh-pyramids | [Annex H](#AnnexH) | No | Triangle Attributes (planned) |
| Point | points | [Annex G](#AnnexG)[[14]](#footnote-14) | Yes | Yes |
| Point Cloud | pointclouds | Annex I | No | Vertex Attributes |

Table 1: 3D Layer Types Supported in I3S

# I3S Specification

This section contains the normative clauses and requirements for implementing I3S. In the property tables, any property highlighted with a **bold** font is mandatory.

## Coordinate Reference Systems (CRS)[[15]](#footnote-15)

The I3S specification supports specifying the Coordinate Reference System (CRS) as OGC Well Known Text, as originally defined in clause 6.4 in OGC Simple Features [99-036/ISO 19125](http://portal.opengeospatial.org/files/?artifact_id=13227) standard. I3S also supports specifying CRS in the ISO/OGC WKT standard ISO 19162:2019, Geographic information – Well-known text representation of coordinate reference systems. This new ISO/OGC Standard specifies an update to the original WKT representation. The two standards are referred to as WKT1 and WKT2 respectively.

### A note on OGC Standards for CRS and 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]](#footnote-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]](#footnote-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]](#footnote-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 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).***

### 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 a global extent
* Render easily in coordinate reference systems for projected[[19]](#footnote-19) CRSs as well as for geographic[[20]](#footnote-20) CRSs
* Support local and global data with very high positional accuracy.

These use cases lead to the following implementation requirements. Note that all I3S profiles support writing 3D content in two modes: global and local. In global mode, only the geographic CRS WGS84 (EPSG 4326) is supported for both index and vertex positions.

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](#_Terms_and_Definitions) (MBS) radius *SHALL* be specified in meters. Allowed CRS specified using an EPSG code is EPSG:4326[[21]](#footnote-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 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 system used by the layer with the [spatialReference](#_spatialReference) property in the [3dSceneLayerInfo](#_3dSceneLayerInfo_[Common_Profiles]) resource. This property is normative.

The spatial reference object is common to all I3S profile types.

## 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*)). These 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\",6378137,298.257223563]],PRIMEM[\"Greenwich\",0],UNIT[\"Degree\",0.017453292519943295]],

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]]​]

The following is an example for 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](#_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](#_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](#_3dSceneLayerInfo)) for more information on the use of the heightModelInfo object.

## 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.

### 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*. Each node has roughly equal amounts of data, and organized into a hierarchical and navigable data structure. The node index allows the client to quickly determine which data it actually needs and allows the server to quickly locate the data requested by any client. Node creation is capacity driven. For example, the smaller the node capacity is, typically the smaller the spatial extent.

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]](#footnote-22) and Octrees[[23]](#footnote-23)) as well as density dependent partitioning of space (e.g. R-Trees[[24]](#footnote-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. A node has an ID that is unique within a layer. I3S supports two types of node ID formats: string based treekeys or as integers based on a fixed linearization of the nodes.

The *treekey* format is loosely modeled on binary search trees. The key indicates both the level and sibling association of a given node. The key also 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. The root node is at level 1 always gets ID root. For example, take the node with treekey "3-1-0". Since this key has 3 numeric elements 3, 1, and 0, we can conclude that the node is on level 4 ("root" node is at level 1) and the parent node is "3-1". An example of this numbering pattern is shown in Figure 1 below.

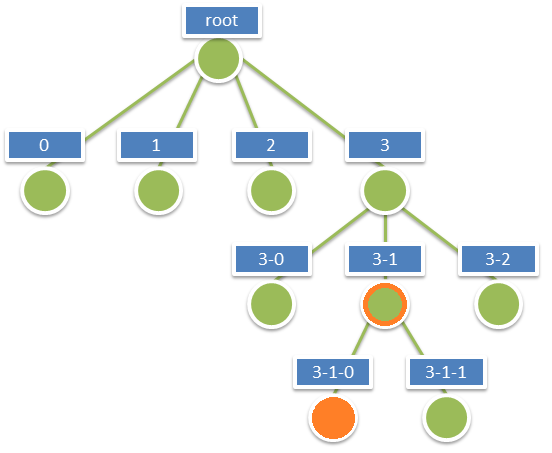


Figure 1: A Sample Index Tree with Treekeys

For example, take the node with treekey "3-1-0". Since it has 3 numeric elements 3, 1 and 0, it can be concluded that the node is on level 4 (The root node is at level 1). Furthermore, the node "3-1" is its parent node.

The information for a node is stored in multiple individually accessible resources. The node index document is a lightweight resource that captures the Bounding Volume Hierarchy (BVH) tree topology for the node. Key components of the document include the node’s bounding volume information, meta-data used for (LoD Switching Models) metrics, as well as parent-child relationships. The node index 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 a node by the client application is based on node’s bounding-volume visibility in the current 3D view. Once the node’s bounding-volume visibility is determined to be within the current 3D view of the application, then further evaluation is performed by the client application to determine the visual quality of the node. This determination is done 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](#_Terms_and_Definitions) (MBS) and [oriented bounding boxes](#_obb) (OBB) as a node’s bounding volume.

Each node includes the set of information covered by the nodes below it and is part of the path of the leaf nodes below it. Interior nodes may have a reduced representation of the information contained in descendant nodes.

The I3S format models node information using a set of resources including NodeIndex Document, FeatureData, Geometry, Attributes, Textures and SharedResource. All these together represent the set of features or data elements for a given node. These resources are always attached to a node.

* *Node Index Document*: A lightweight resource representing a node, its topology within the tree and includes references to other sub-resources.
* *Feature Data:* A text sub-resource for a node 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 uses either 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 that use the separate binary sub-resources do not need to first fetch the feature-data resource in order to interpret them. All binary data is stored using a little-endian byte ordering.

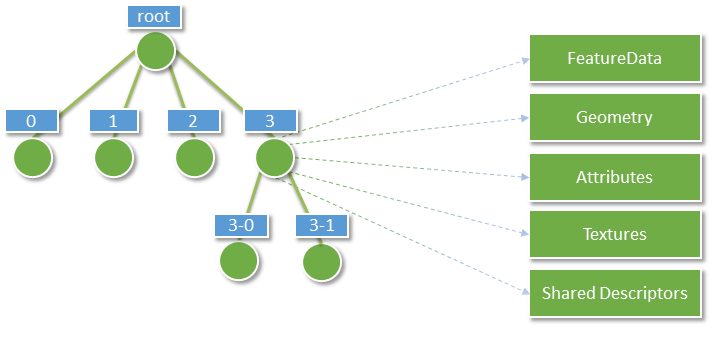
[](https://github.com/Esri/i3s-spec/blob/master/format/images/figure-02.png)

Figure 2: Nodes and their attached resources

Each node has exactly one NodeIndexDocument and one SharedDescriptors document. The 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 of FeatureData and Geometry resources. 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.

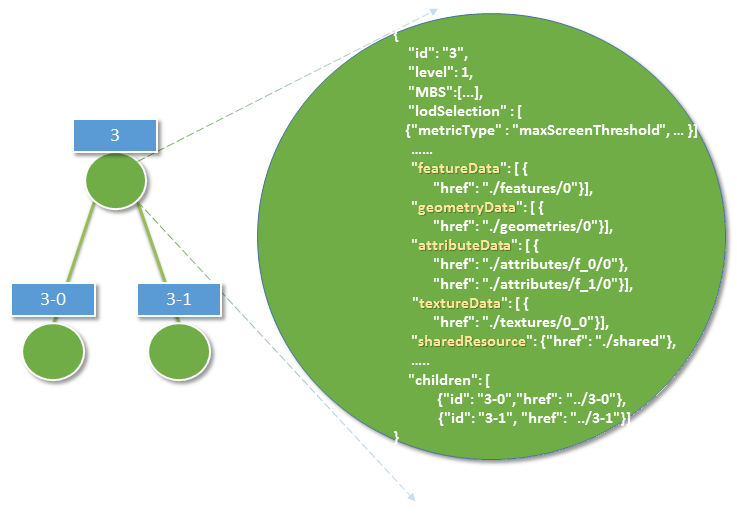
[](https://github.com/Esri/i3s-spec/blob/master/format/images/figure-03.png)

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 circles.
* Node Identifiers are in blue boxes above a node and represent the identifier or address for each node.
* Features are in orange rectangles with each node. 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 geometry resource associated with each node is represented by the turquoise boxes. 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]](#footnote-25).
* Note: 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.
* 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](#_Class_LodSelection)) 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.

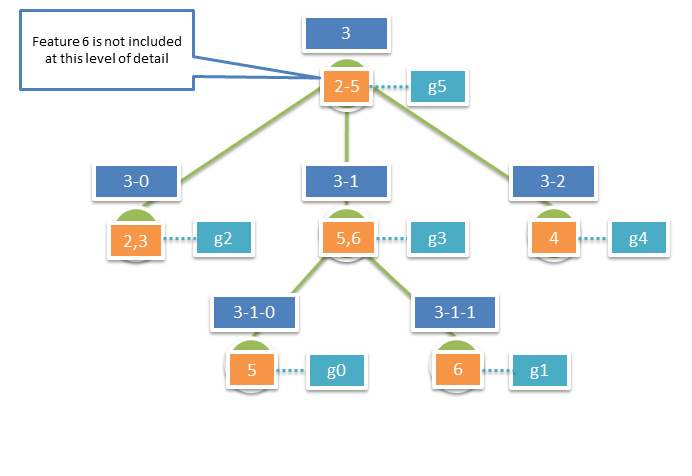
[](https://github.com/Esri/i3s-spec/blob/master/format/images/figure-04.png)

Figure 4: Example illustrating the composition of an I3S tree for a 3D Object Indexed Scene Layer with a mesh pyramid profile.

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.*

### Geometry Model and Storage

All Scene Layer types make use of the same fundamental set of geometry types: points, lines, and triangles.

Array Buffer View[[26]](#footnote-26) geometry property declarations control geometries storage and consumption representation. I3S provides full control over those properties, such as per-vertex layout of components (e.g. position, normal and texture coordinates). This orders the vertex position, normal and texture coordinates to ensure the same pattern 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.

For more details regarding 3D objects and point scene layer, see [Geometry](#_Geometry).

For more details regarding point cloud scene layer, see [defaultGeometrySchema](#_defaultGeometrySchema).

### 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. Both integrated mesh and 3D object profile support textures. Authoring applications can provide additional texture formats using textureEncoding declarations.

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]](#footnote-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](#_Textures.bin) section for more on texture format, texture coordinate, texture atlas usage and regions discussion.

### Attribute Model and Storage

I3S supports the following two patterns of accessing attribute data. They can be accessed:

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 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 the I3S store. I3S clients can still choose to use both of these modes even if the attributes are fully cached within I3S store. The binary storage representation provides a significant performance benefit

Clients can use either method if the attributes are cached. The attribute values are stored as a geometry aligned, per field (column), key-value pair arrays.

For more details regarding point cloud scene layer, see AttributeInfo.

For more details on all other scene layer types, see [Attribute](#_Attribute_Data).

## Bounding Volume Hierarchy

Bounding volume hierarchy (BVH) is based on minimum bounding sphere (MBS) or oriented bounding box (OBB). The mesh-pyramids profile supports specifying the bounding volume in either MBS or OBB representation. OBB is the more optimal representation and implementers are encouraged to output node bounding volume in OBB format. Point cloud profile supports OBB representation only.

For more details regarding the two types of bounding volumes see minimum bounding box and [oriented bounding box](#_obb) sections.

## Level of Detail

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. This is similar to image pyramids and also similar to raster and tiled vector data 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. Levels of Detail can be used to split heavy features, thin or cluster for better visuals, and integrate externally authored files.

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 (batches) for a set of geometries within a level of detail. This batching is 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, preserving the feature to geometry element mapping within the consolidated geometries.

### Discrete LoDs

I3S supports a *Discrete* LoD approach, where different Level of Detail 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.

By using only information found in the node index document, such as bounding volume and level of detail selection metrics, a client application traversing an I3S tree can readily decide if it needs to:

1. Stop 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](#_LoD_Selection_Metrics) and permits different [LoD Switching Models](#_LoD_Switching_Modes). 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.

### 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 levels of detail can optionally be automatically generated by extending the viewing range of each semantically input level of detail.

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 depth of the I3S index tree is fixed to the number of levels of detail present in the input. 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.

### LoD Switching Modes

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

#### Node Switching

Node switching lets clients focus on the display of a node as a whole. A node switch occurs when the content from a node's children is used to replace the content of an existing node. This can include features, geometry, attributes and textures. Node switching can be helpful when the user needs to see 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. Due to generalization at lower Levels of Detail, not all features are present in reduced level of detail 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 feature IDs. These 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.

### 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 a 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, the method used to create the levels depends on the Scene Layer type..

|  | **3D Object** | **Points** | **Point Clouds** | **Building Scene Layer** |
| --- | --- | --- | --- | --- |
| Mesh-pyramids |  |  |  |  |
| Thinning |  |  |  |  |
| Clustering |  |  |  |  |
| Generalization |  |  |  |  |

Table 2: Example Level of Detail generation methods based on Scene Layer type

### LoD Selection Metrics

A client needs information to determine whether a node's contents are "good enough" to render in the current 3D view. This metric can be used by the client to determine whether a representation is of the correct quality. Publishers can add as many LodSelection objects as desired but must provide one if the layer's lodType is not null. Of the three min/avg/max values, typically only one or two are used. Selection criteria include constraints such as resolution, screen size, bandwidth and available memory and target minimum quality goals.

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| **metricType** | string | The name of the error metric, one of {maxScreenThreshold, maxScreenThresholdSQ,screenSpaceRelative, distanceRangeFromDefaultCamera, maxScreenThresholdSQ, effectiveDensity } |
| **maxError** | number | Maximum metric value, expressed in the CRS of the vertex coordinates or in reference to other constants such as screen size. |

Multiple LoD selection metrics can be included, as in the following example:

"lodSelection": [

{

"metricType": "maxScreenThreshold",

"maxError": 20.530693054199219

},

"metricType": "maxScreenThresholdSQ",

"maxError": 331.05267333984375

},

{

"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: 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). Typically, a client application consuming a node resource will project the nodes bounding volume (in this case sphere) on screen plane and compute its radius in pixels. The application can then switch the LoD to children node if this radius is bigger than the value defined for the maxError of the maxScreenThreshold metric. Used by mesh pyramids.

maxScreenThresholdSQ: is the metric type used when the bounding volume of a node is Oriented Bounding Box (OBB). This metric is equivalent to maxScreenThreshold and is calculated as:

maxScreenThresholdSQ = PI \* 0.25 \* maxScreenThreshold \* maxScreenThreshold

screenSpaceRelative: The scale of the node's minimum bounding volume. used by the points profile.

distanceRangeFromDefaultCamera: Normalized distance of the node's minimum bounding volume from the camera. Used by the points profile.

effectiveDensity: Estimation of the point density covered by the node. Used by Point Clouds.

## JSON Resources Schema and Documentation

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

### Basic Value Types

A value schema ensures that the JSON properties follow a fixed pattern and support the following data types:

* String: 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" time zone, 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").

### 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)

### 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 data available in some data store in which 3D content has been authored and is ready for publication (visualization). This file is automatically generated by the Scene Server for each service instance and is not part of a Scene Layer Package (SLPK) file.

The SceneServiceInfo has the following structure:

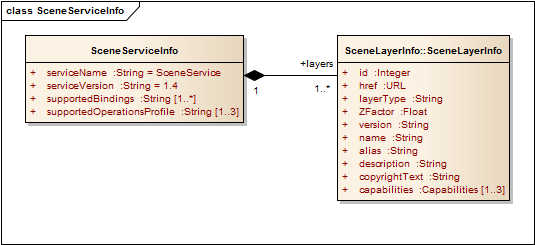


Figure 5: Logical Schema of the 3dSceneServiceInfo document

This file is automatically generated by a Scene Server for each service instance and is not part of a Scene Layer Package (SLPK) file. It is included here only for reference.

#### Class SceneServiceInfo

SceneServiceInfo is the major object in the 3dSceneServiceInfo document. The SceneServiceInfo file is a JSON file that describes the capability and data sets offered by an instance of a Scene Service. There SHALL always be exactly one SceneServiceInfo object in the document. This document describes an active 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. |
| supportedOperations | String[1..3] | Supported profiles of the service from the choice {Base, Dynamic, Editing}. |
| layers | 3dSceneLayerInfo[1..\*] | The full [3dSceneLayerInfo](#_3dSceneLayerInfo) information. |

Table 3: Attributes of Class SceneServiceInfo within SceneServiceInfo document

For service examples, see Annex E - Scene Service Access to REST Resources.

### 3dSceneLayerInfo [Common Profiles]

The Class 3dSceneLayerInfo describes the properties of a single layer in a store, including the default symbology to use. Every scene layer contains 3DSceneLayerInfo. 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:

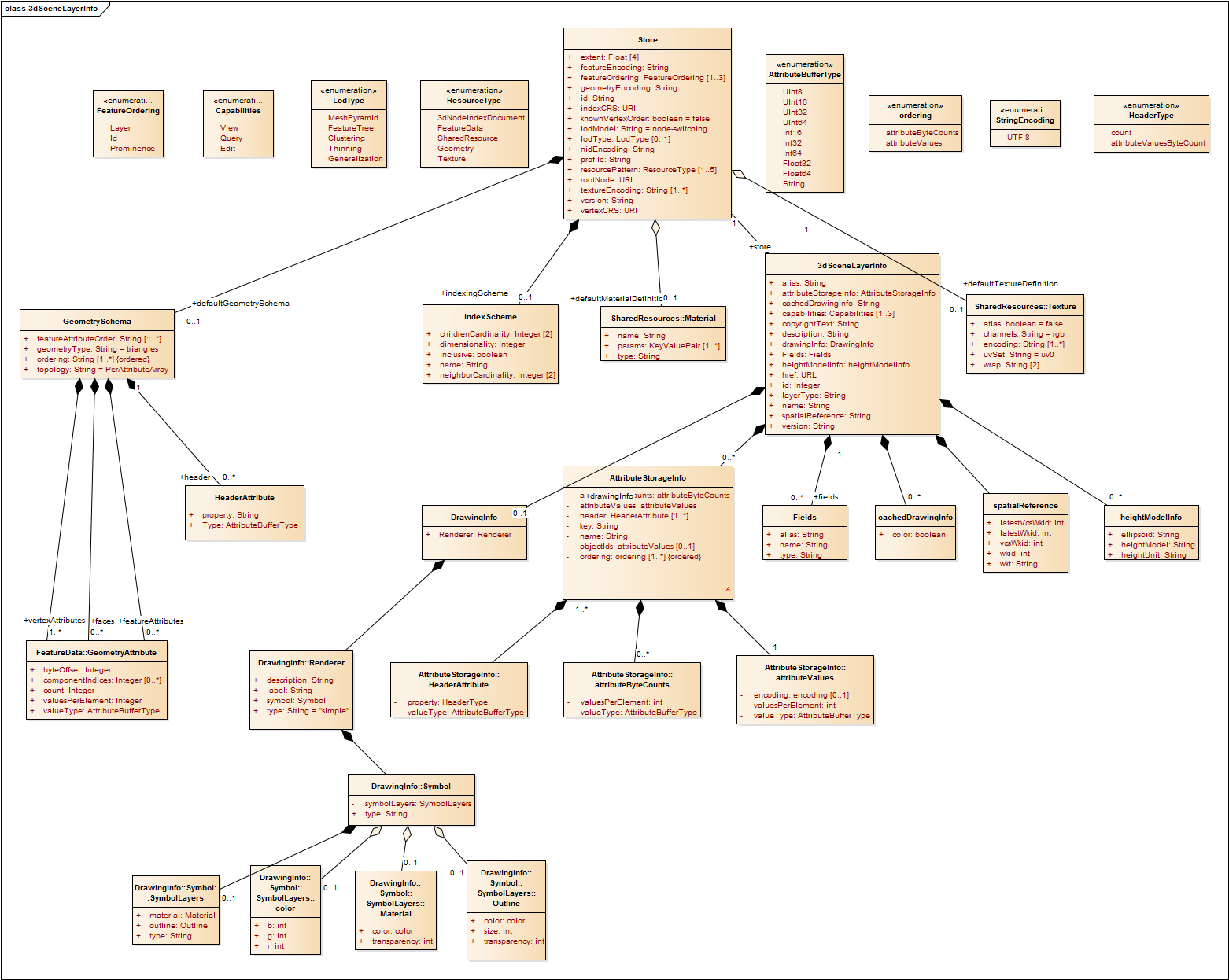


Figure 6: Logical schema of the 3dSceneLayerInfo document

#### Class 3dSceneLayerInfo (Common Profiles)

The 3dSceneLayerInfo is a major object in the 3dSceneLayerInfo document. A SceneServiceInfo document can contain 1...\* 3dSceneLayerInfo documents. Each 3dSceneLayerInfo object describes a Layer. For features based scene layers, such as 3D objects or point scene layers, may include the default symbology, as specified in the sub-class *drawingInfo*, which contains stylization information for a feature 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 layer type, one of {Point, Line, Polygon, 3DObject, PointCloud, 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. |
| serviceUpdateTimeStamp | *serviceUpdateTimeStamp* | The time of the last update |
| 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 supported by this layer.  Possible values for each array string:  View: View is supported.  Query: Query is supported.  Edit: Edit is defined. |
| ZFactor | number | ZFactor to define conversion factor for elevation unit. |
| cachedDrawingInfo | ***CachedDrawingInfo*** | Indicates if any stylization information represented as *drawingInfo* is additionally captured as part of the binary mesh representation. This helps for optimal client side access. Currently color component of the *drawingInfo* is supported. |
| drawingInfo | ***Class DrawingInfo*** | Represents the stylization information of the layer. |
| elevationInfo | ***elevationInfo*** | An object containing elevation drawing information. If absent, any content of the scene layer is drawn at its z coordinate. |
| popupinfo | ***popupInfo*** | PopupInfo of the scene layer. |
| disablePopup | boolean | Indicates if client application will show the popup information. |
| **store** | ***Class Store*** | The 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. |
| statisticsInfo | ***statisticsInfo***[] | Contains the statistical information for a layer. |
| fields | ***Class Field***[] | 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 | Class ***attributeStorageInfo*** | Provides the schema and layout used for storing attribute content in binary format in I3S. |

Note: properties in ***bold*** are required

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

See Annex J for Examples.

#### Class Store

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

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| id | String | A Store ID, unique across a SceneServer. Enables the client to discover which layers a part of a common store, if any. {meshes, polygons, points, lines, analytics, meshpyramids, pointclouds, symbols} |
| **profile** | String | Indicates which profile this scene store fulfills. One of {meshes, points, analytics, meshpyramids, symbols, PointCloud}. |
| resourcePattern | String [] | Indicates the resources needed for rendering and the required order in which the client should load them.  Possible values for each array string:   * 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, its spatial extent. * SharedResource: Shared resources are models or textures that can be shared among features within the same layer. * 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. * Geometry: Each geometry resource is an array of geometries. * Texture: The texture resource for a node contains the images that are used as textures for the features stored in the node. * Attributes: Attribute resource for node containing feature data attributes |
| rootNode | string | 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 | Number[4] | The 2D spatial extent (xmin, ymin, xmax, ymax) of this store, in the horizontal indexCRS |
| indexCRS | String | The horizontal CRS used for all minimum bounding spheres (mbs) in this store, identified by an OGC URL. |
| vertexCRS | String | 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.  Possible values are:   * east-north-up: 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. * earth-centered: 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. * vertex-reference-frame: 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 | string | MIME type for the encoding used for the Node Index Documents; format: application/vnd.ogc.I3S.json+gzip; version=1.6 |
| featureEncoding | string | MIME type for the encoding used for the Feature Data Resources; format: application/vnd.ogc.I3S.json+gzip; version=1.6 |
| geometryEncoding | string | MIME type for the encoding used for the Geometry Resources; format: application/octet-stream; version=1.6 |
| textureEncoding | string[] | 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.  Possible values are:   * MeshPyramid: Used for integrated mesh and 3D Object Scene layers, which share similar data partitioning as well as traversal patterns and hence belong to the same profile. * AutoThinning: Used by point scene layer. Indicates the I3S generator (‘cooker’) used automatic data thining techniques to build interior nodes (non-leaf nodes). * Clustering: Used by point scene layer. Indicates the ‘cooker’ used data clustering algorithims to build interior nodes (non-leaf nodes). * Generalizing: Also used by point scene layer. Indicates the ‘cooker’ used data generalization algorithims/techniques to build interior nodes (non-leaf nodes). |
| lodModel | String | optional field to indicate which LoD Switching mode clients have to use. One of {\*node-switching\*, none}. |
| **defaultGeometrySchema** | [***defaultGeometrySchema***](#_defaultGeometrySchema) | 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 | [***texture***](#_Class_Texture) | A common, global TextureDefinition 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 | [***material***](#_Class_Material) | If a store uses only one material, it can be defined here entirely as a MaterialDefinition |

Note: properties in ***bold*** are required

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

For more details regarding point scene layer, see the store point scene layer.  
For more details regarding point cloud scene layer, see the store point cloud scene layer.

#### defaultGeometrySchema

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 and 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[] | Declares the topology of embedded geometry attributes. When 'Indexed', the indices must also be declared in the geometry schema ('faces') and precede the vertexAttribute data.  Possible values are:   * PerAttributeArray * Indexed: When Indexed, the indices must also be declared in the geometry schema (faces) and precede the vertexAttribute data. |
| **header** | [***HeaderAttribute***](#_Class__HeaderAttribute)[] | Defines header fields in the Geometry resources of this store that precede the vertex (and index) data |
| **ordering** | String[] | Provides the order of the keys in vertexAttributes and faceAttributes, if present. |
| **vertexAttributes** | [***vertexAttributes***](#_VertexAttributes) | Declaration of the attributes per vertex in the geometry, such as position, normals or texture coordinates |
| faces | [***vertexAttributes***](#_VertexAttributes) | Declaration of the indices into vertex attributes that define faces in the geometry, such as position, normals or texture coordinates |
| **featureAttributeOrder** | String[] | Provides the order of the keys in featureAttributes, if present. |
| **featureAttributes** | geometryFeature | Declaration of the attributes per feature in the geometry, such as feature ID or face range |

Note: properties in ***bold*** are required

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

For more details regarding point scene layer, see the default geometry schema point cloud scene layer.

#### 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} |

Note: properties in ***bold*** are required

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

**Example**

{

"property": "vertexCount",

"type": "UInt32"

}

#### 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[] | The display alias to be used for this field. |
| domain | ***domain*** | Array of domains defined for a field. |

Table 6: Attributes of the Class *Field* within the 3dSceneLayerInfo document

The following is a JSON example of the ***field*** class

{

"name": "CreatedPhase",

"type": "FieldTypeInteger",

"alias": "CreatedPhase",

"domain": {

"type": "codedValue",

"name": "Phases",

"description": "Phases",

"codedValues": [

{

"name": "Existing",

"code": 0

},

{

"name": "Baby Room Overhaul",

"code": 1

},

{

"name": "Roof Garden",

"code": 2

}

],

"mergePolicy": "esriMPTDefaultValue",

"splitPolicy": "esriSPTDefaultValue"

}

}

##### domain

(I3S Attribute (i.e. Field) domain)

Attribute domains are rules that describe the legal values of a field type, providing a method for enforcing data integrity. Attribute domains are used to constrain the values allowed in a particular attribute. Using domains helps ensure data integrity by limiting the choice of values for a particular field. Attribute domains can be shared across scene layers like 3D Object Scene Layers or Building Scene Layers.

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| **type** | string | Type of domain  Possible values are:   * codedValue * range |
| **name** | string | Name of the domain. Must be unique per Scene Layer. |
| description | string | Description of the domain |
| fieldType | string | The field type is the type of attribute field with which the domain can be associated.  Possible values are:   * FieldTypeDate * FieldTypeSingle * FieldTypeDouble * FieldTypeInteger * FieldTypeSmallInteger * FieldTypeString |
| range | number[2] | Range of the domain (numeric types only) |
| codedValues | ***domainCodedValue***[] | Range of the domain (string types only) |

*Note: properties in* ***bold*** *are required*

A range domain specifies a valid range of values for a numeric attribute. When creating a range domain, you enter a minimum and maximum valid value. A range domain can be applied to short-integer, long-integer, float, double, and date attribute types.

A coded value domain can apply to any type of attribute—text, numeric, date, and so on. Coded value domains specify a valid set of values for an attribute.

The following is a JSON example of a *domain* encoding.

{

"type": "codedValue",

"name": "Phases",

"description": "Phases",

"codedValues": [

{

"name": "Existing",

"code": 0

},

{

"name": "Baby Room Overhaul",

"code": 1

},

{

"name": "Roof Garden",

"code": 2

}

],

}

##### domainCodedValue

Quite often, in a controlled vocabulary, such as in a land use classification, coded values are used represent categories of a feature. For example, “Res” could refer to Residential. The *domainCodedValue* class allows for the specification of these codes values.

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| **name** | string | Text representation of the domain value. |
| **code** | string, number | Coded value (i.e. field value). |

**Example 1**

{

"name": "code 1.5 description",

"code": 1.5

}

**Example 2**

{

"name": "coded 3000.1 desc",

"code": 3000.3

}

#### 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.

For more details regarding point cloud scene layer, see attributeInfo.

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| **key** | string | The unique field identifier key. |
| **name** | string | The name of the field. |
| **header** | [***headerValue***](#_value)[] | 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[] | 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 | ***value*** | The element type of the attributeByteCounts property, from {UInt32}. |
| attributeValues | ***value*** | The element type of the attributeValues property, from {UInt8, UInt16, UInt32, UInt64, Int16, Int32, Int64 or Float32, Float64} |
| objectIds | ***value*** | Stores the object-id values of each feature within the node. |

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

{

"key": "f\_2",

"name": "Family",

"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

}

}

Example: attributeStorageInfo for 3d object scene layer

#### Class IndexScheme (deprecated)

**Warning: This class has been deprecated in version 1.1.** This class is no longer supported. Use at your own risk. This class will remain in the I3S Community Standard until a version two of the standsrd is approved for publication. At that time this clause will be removed.

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 8: Attributes of the Class *IndexScheme* within the 3dSceneLayerInfo document

#### 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** | ***Class Renderer*** | The renderer object encapsulates the drawing information of the layer. |

Table 9Attributes of the Class *CachedDrawingInfo* within the 3dSceneLayerInfo document

For more details regarding point cloud scene, see drawing info point cloud scene layer.

##### 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 13*: Attributes of the Class Material within the 3dSceneLayerInfo document*

##### 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 14*: Attributes of the Class Material within the 3dSceneLayerInfo document*

##### 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 in this I3S layer will be drawn. |

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

##### 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 16*: Attributes of the Class CachedDrawingInfo within the 3dSceneLayerInfo document*

#### spatialReference

The spatialReference object is located at the top level of the JSON hierarchy. A spatial reference can be defined using a well-known ID (WKID) or well-known text (WKT). The default tolerance and resolution values for the associated coordinate system are used.

A spatial reference can optionally include a definition for a vertical coordinate system (VCS), which is used to interpret the z values in the geometry.

Note: Please see detailed CRS discussion with examples [here](#_Coordinate_Reference_Systems).

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| latestVcsWkid | integer | The current WKID value of the vertical coordinate system. |
| latestWkid | integer | Identifies the current WKID value associated with the same spatial reference. For example a WKID of '102100' (Web Mercator) has a latestWKid of '3857'. |
| vcsWkid | integer | The WKID value of the vertical coordinate system. |
| wkid | integer | The well-known ID (WKID) of the coordinate system. Specify either WKID or the well-known text (WKT) of the coordinate system. |
| wkt | string | The well-known text (WKT) of the coordinate system. Specify either WKT or WKID of the coordinate system (but not both) |

#### heightModelInfo

The I3S standard accommodates declaration of a vertical coordinate system that may either be ellipsoidal or gravity-related. This allows for a diverse range of fields and applications where the definition of elevation/height is important.

Note: Please see detailed heightInfo discussion with examples [here](#_Height_Models).

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| heightModel | string | Represents the height model type.  Possible values are:   * gravity\_related\_height * ellipsoidal |
| vertCRS | string | Represents the vertical coordinate system. |
| heightUnit | string | Represents the unit of the height.  Possible values are:   * meter * us-foot * foot * clarke-foot * clarke-yard * clarke-link * sears-yard * sears-foot * sears-chain * benoit-1895-b-chain * indian-yard * indian-1937-yard * gold-coast-foot * sears-1922-truncated-chain * us-inch * us-mile * us-yard * millimeter * decimeter * centimeter * kilometer |

#### elevationInfo

The elevationInfo defines how content in a scene layer is aligned to the ground. For example, the feature is on the ground or at an absolute height.

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| mode | string | The mode of the elevation. Possible values are:   * relativeToGround * absoluteHeight * onTheGround * relativeToScene |
| offset | number | Offset is always added to the result of the above logic except for onTheGround where offset is ignored. |
| unit | string | A string value indicating the unit for the values in elevationInfo |

#### popupInfo

The properties in the popupInfo class define the look and feel of popup windows when a user clicks or queries a feature.

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| title | string | A string that appears at the top of the popup window as a title |
| description | string | A string that appears in the body of the popup window as a description. It is also possible to specify the description as HTML-formatted content. |
| expressionInfos | [] | Reserved for future use. |
| fieldInfos | [] | Array of fieldInfo information properties. This information is provided by the service layer definition. A fieldInfo class defines a set of properties that relate to the [Attribute Data](#_Attribute_Data) of an I3S layer.  The fieldInfo class contains properties such as 'fieldName'{the name of the field the fieldinfo relates to}, 'isEditable' {indicates if the fieldInfo property is editable}, lable {typically used to store a user-friendly name} and visible {indicates if this fieldInfo should be visible}.  The fieldName of a fieldInfo class directly refrences the 'name' object of the fields array (layers[id].fields[id]) as declared in the [3dSceneLayerInfo](#_Class__3dSceneLayerInfo) class. |
| mediaInfos | [] | Array of various mediaInfo to display. Can be of type image, piechart, barchart, columnchart, or linechart. The order given is the order in which it displays. |
| popupElements | [] | An array of popupElement objects that represent an ordered list of popup elements |

#### serviceUpdateTimeStamp

This object provides a time stamp about when the I3S service or the source of the service was created or updated.

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| **lastUpdate** | number | Specifies the moment in milliseconds when the I3S service was created or updated. |

*Note: properties in* ***bold*** *are required*

**Example**

{

"lastUpdate": 1518827901690

}

#### statisticsInfo

This class is used to describe the statistics for the scene layer. Statistical information helps clients define symbology, definition queries or other functionality which is depending on statistical information.

For more details regarding point cloud scene layers, see ***statistics***.

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| **key** | string | Key indicating the resource of the statistics. For example f\_1 for ./statistics/f\_1 |
| **name** | string | Name of the field of the statistical information. |
| **href** | string | The URL to the statistics information. For example ./statistics/f\_1 |

*Note: properties in* ***bold*** *are required*

**Example: statisticsInfo for 3D Object scene layer.**

{

"key": "f\_1",

"name": "Category",

"href": "./statistics/f\_1"

}

#### VertexAttributes

VertexAttributes describe valid properties for a single vertex.

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| **position** | *Class GeometryAttribute* | Vertex position |
| **normal** | *Class GeometryAttribute* | vertex normal |
| **uv0** | *Class GeometryAttribute* | First set of UV coordinates |
| **color** | *Class GeometryAttribute* | Colors attribute |
| region | *Class GeometryAttribute* | Regions attribute |

*Note: properties in* ***bold*** *are required*

**Example**

{

"position": {

"byteOffset": 8,

"valueType": "Float32",

"valuesPerElement": 3

},

"normal": {

"byteOffset": 2672,

"valueType": "Float32",

"valuesPerElement": 3

},

"uv0": {

"byteOffset": 5336,

"valueType": "Float32",

"valuesPerElement": 2

},

"color": {

"byteOffset": 7112,

"valueType": "UInt8",

"valuesPerElement": 4

}

}

#### value

Value declares the headers of the binary attribute data. Must be 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.

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| **valueType** | string | Defines the value type. |
| encoding | string | Encoding method for the value. |
| valuesPerElement | number | Number of values per element. |

### 3dNodeIndexDocument

The 3dNodeIndexDocument JSON file describes a single index node within a store. This includes 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, provides sufficient data for simple visualization by rendering the centroids as point features for example. This can help the user understand the overall distribution of the data.

The 3dNodeIndexDocument has the following structure:

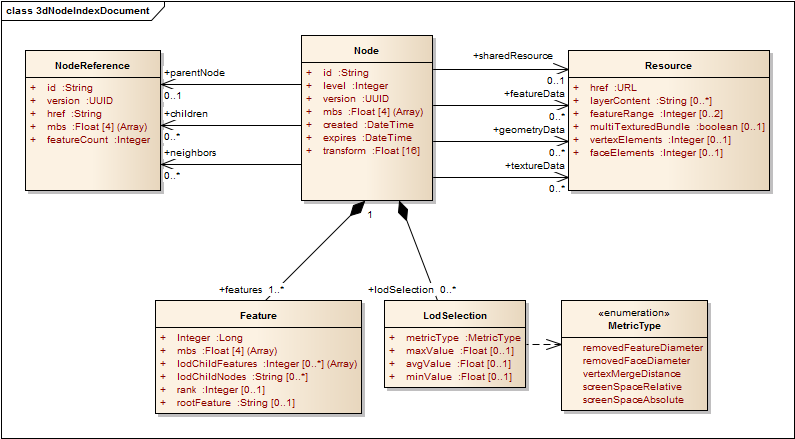


Figure 7: Logical schema of the 3dNodeIndexDocument

#### Class 3dNodeIndexDocument

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 | string | The version (store update session ID) of this node. |
| mbs | Number[4] | An array of four doubles, corresponding to x, y, z and radius of the minimum bounding sphere of a node. |
| obb | [***obb***](#_obb) | Describes oriented bounding box. |
| created | String | 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 | String | 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 | Number[16] | Optional, 3D (4x4) transformation matrix expressed as a linear array of 16 values. |
| parentNode | *nodeReference* | Reference to the parent Node of a Node. |
| children | *nodeReference* [] | Reference to the child Nodes of a Node. |
| neighbors | *nodeReference* [] | Reference to the neighbor (same level, spatial proximity) Nodes of a Node. |
| sharedResource | *Resource* | Resource reference describing a shared resource document. |
| featureData | resource[] | Resource reference describing a FeatureData document. |
| geometryData | resource[] | Resource reference describing a geometry resource. |
| textureData | resource[] | Resource reference describing a texture resource. |
| attributeData | resource[] | Resource reference describing a FeatureData document. |
| lodSelection | [***lodSelection***](#_Class_LodSelection)[] | Metrics for LoD Selection, to be evaluated by the client. |
| features | [***lodSelection***](#_Class_LodSelection)[] | A list of summary information on the features present in this Node, used for pre-visualisation and LoD switching in featureTree LoD stores. |

Note: properties in ***bold*** are required

Table 17: Attributes of the Class *Node* within the NodeIndexDocument

#### nodeReference

Class nodeReference specifies properties for a pointer to another node , which could be 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 | number[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 | number | Number of features in the referenced node and its descendants, down to the leaf nodes. |
| obb | obb | Describes oriented bounding box. |

Note: properties in ***bold*** are required

Table 18: Attributes of the Class *NodeReference* within the NodeIndexDocument

#### 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[] | 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 | Number[] | 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 | String | 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 | number[] | 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 | number[] | 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. |

Note: properties in ***bold*** are required

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

**Example**

{

"href": "./features/0",

"featureRange": [

0,

3

]

}

#### Class Feature (Deprecated)

**Warning: This Class has been deprecated for version 1.1.** This Class is no longer supported. The properties specified in this class have been integrated into the 3dNodeIndexDocument as [FeatureData](https://github.com/cnreediii/ogc-i3s-community-1.1/blob/master/clause_7_normative_text.adoc#featuredata). This deprecated class will remain in the OGC I3S Community Standard until such time as Version 2 of this Community Standard is approved. At that time, this deprecated class will be removed from the standard.

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** | Number | 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 | Number[4] | An array of four doubles, corresponding to x, y, z and radius of the minimum bounding sphere of the referenced node. |
| lodChildFeatures | Number[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 | Number[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 20: Attributes of the Class *Feature* within the NodeIndexDocument

#### 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.

**Warning: Deprectated elements.** The deprecated properties are highlighted. These deprecated elements will remain in the OGC I3S Community Standard until such time as Version 2 of this Community Standard is approved. At that time, these deprecated elements will be removed from the standard.

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| **metricType** | String | The name of the error metric, one of {maxScreenThreshold, screenSpaceRelative, ...} |
| **maxError** | number | Maximum metric value, expressed in the CRS of the vertex coordinates or in reference to other constants such as screen size. |
| 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 21: Attributes of the Class *LodSelection* within the NodeIndexDocument

**Example:** LOD Selection example

{

"metricType": "maxScreenThreshold",

"maxError": 34.87550189480981

}

#### obb

This class defines the properties for an oriented bounding box.

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| **center** | number[3] | The center point of the oriented bounding box. For a global scene, i.e. XY coordinate system in WGS1984, center is in longitude of decimal degrees, latitude of decimal degrees, elevation in meters. |
| **halfSize** | number[3] | Half size of the oriented bounding box in spatial reference units (or meters for global scenes). |
| **quaternion** | number[4] | Orientation of the oriented bounding box as a 4-component quaternion. For global scene, quaternion is in Earth-Centric-Earth-Fixed (ECEF) Cartesian space. ( Z+ : North, Y+ : East, X+: lon=lat=0.0). |

Note: properties in ***bold*** are required

**Example:** Global scene (WSG84) oriented-bounding box

{

"center": [

-105.01482,

39.747244,

1596.040551

],

"halfSize": [

29.421873,

29.539055,

22.082193

],

"quaternion": [

0.420972,

-0.055513,

-0.118217,

0.897622

]

}

### 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. Point Clouds do not have feature data. Features have the following structure:

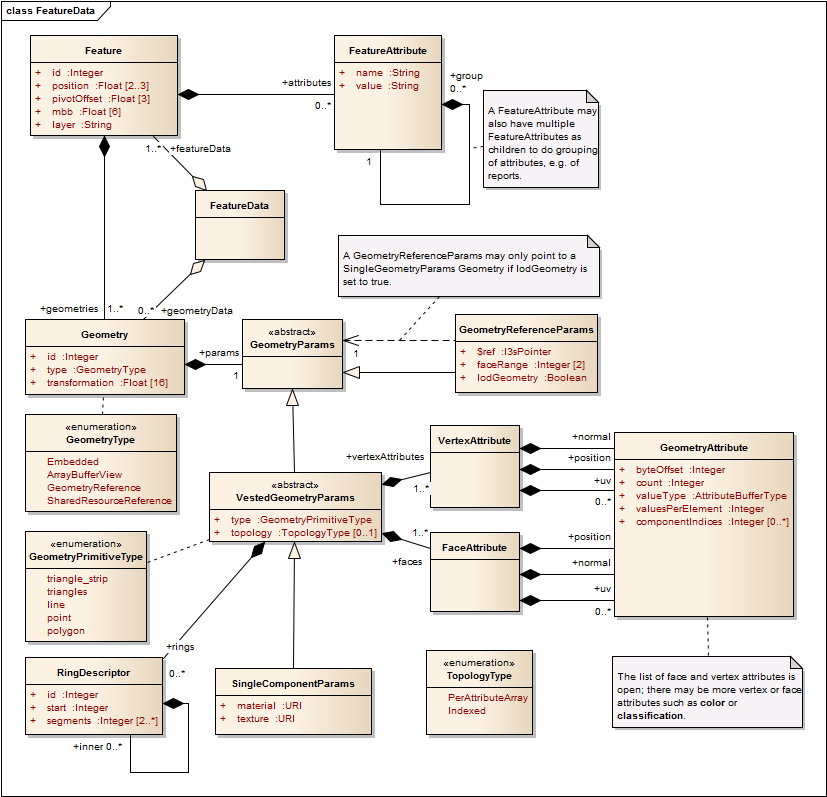


Figure 8: Logical schema of the FeatureData document

#### Class 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.

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| **id** | Integer | Feature ID, unique within the Node. If lodType is FeatureTree, the ID SHALL be unique in the store. |
| position | Number[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 | Number[3] | An array of three doubles, providing an optional, "semantic" pivot offset that can be used to, for example, correctly drape tree symbols. |
| mbb | Number[6] | An array of six doubles, corresponding to xmin, ymin, zmin, xmax, ymax and zmax 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***](#_Class_featureAttribute) | The list of attributes for this feature. |
| geometries | [***geometry***](#_Geometry.bin) | The list of geometries the feature has. A feature always SHALL have at least one Geometry. |

Table 22: Attributes of the Class *Feature* within the FeatureData document

#### 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 23: Attributes of the Class *FeatureAttribute* within the FeatureData document

#### Class geometry

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

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| **id** | Number | 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 | number[16] | 3D (4x4) transformation matrix expressed as a linear array of 16 values. |
| params | [***geometryParams***](#_Class_GeometryParams) See 7.6.6.4 below | The parameters for a geometry, as an Embedded GeometryParams object, an ArrayBufferView, a GeometryReference object, or a SharedResourceReference object. |

Note: properties in ***bold*** are required

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

Example: Geometry

{

"id": 0,

"type": "ArrayBufferView",

"transformation": [

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0

]

}

#### Class GeometryParams

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

This is one of:

* [geometryReferenceParams](#_Class_GeometryReferenceParams)
* [vestedGeometryParams](#_Class_VestedGeometryParams)
* [singleComponentParams](#_Class_SingleComponentParams)

These are described below.

#### 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** |
| **href** | String | In-document absolute reference to full geometry definition (Embedded or ArrayBufferView) using the [I3S json pointer](#_Pointers) syntax. |
| 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). |
| faceRange | Number[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 25: Attributes of the Class *GeometryReferenceParams* within the FeatureData document

**Example:** geomtryReferenceParams

{

"type": "GeometryReference",

"href": "/geometryData/1",

"faceRange": [

0,

195

]

}

#### 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 | String | Declares the typology of embedded geometry attributes or those in a geometry resources. When 'Indexed', the indices (faces) SHALL also be declared.  Possible values are:   * PerAttributeArray * InterleavedArray * Indexed |
| vertexAttributes | ***VertexAttributes*** | 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 | ***Class geometryAttribute*** | 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 26: Attributes of the Class *VestedGeometryParams* within the FeatureData document

**Example:** vestedGeometryParams

{

"type": "triangles",

"vertexAttributes": {

"position": {

"byteOffset": 8,

"valueType": "Float32",

"valuesPerElement": 3

},

"normal": {

"byteOffset": 2672,

"valueType": "Float32",

"valuesPerElement": 3

},

"uv0": {

"byteOffset": 5336,

"valueType": "Float32",

"valuesPerElement": 2

},

"color": {

"byteOffset": 7112,

"valueType": "UInt8",

"valuesPerElement": 4

}

}

}

#### Class SingleComponentParams

Objects of this type extend VestedGeometryParams and use one texture and one material. They can be used with aggregated LoD geometries. Component objects provide information on parts of the geometry they belong to, specifically with which material and texture to render them.

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| material | string | URL - 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 | string | URL - I3S Pointer reference to the material definition in this node's shared resource, from its root element. If present, used for the entire geometry. |
| **id** | number | The ID of the component, only unique within the Geometry. |
| materialID | number | UUID of the material, as defined in the shared resources bundle, to use for rendering this component. |
| textureID | number[] | Optional ID of the texture, as defined in shared resources, to use with the material to render this component. |
| regionID | number[] | Optional ID of a texture atlas region which to use with the texture to render this component. |

Note: Properties in bold are mandatory

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

#### Class Component

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 28: Attributes of the Class *Component* within the FeatureData document

#### 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 | Number | 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** | Number | The number of values needed 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. |

Note: Properties in bold are mandatory.

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

## 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.

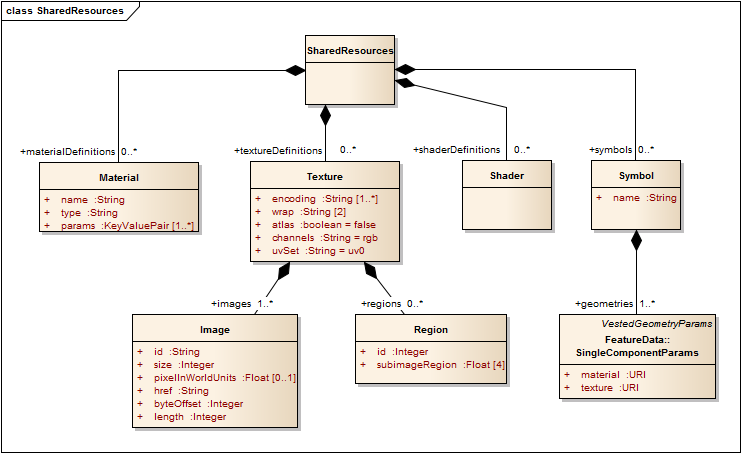


Figure 9: Logical schema of the SharedResources document

### Class SharedResource

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

#### 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 parameters 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 | Number | Indicates whether the transparency of this material; 0 = opaque, 1 = fully transparent. |
| params.reflectivity | Number | Indicates reflectivity of this Material. |
| params.shininess | Number | Indicates shininess of this Material. |
| params.ambient | Number [] | Ambient color of this Material. |
| params.diffuse | Number [] | Diffuse color of this Material. |
| params.specular | Number [] | 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 30: Attributes of the Class *Material* within the SharedResources document

#### Class Texture

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

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| **encoding** | string[] | MIMEType - The encoding/content type that is used by all images in this map |
| wrap | String[] | 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[] | 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, ... |
| images | [***Image***](#_Class_Image)[] | An image is a binary resource, containing a single raster that can be used to texture a feature or symbol. |

Table 31: Attributes of the Class *Texture* within the SharedResources document

Note: Mandatory properties in bold are mandatory.

#### 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](#_Textures.bin).

|  |  |  |
| --- | --- | --- |
| **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 32: Attributes of the Class Image within the SharedResources document

Note: Properties in bold are mandatory.

#### 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. ClassBreak Renderer defines the symbol of each feature in a Layer based on the value of a numeric attribute. Symbols are assigned based on classes or ranges of data. Each feature is assigned a symbol based on the class break in which the value of the attribute falls. UniqueValueRenderer allows you to symbolize features in a Layer based on one or more matching string attributes. This is typically done by using unique colors, fill styles, or images to represent features with equal values in a string field. |
| 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 33*: Attributes of the Class Renderer within the 3dSceneLayerInfo document*

#### 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 SHALL be {\\*MeshSymbol3D\\*}. |
| symbolLayers | Renderer::SymbolLayers | An object that represents how all features of this I3S layer will be drawn. |

Table 34*: Attributes of the Class Symbol within the 3dSceneLayerInfo document*

#### 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 35*: Attributes of the Class SymbolLayers within the 3dSceneLayerInfo document*

#### Class ShaderDefinition

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

#### Class Symbol

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

# I3S File Formats

## Textures

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 bundling strategy is determined by the authoring application so that specific aspects of the materials and textures used can be taken into account, such as tiling.

### Texture Recommendations and Requirements

The number and volume of textures tends to be the limiting display factor, especially for web and mobile clients. Thus, this standard provides several recommendations and requirements on texture resources that are delivered as part of an Indexed 3D Scene.

### Image Formats

I3S supports multiple texture formats. The choice of format depends on the use case. 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).

### Texture Sets

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

### Atlas usage and Regions

Individual textures should be aggregated into texture atlases. Each individual texture becomes a subtexture. As with all texture resources, the atlas has to be 2n-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 2n-sized, with n being in the range [3,12]. Otherwise if their width or height dimension is not 2n, 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 2n 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.

Subtexture pixels 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.

### Texture coordinates

Client capabilities for handling complex UV cases vary widely, so texture coordinates are used. 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.

### 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);

#### 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 |

## Geometry

The binary geometry attribute file follows the [Khronos[[29]](#footnote-29) Typed Array specification](http://www.khronos.org/registry/typedarray/specs/latest/)[[30]](#footnote-30) in the ECMAScript® 2015 Language Specification. Citing the overview of that specification:

“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)."

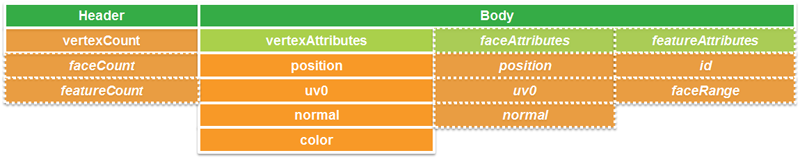
[](https://github.com/Esri/i3s-spec/blob/master/format/images/figure-10.png)

Figure 10: Geometry Buffer Layout with headers

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

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.

## Attribute Data

Attribute data is stored within I3S layers as part of the Scene Service cache along with geometry, texture, and material resources in an optimized, render 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 in the fields array and attributeStorageInfo array 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.*

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

Clients can use the key property (layers[id].attributeStorageInfo[].key) of the attributeStorageInfo to uniquely identify and request the attribute resource through a RESTful API, called **attributes**. Clients use the *attributeStorageInfo* metadata to decode the retrieved attribute binary content.

The attribute resource header contains:

* 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* [*code 1.*](https://github.com/Esri/i3s-spec/blob/master/profiles/meshpyramids/examples/3dSceneLayer.js) *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:

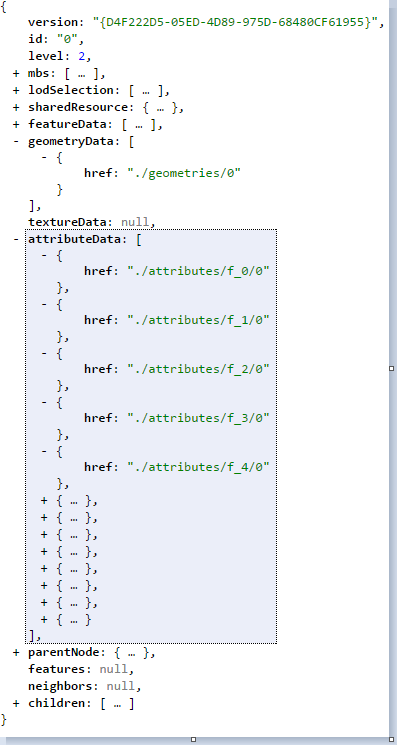
[](https://github.com/Esri/i3s-spec/blob/master/format/images/figure-12.png)

Figure 12: A node resource document

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

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

The attributes REST API allows client apps to fetch the attribute records of a field using its key property directly from a scene service layer. As a result, every scene node (with the exception of 'root' node), exposes 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: URL: **http://<sceneservrice-url>/attributes/<field\_key>/<id>**

* *attributes*: the RESTful resource responsible for fetching the binary attribute resource. A client application will be able to decode the content of this *attribute* resource based on the metadata information found in the scene layer *attributeStorageInfo* array (which adequately describes the content of the binary data).
* *field\_key*: the key value used to request the desired feature attribute content.
* *Id*: 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.

### A typical usage pattern of the attributes REST API

1. A client should get the attribute field from the metadata by fetching the scene server layers REST resource prior to symbolizing the node base on attribute information. 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* contains metadata information about the binary *attribute* resources. The attributeStorageInfo properties are defined in clause <insert reference>.

*Note: properties in* ***bold*** *are required*

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).

[](https://github.com/Esri/i3s-spec/blob/master/format/images/figure-13.png)

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 3D Scene Layer examples section.*

1. 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:

* 1. http://<myserver>/<my product>/rest/services/Hosted/SanFran/SceneServer/layers/0/nodes/0-0-0-0/**attributes/0/f\_1**
  2. 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*).

### Attribute Resources: Details

A numeric attribute resource is a single, one dimensional array. A string attribute resource is two, sequential, one dimensional arrays. 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](https://github.com/Esri/i3s-spec/blob/master/format/images/figure-14.png)). This layer has 6 fields named 'OID', 'Shape', 'NEAR\_FID', 'NEAR\_DIST', 'Name' and 'Building\_ID'.

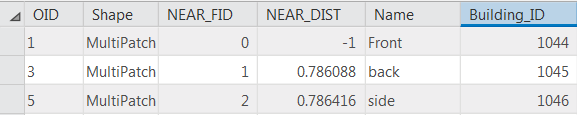
[](https://github.com/Esri/i3s-spec/blob/master/format/images/figure-14.png)

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](https://github.com/Esri/i3s-spec/blob/master/format/images/figure-11.png) and [Figure 13](https://github.com/Esri/i3s-spec/blob/master/format/images/figure-13.png), 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](https://github.com/Esri/i3s-spec/blob/master/format/images/figure-14.png)), which is of *FieldTypeGeometry* type, will not be included in the attribute cache of a scene layer.

Table 25 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 33: Attribute data types supported by a scene service layer.

NOTE: 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](https://github.com/Esri/i3s-spec/blob/master/format/images/figure-14.png) let's see how it maps to the different types of *attribute* resources.

* ***OID*** (type 'FieldTypeOID') is by default represented as an *UInt32-Array*, with a simple 1-d array of *UInt32* value type.
* **NEAR-FID** (type 'FieldTypeInteger') is represented as an *Int32-Array,* with a simple 1-d array of *Int32* value type.
* **NEAR\_DIST** (type 'FieldTypeDouble') is represented as a *Double-Array*, with a 1-d array of *Float64* value type.
* **Name** (type 'FieldTypeString') is represented as a *String-Array*. A String-Array supports storage of variable length strings and is stored as two arrays in sequence. The first fixed length array has the byte counts of each string (null terminated) in the second array. 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](https://github.com/Esri/i3s-spec/blob/master/format/images/figure-14.png) has 5 binary resources, as identified by keys *f\_0\_, f\_1\_, f\_2\_, f\_3\_* and *f\_4* and are accessible by their respective rest resource URLs (\_.../nodes/<nodeID>/attributes/0/f\_0, .../nodes/<nodeID>/attributes/0/f\_1, etc..).

## 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.

# Supported I3S Layer Structures

This section provides detailed information on the supported I3S layer structures for:

* Point Scene Layers
* Integrated Scene Mesh Layer
* 3D Object Scene Layer
* Point Cloud Scene Layer

Profiles of the mandatory classes that define the layer are provided in a corresponding annex.

## Point Scene Layer

Point scene layers contain point features and their attributes. Point scene layers are often used to visualize large amounts of 3D data like trees or buildings. Most phenomena that can be visualized by 3D symbols can be displayed with a point scene layers.

**

Example of a point scene layer

### Point Scene Layer Structure

Point scene layers contain point features and their attributes. Point scene layers are often used to visualize large amounts of 3D data like trees or buildings. Most phenomena that can be visualized by 3D symbols can be displayed with a point scene layers.

The point scene layer is structured into a tree of multiple JSON files. Besides storing information in the JSON format, some are also provided as binary buffer. Point scene layers can be used to create a scene layer package (\*.slpk) or an I3S service. A point scene layer *SHALL* contain profiles of the following classes:

* [Layer description](#_3dSceneLayerInfo)
* Nodes containing [Geometry](#_defaultGeometrySchema) and [Attributes](#_Attribute_Data)
* [Node Index Document](#_Class_3dNodeIndexDocument)
* [Statistics](#_statisticsInfo)

Please visit Annex G for details on the Point Scene Layer Profile as well as a full JSON example.

### Example of point scene layer structure

.<host>/SceneServer/layers

+--0 // scene layer document

+-- nodes

| +--root

| | +-- attributes

| | | +--f\_2

| | | +--f\_4

| | | +--(...)

| | +-- features

| | | +-- 0

| +-- (...)

+--statistics

| +-- f\_2

| | +--0

| +-- f\_4

| | +--0

| +-- (...)

+--resources

+-- styles

| +-- root

| +-- web

### HTTP API Overview

The following API methods are available for the I3S Point Scene Layer:

| **Resource** | **Type** | **Description** | **URL Template** |
| --- | --- | --- | --- |
| Scene Layer Document | JSON | This is the root document for the service that will contain properties common to the entire layer. | http://serviceURL/layers/{layerID} |

* layerID: Integer. ID of the associated layer.

Example: <http://my.server.com/PointSceneLayer/SceneServer/layers/0>

| **Resource** | **Type** | **Description** | **URL Template** |
| --- | --- | --- | --- |
| Node Document | JSON | Description of the node. | http://serviceURL/layers/{layerID}/nodes/{resourceID} |

* layerID: Integer. ID of the associated layer.
* resourceID: String. ID of the associated resource.

Example: <http://my.server.com/PointSceneLayer/SceneServer/layers/0/nodes/98>

| **Resource** | **Type** | **Description** | **URL Template** |
| --- | --- | --- | --- |
| Statistics | JSON | The statistics for the entire layer for a specific attribute. | http://serviceURL/layers/{layerID}/statistics/f\_{attributeID}/0 |

* layerID: Integer. ID of the associated layer.
* attributeID: Integer. ID of the specific attribute for the layer.

Example: <http://my.server.com/PointSceneLayer/SceneServer/layers/0/statistics/f_48/0>

| **Resource** | **Type** | **Description** | **URL Template** |
| --- | --- | --- | --- |
| Attributes | JSON | The attributes for the entire layer for a specific attribute. | http://serviceURL/layers/{layerID}/attributes/f\_{attributeID}/0 |

* layerID: Integer. ID of the associated layer.
* attributeID: Integer. ID of the specific attribute for the layer.

Example: <http://my.server.com/PointSceneLayer/SceneServer/layers/0/attributes/f_48/0>

| **Resource** | **Type** | **Description** | **URL Template** |
| --- | --- | --- | --- |
| Feature | JSON | Point location and feature IDs. | http://serviceURL/layers/{layerID}/nodes/{resourceID}/features/0 |

* layerID: Integer. ID of the associated layer.
* resourceID: String. ID of the associated node.

Example: <http://my.server.com/PointSceneLayer/SceneServer/layers/0/nodes/98/features/0>

## Integrated Mesh Scene Layer

Integrated mesh scene layers are generally created for citywide 3D mapping. Integrated mesh scene layers include an entire surface and cannot be restyled. Three-dimensional mesh data are typically captured by an automated process (e.g. drone) for constructing 3D objects out of large sets of overlapping imagery. The result integrates the original input image information as a textured mesh including 3D objects, such as buildings and trees, and elevation information.



*Example of integrated mesh scene layer*

### Integrated Mesh Scene Layer Structure

The Integrated Mesh scene layer is structured into a tree of multiple JSON files. Besides storing information in the JSON format, some are also provided as binary buffer. Integrated mesh scene layers can be used to create a scene layer package (\*.slpk) or an I3S service. An Integrated Mesh scene layer *SHALL* contain profiles of the following classes:

* [Layer description](#_3dSceneLayerInfo)
* Nodes containing [Geometry](https://github.com/ArcGIS/i3s-spec-internal/blob/master/docs/1.6/geometry.cmn.md), [Feature Data](https://github.com/ArcGIS/i3s-spec-internal/blob/master/docs/1.6/featureData.cmn.md), and [Texture](https://github.com/ArcGIS/i3s-spec-internal/blob/master/docs/1.6/texture.cmn.md)
* [Shared Resources](https://github.com/ArcGIS/i3s-spec-internal/blob/master/docs/1.6/sharedResource.cmn.md)

Please visit Annex H for more details on the Integrated Mesh Layer Profile as well as a JSON example.

### Example of integrated mesh scene layer structure

.<host>/SceneServer/layers

+--0 // scene layer document

+-- nodes

| +--0

| | +-- geometries

| | | +-- 0

| | | +-- 1

| | | +--(...)

| | +-- textures

| | | +-- 0

| | | +-- 1

| | | +--(...)

| | +-- shared

| +-- (...)

### Integrated Mesh Scene Layer HTTP API Overview

The following API methods are available for Integrated Mesh Scene Layer:

| **Resource** | **Type** | **Description** | **URL Template** |
| --- | --- | --- | --- |
| Scene Layer Document | JSON | This is the root document for the service that will contain properties common to the entire layer. | http://serviceURL/layers/{layerID} |

* layerID: Integer. ID of the associated layer.

Example: <http://my.server.com/IntegratedMeshSceneLayer/SceneServer/layers/0>

| **Resource** | **Type** | **Description** | **URL Template** |
| --- | --- | --- | --- |
| Node Document | JSON | Description of the node. | http://serviceURL/layers/{layerID}/nodes/{resourceID} |

* layerID: Integer. ID of the associated layer.
* resourceID: String. ID of the associated resource.

Example: <http://my.server.com/IntegratedMeshSceneLayer/SceneServer/layers/0/nodes/98>

| **Resource** | **Type** | **Description** | **URL Template** |
| --- | --- | --- | --- |
| Textures | JPG, PNG, DDS, KTX | The texture resource (image) | http://serviceURL/layers/{layerID}/nodes/{resourceID}/textures/{texture ID} |

* layerID: Integer. ID of the associated layer.
* resourceID: String. ID of the associated node.
* textureID: String. This ID returns one of the textures available for this node. The same texture may be available in different formats.

Example: <http://my.server.com/IntegratedMeshSceneLayer/SceneServer/layers/0/nodes/98/textures/1>

| **Resource** | **Type** | **Description** | **URL Template** |
| --- | --- | --- | --- |
| Geometries | bin | The geometry resource (mesh information) | http://serviceURL/layers/{layerID}/nodes/{resourceID}/geometries/0 |

* layerID: Integer. ID of the associated layer.
* resourceID: String. ID of the associated node.

Example: <http://my.server.com/IntegratedMeshSceneLayer/SceneServer/layers/0/nodes/98/geometries/0>

| **Resource** | **Type** | **Description** | **URL Template** |
| --- | --- | --- | --- |
| Shared Resources | JSON | Texture and material descriptions. | http://serviceURL/layers/{layerID}/nodes/{resourceID}/shared |

* layerID: Integer. ID of the associated layer.
* resourceID: String. ID of the associated node.

Example: <http://my.server.com/IntegratedMeshSceneLayer/SceneServer/layers/0/nodes/98/shared>

## 3D Object Scene Layer

A 3D object scene layer is used to visualize 3D objects. 3D object scene layers are often created from GIS data with attributes and explicitly modeled in 3D. These attributes allow definition queries to specify symbology and other properties in lieu of setting properties for each object individually. A 3D object scene layer can efficiently create and share just a few buildings or an entire city.

[](https://github.com/cnreediii/ogc-i3s-profiles/blob/master/standard_template/standard/images/LyonTextured.png)

Realistic 3D Object Scene Layer with textures

[](https://github.com/cnreediii/ogc-i3s-profiles/blob/master/standard_template/standard/images/LyonThematic.png)

Thematic 3D Object Scene Layer without textures

### 3D Object Scene Layer Structure

The 3D object scene layer is structured into a tree of multiple JSON files. Besides storing information in the JSON format, some are also provided as binary buffer. A 3D object scene layer can be used to create a scene layer package (\*.slpk) or an I3S service. A 3D object scene layer contains the following:

* Layer description (3DSceneLayer)
* Nodes containing Geometry (geometry) and Attributes (attributeStorageInfo)
* Node Index Document (3DNodeIndexDocument)
* Statistics (statsInfo)

### Example of 3D Object layer structure

.<host>/SceneServer/layers

+--0 // scene layer document

+-- nodes

| +--0

| | +-- attributes

| | | +--f\_2

| | | +--f\_4

| | | +--(...)

| | +-- geometries

| | | +-- 0

| | +-- textures

| | | +-- 0

| | | +-- 0\_0\_1

| | | +--(...)

| | +-- shared

| | (...)

+--statistics

| +-- f\_2

| | | +--0

| +-- f\_4

| | | +--0

| +-- (...)

### HTTP API Overview

The following API methods are available for the 3D Objects scene layer:

| **Resource** | **Type** | **Description** | **URL Template** |
| --- | --- | --- | --- |
| Scene Layer Document | JSON | This is the root document for the service that will contain properties common to the entire layer. | http://serviceURL/layers/{layerID} |

* layerID: Integer. ID of the associated layer. Esri products expect this to be 0.

Example: <http://my.server.com/3DObjectSceneLayer/SceneServer/layers/0>

| **Resource** | **Type** | **Description** | **URL Template** |
| --- | --- | --- | --- |
| Node Document | JSON | Description of the node. | http://serviceURL/layers/{layerID}/nodes/{resourceID} |

* layerID: Integer. ID of the associated layer. Esri products expect this to be 0.
* resourceID: String. ID of the associated resource.

Example: <http://my.server.com/3DObjectSceneLayer/SceneServer/layers/0/nodes/98>

| **Resource** | **Type** | **Description** | **URL Template** |
| --- | --- | --- | --- |
| Textures | JPG, PNG, DDS, KTX | The texture resource (image) | http://serviceURL/layers/{layerID}/nodes/{resourceID}/textures/{texture ID} |

* layerID: Integer. ID of the associated layer. Esri products expect this to be 0.
* resourceID: String. ID of the associated node.
* textureID: String. This ID returns one of the textures available for this node. The same texture may be available in different formats.

Example: <http://my.server.com/3DObjectSceneLayer/SceneServer/layers/0/nodes/98/textures/1>

| **Resource** | **Type** | **Description** | **URL Template** |
| --- | --- | --- | --- |
| Geometries | bin | The geometry resource. | http://serviceURL/layers/{layerID}/nodes/{resourceID}/geometries/0 |

* layerID: Integer. ID of the associated layer. Esri products expect this to be 0.
* resourceID: String. ID of the associated node.

Example: <http://my.server.com/3DObjectSceneLayer/SceneServer/layers/0/nodes/98/geometries/1>

| **Resource** | **Type** | **Description** | **URL Template** |
| --- | --- | --- | --- |
| Statistics | JSON | The statistics for the entire layer for a specific attribute. | http://serviceURL/layers/{layerID}/statistics/f\_{attributeID}/0 |

* layerID: Integer. ID of the associated layer. Esri clients expect this to be 0.
* attributeID: Integer. ID of the specific attribute for the layer.

Example: <http://my.server.com/3DObjectSceneLayer/SceneServer/layers/0/statistics/f_48/0>

| **Resource** | **Type** | **Description** | **URL Template** |
| --- | --- | --- | --- |
| Attributes | JSON | The attributes for the entire layer for a specific attribute. | http://serviceURL/layers/{layerID}/attributes/f\_{attributeID}/0 |

* layerID: Integer. ID of the associated layer. Esri products expect this to be 0.
* attributeID: Integer. ID of the specific attribute for the layer.

Example: <http://my.server.com/3DObjectSceneLayer/SceneServer/layers/0/attributes/f_48/0>

| **Resource** | **Type** | **Description** | | **URL Template** |
| --- | --- | --- | --- | --- |
| Shared Resources | JSON | Texture and material descriptions. | http://serviceURL/layers/{layerID}/nodes/{resourceID}/shared | |

* layerID: Integer. ID of the associated layer. Esri products expect this to be 0.
* resourceID: String. ID of the associated node.

Example: <http://my.server.com/3DObjectSceneLayer/SceneServer/layers/0/nodes/98/shared>

The following API methods are available for 3D Object scene layer:

| **Resource** | **Type** | **Description** | **URL Template** |
| --- | --- | --- | --- |
| Scene Layer Document | JSON | This is the root document for the service that will contain properties common to the entire layer. | http://serviceURL/layers/{layerID} |

* layerID: Integer. ID of the associated layer. Esri products expect this to be 0.

Example: <http://my.server.com/3DObjectSceneLayer/SceneServer/layers/0>

| **Resource** | **Type** | **Description** | **URL Template** |
| --- | --- | --- | --- |
| Node Document | JSON | Description of the node. | http://serviceURL/layers/{layerID}/nodes/{resourceID} |

* layerID: Integer. ID of the associated layer. Esri products expect this to be 0.
* resourceID: String. ID of the associated resource.

Example: <http://my.server.com/3DObjectSceneLayer/SceneServer/layers/0/nodes/98>

| **Resource** | **Type** | **Description** | **URL Template** |
| --- | --- | --- | --- |
| Textures | JPG, PNG, DDS, KTX | The texture resource (image) | http://serviceURL/layers/{layerID}/nodes/{resourceID}/textures/{texture ID} |

* layerID: Integer. ID of the associated layer. Esri products expect this to be 0.
* resourceID: String. ID of the associated node.
* textureID: String. This ID returns one of the textures available for this node. The same texture may be available in different formats.

Example: <http://my.server.com/3DObjectSceneLayer/SceneServer/layers/0/nodes/98/textures/1>

| **Resource** | **Type** | **Description** | **URL Template** |
| --- | --- | --- | --- |
| Geometries | bin | The geometry resource. | http://serviceURL/layers/{layerID}/nodes/{resourceID}/geometries/0 |

* layerID: Integer. ID of the associated layer. Esri products expect this to be 0.
* resourceID: String. ID of the associated node.

Example: <http://my.server.com/3DObjectSceneLayer/SceneServer/layers/0/nodes/98/geometries/1>

| **Resource** | **Type** | **Description** | **URL Template** |
| --- | --- | --- | --- |
| Statistics | JSON | The statistics for the entire layer for a specific attribute. | http://serviceURL/layers/{layerID}/statistics/f\_{attributeID}/0 |

* layerID: Integer. ID of the associated layer. Esri clients expect this to be 0.
* attributeID: Integer. ID of the specific attribute for the layer.

Example: <http://my.server.com/3DObjectSceneLayer/SceneServer/layers/0/statistics/f_48/0>

| **Resource** | **Type** | **Description** | **URL Template** |
| --- | --- | --- | --- |
| Attributes | JSON | The attributes for the entire layer for a specific attribute. | http://serviceURL/layers/{layerID}/attributes/f\_{attributeID}/0 |

* layerID: Integer. ID of the associated layer. Esri products expect this to be 0.
* attributeID: Integer. ID of the specific attribute for the layer.

Example: <http://my.server.com/3DObjectSceneLayer/SceneServer/layers/0/attributes/f_48/0>

| **Resource** | **Type** | **Description** | **URL Template** |
| --- | --- | --- | --- |
| Shared Resources | JSON | Texture and material descriptions. | http://serviceURL/layers/{layerID}/nodes/{resourceID}/shared |

* layerID: Integer. ID of the associated layer. Esri products expect this to be 0.
* resourceID: String. ID of the associated node.

Example: <http://my.server.com/3DObjectSceneLayer/SceneServer/layers/0/nodes/98/shared>

## Point Cloud Scene Layer

Point cloud scene layers are designed to quickly display large volumes of symbolized and filtered point cloud data. They are optimized for the displaying and sharing a variety of sensor data, including LiDAR.

Point cloud scene layers are scalable, which allows for efficiency when working with large datasets. While rendering very large point cloud datasets can be slow and limited by hardware, point cloud scene layers are efficient because they are rendered at an optimized point resolution for the specified area.

Point cloud scene layers also support caching attributes like RGB, Intensity, Flags, Class Code, Returns, User Data, Point Source ID, GPS Time, Scan Angle and Near Infrared. This allows client applications to update the symbology as well as query point information.

[](https://github.com/cnreediii/ogc-i3s-profiles/blob/master/standard_template/standard/images/PointCloud.png)

Example of point cloud rendering

### Point Cloud Scene Layer Structure

The point cloud scene layer is structured into a tree of multiple JSON files. Besides storing information in the JSON format, some are also provided as binary buffer. Point cloud scene layers can be used to create a scene layer package (\*.slpk) or a I3S service. Since an \*.slpk file can contain millions of documents, an [SLPK hash table](slpk\_hash\_table.pcsl.md) improves performance when scanning the slpk. A point cloud scene layer contains the following:

* Layer description
* Nodes containing Geometry and Attributes
* Node pages
* Statistics

### Example of point cloud layer structure

.<host>/SceneServer/layers

+--0 // layer description (named 3dSceneLayer.json in SLPK)

+-- nodepages

| +-- 0

| +-- 1

| +-- 2

| +-- (...)

| +-- 4

+-- nodes

| +--0

| | +-- attributes

| | | +--2

| | | +--4

| | | +--8

| | | +--(...)

| | +-- geometries

| | | +-- 0

| +--1

| | (...) //same structure for all nodes

| +--...

| +-- 259

| | (...) //same structure for all nodes

+--statistics

| +-- 2

| +-- 4

| +-- 8

| +-- (...)

### HTTP API Overview

The following API methods are available for accessing a point cloud scene layer:

| **Resource** | **Type** | **Description** | **URL Template** |
| --- | --- | --- | --- |
| Scene Layer Document | JSON | This is the root document for the service that will contain properties common to the entire layer. | http://serviceURL/layers/{layerID} |

* layerID: Integer. ID of the associated layer. Esri products expect this to be 0.

Example: <http://my.server.com/PointCloudSceneLayer/SceneServer/layers/0>

| **Resource** | **Type** | **Description** | **URL Template** |
| --- | --- | --- | --- |
| Node Page | JSON | A page of nodes. | http://serviceURL/layers/{layerID}/nodepages/{nodePageID}/ |

* layerID: Integer. ID of the associated layer. Esri products expect this to be 0.
* nodePageID: Integer. ID of the associated node page.

Example: <http://my.server.com/PointCloudSceneLayer/SceneServer/layers/0/nodepages/8>

| **Resource** | **Type** | **Description** | **URL Template** |
| --- | --- | --- | --- |
| Geometries | lepcc | The point coordinate values within the node. | http://serviceURL/layers/{layerID}/nodes/{resourceID}/geometries/0 |

* layerID: Integer. ID of the associated layer. Esri products expect this to be 0.
* resourceID: Integer. ID of the associated node.

Example: <http://my.server.com/PointCloudSceneLayer/SceneServer/layers/0/nodes/98/geometries/0>

| **Resource** | **Type** | **Description** | **URL Template** |
| --- | --- | --- | --- |
| Attributes | bin | The value for a specific attribute within a node. | http://serviceURL/layers/{layerID}/attributes/{attributeID} |

* layerID: Integer. ID of the associated layer. Esri products expect this to be 0.
* attributeID: Integer. ID of the specific attribute for the layer.

Example: <http://my.server.com/PointCloudSceneLayer/SceneServer/layers/0/attributes/64>

| **Resource** | **Type** | **Description** | **URL Template** |
| --- | --- | --- | --- |
| Statistics | JSON | The statistics for the entire layer for a specific attribute. | http://serviceURL/layers/{layerID}/statistics/{attributeID} |

* layerID: Integer. ID of the associated layer. Esri products expect this to be 0.
* attributeID: Integer. ID of the specific attribute for the layer.

Example: <http://my.server.com/PointCloudSceneLayer/SceneServer/layers/0/statistics/64>

## SLPK hash table

Scanning an SLPK (ZIP store) containing millions of documents is usually inefficient and slow. To improve first load and file scanning performances, a hash table file may be added to the SLPK.

### To create the SLPK hash-table

1. The offset of each file is known. For example, the byte offset from the beginning of the slpk file to the first byte of its ZIP local file header. See ZIP specification for reference.
2. Convert all file paths to their canonical form:
   * lower case
   * forward-slash path separator ( / )
   * no heading forward-slash (e.g. /my/PATH.json becomes my/path.json ).
3. Compute MD5 128 bit-hash for each canonical path to create an array of key-value pairs [MD5-digest →Offset 64bit].
4. Sort the key-value pairs by ascending keys using the following comparison based on little-endian architecture:

//for performance the following C++ comparator is used: (\*\*little-endian\*\*)

typedef std::array< unsigned char, 16 > md5\_hash;

bool less\_than( const md5\_hash& hash\_a, md5\_hash& hash\_b )

{

const uint64\* a = reinterpret\_cast<const uint64\*>(&hash\_a[0]);

const uint64\* b = reinterpret\_cast<const uint64\*>(&hash\_b[0]);

return a[0] == b[0] ? a[1] < b[1] : a[0] < b[0];

}

1. Write this sorted array as the last file of the SLPK archive (last entry in the ZIP central directory). The file must be named @specialIndexFileHASH128@. Each array element is 24-bytes long:
   * 16 bytes for the MD5-digest and 8 bytes for the offset
   * In little-endian order
   * No padding
   * No header

### To read SLPK hash-table

1. Convert the input path to canonical form and compute its md5 hash (i.e. key).
2. Search for key in the hash table. This can be easily implemented as a dichotomic (or binary) search since the keys are sorted.
3. Retrieve the file from the ZIP archive using the offset associated with key.

# Additional Informative Information

## 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. Here are a few implementation options:

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: Supports clients that want to get more complete meta-information about node’s position within the BVH hierarchy 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: Supports paged access pattern

Minimal structure elements: bounding volume; first-child reference; child-count; LoD selection data; etc..

1. 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)

1. 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

## 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  
• Texture and attribute data be created as JSON for index and metadata, and binary for large geometries  
• A Scene Layer Package format for distribution, or direct use, of the scene layer as a single file (see SLPK section)  
• Direct access is enabled through optional paired services that expose query-able and updatable RESTful endpoints  
• Explicit control over bounding index shape and per-node switching behavior to provide for optimized display and query;  
• Bounding volume hierarchy (BVH) based on bounding spheres (MBS) as well as oriented bounding boxes (OBB);  
• Scene layers may be created in Cartesian 3D or in global 3D world coordinate reference systems

# 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](#AnnexF) and their URL scheme.

Alternatively, a scene layer can be delivered as a 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)](#_Scene_Layer_Packages) 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.

## Scene Layer Packages

Scene Layer Packages (SLPK) allow a complete I3S layer, with all resources, to be transported or exchanged as a single file. A SLPK package can be consumed by applications directly.

The format of the package itself is defined as follows:

* The Archive type is always Zip64[[31]](#footnote-31).
  + 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.
  + 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[[32]](#footnote-32) compression applied to them.
* For resource compression, only the GZIP scheme is supported, as DEFLATE is not universally supported in all browsers.

The layout shown 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. The top level includes a nodes folder with

* A subfolder that contains all node resources
* A metadata.json file that describes the content of the SLPK
* A 3dSceneLayer.json.gz file that defines the Scene Layer

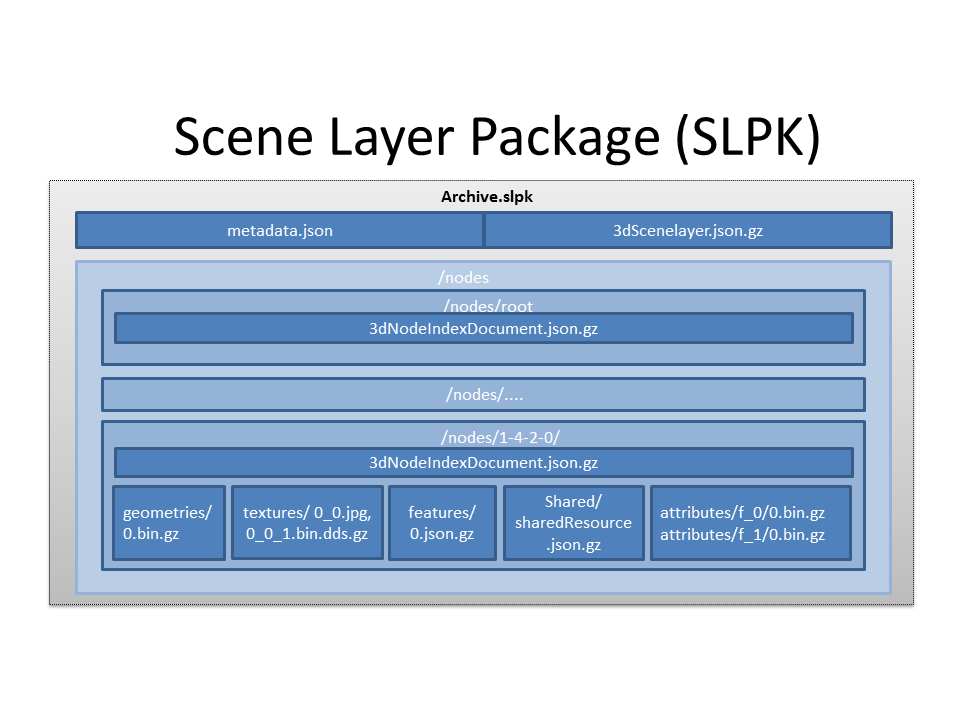


Figure 15: Example of a SLPK with BASIC folder layout

The 3dNodeIndexDocument.json.gz, features/0.json.gz and SharedResource.json.gz correspond to 3dNodeIndexDocument, featureData and SharedResource documents of the Scene Layer, and are JSON with GZIP compression.

An SLPK with basic folder layout has at the top level 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 above, the nodes subfolder contains, nodes named root, 1-4-2-0, and other nodes not pictured. All file resources within a particular node (e.g. 1-4-2-0), can be individually compressed with GZIP (indicated by the file extension .gz). Note, the texture resource is not compressed because it is an image (JPEG textures/0\_0.jpg).

Resources in subfolders, like geometries and attributes, are serialized as binary, and correspond to the geometryData and attributeData (e.g. geometries/0.bin.gz and attributes/f\_0/bin.gz)

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

### Metadata

Every SLPK archive has a metadata.json file. The following entries are required and must be of the specified type. The default is in **bold**.:

|  |  |  |
| --- | --- | --- |
| **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. |

## 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. The default is in **bold**.

|  |  |  |
| --- | --- | --- |
| **I3S Resources** | **Required** | **Notes** |
| /SceneServer | Yes | The [SceneServiceInfo](https://github.com/Esri/i3s-spec/blob/master/format/Indexed%203d%20Scene%20Layer%20Format%20Specification.md#_7_5) 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 | Yes | 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 | Yes | 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 | Yes | 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 | Yes | 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 | No | 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 | Yes | 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 | No | 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 | No | 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 | No | 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 | No | same as the attributeData resource f\_0/0 above |
| .... | .... | .... |
| /SceneServer/layers/0/nodes/1-4-2-0 | Yes | same as node resource root and 0 |

Table 34: example showing the layout of a SceneService

*Table 34 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 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. |
| 4/25/2019 | 1.1 | Carl Reed | Numerous | Version 1.1 revisions. See revision notes for details. |
| 6/5/2019 | 1.1 | Carl Reed | Various | Incorporated changes based on discussion with Tam on a number of questions the editor had. All resolved. |

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:

**Common Services Information**

* URL Pattern: http://<hostname>/<server-name>/rest/services/
* Method: GET
* Example Service: http://3dcities.maps.arcgis.com/arcgis/rest/services/
* Returns: List of all services running on the server instance.

**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](http://tiles.arcgis.com/tiles/z2tnIkrLQ2BRzr6P/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](http://tiles.arcgis.com/tiles/z2tnIkrLQ2BRzr6P/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](http://tiles.arcgis.com/tiles/z2tnIkrLQ2BRzr6P/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](http://tiles.arcgis.com/tiles/z2tnIkrLQ2BRzr6P/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](http://tiles.arcgis.com/tiles/z2tnIkrLQ2BRzr6P/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](http://tiles.arcgis.com/tiles/z2tnIkrLQ2BRzr6P/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](http://tiles.arcgis.com/tiles/z2tnIkrLQ2BRzr6P/arcgis/rest/services/New_York_LoD2_3D_Buildings/SceneServer/layers/0/nodes/5-1-0-0-0/attributes/f_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 G: I3S Scene Layer Profile for Points

This Annex specifies profiles of required classes to define an I3S Point Scene Layer.

3DSceneLayerInfo [Point Profile]

The object 3DSceneLayerInfo describes the properties of a layer in a store. Every scene layer contains 3DSceneLayerInfo. For features based scene layers, such as 3D objects or point scene layers, may include the default symbology, as specified in the drawingInfo, which contains stylization information for a feature layer.

The following are the mandatory 3DSceneLayerInfo properties for the Point Scene Layer Profile. Go to [Class 3DSceneLayerInfo](#_3dSceneLayerInfo) for a complete description of this class and its properties.

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| **id** | integer | Unique numeric ID of the layer. |
| **layerType** | string | The user-visible layer type.  Shall be: Point |
| **version** | string | The ID of the last update session in which any resource belonging to this layer has been updated. |
| **capabilities** | string[] | Capabilities supported by this layer.  Possible values for each array string:   * View: View is supported. * Query: Query is supported. * Edit: Edit is defined. |
| **store** | [store](#_Class_Store) | The 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. |

The following is an example JSON encoding for the 3DSceneLayerInfo Point Scene Layer Profile.

{

"id": 0,

"version": "39054BC8-A656-489C-B574-A717BC399259",

"name": "Trees\_Portland\_AllTypes",

"serviceUpdateTimeStamp": {

"lastUpdate": 1543373967000

},

"href": "./layers/0",

"layerType": "Point",

"spatialReference": {

"wkid": 4326,

"latestWkid": 4326,

"vcsWkid": 105790,

"latestVcsWkid": 3855

},

"heightModelInfo": {

"heightModel": "gravity\_related\_height",

"vertCRS": "EGM2008\_Geoid",

"heightUnit": "meter"

},

"ZFactor": 0.30480060960121924,

"alias": "Trees\_Portland\_AllTypes",

"description": "Trees\_Portland\_AllTypes",

"copyrightText": "",

"capabilities": [

"View",

"Query"

],

"elevationInfo": {

"mode": "relativeToGround",

"featureExpression": {

"value": 0

},

"unit": "us-foot"

},

"drawingInfo": {

"renderer": {

"type": "uniqueValue",

"styleName": "RealisticTreesStyle",

"field1": "type",

"visualVariables": [

{

"type": "sizeInfo",

"field": "height",

"axis": "height",

"valueUnit": "feet"

},

{

"type": "sizeInfo",

"field": "diameter",

"axis": "widthAndDepth",

"valueUnit": "feet"

}

]

}

},

"popupInfo": {

"title": "{name}",

"mediaInfos": [],

"fieldInfos": [

{

"fieldName": "OBJECTID",

"visible": true,

"isEditable": false,

"label": "OBJECTID"

},

{

"fieldName": "name",

"visible": true,

"isEditable": true,

"label": "Name"

},

{

"fieldName": "TreeFID",

"visible": true,

"isEditable": true,

"label": "Tree Feature ID"

},

{

"fieldName": "description",

"visible": true,

"isEditable": true,

"label": "Description"

},

{

"fieldName": "attribution",

"visible": true,

"isEditable": true,

"label": "Attribution/Source"

}

],

"popupElements": [

{

"fieldInfos": [

{

"fieldName": "OBJECTID",

"visible": true,

"isEditable": false,

"label": "OBJECTID"

},

{

"fieldName": "name",

"visible": true,

"isEditable": true,

"label": "Name"

},

{

"fieldName": "TreeFID",

"visible": true,

"isEditable": true,

"label": "Tree Feature ID"

},

{

"fieldName": "description",

"visible": true,

"isEditable": true,

"label": "Description"

}

],

"type": "fields"

}

],

"expressionInfos": []

},

"disablePopup": false,

"store": {

"id": "9FA4A13D-2FA3-4F35-B662-D0280C291EB8",

"profile": "points",

"resourcePattern": [

"3dNodeIndexDocument",

"Attributes",

"featureData"

],

"rootNode": "./nodes/root",

"version": "1.6",

"extent": [

-122.679052770042688,

45.520252738397879,

-122.673035202944419,

45.5241044684515472

],

"indexCRS": "http://www.opengis.net/def/crs/EPSG/0/4326",

"vertexCRS": "http://www.opengis.net/def/crs/EPSG/0/4326",

"nidEncoding": "application/vnd.esri.i3s.json+gzip; version=1.6",

"featureEncoding": "application/vnd.esri.i3s.json+gzip; version=1.6",

"attributeEncoding": "application/octet-stream; version=1.6",

"lodType": "AutoThinning",

"lodModel": "node-switching"

},

"fields": [

{

"name": "OBJECTID",

"type": "FieldTypeOID",

"alias": "OBJECTID"

},

{

"name": "name",

"type": "FieldTypeString",

"alias": "Name"

},

{

"name": "TreeFID",

"type": "FieldTypeString",

"alias": "Tree Feature ID"

},

{

"name": "description",

"type": "FieldTypeString",

"alias": "Description"

},

{

"name": "attribution",

"type": "FieldTypeString",

"alias": "Attribution/Source"

}

],

"attributeStorageInfo": [

{

"key": "f\_0",

"name": "OBJECTID",

"header": [

{

"property": "count",

"valueType": "UInt32"

}

],

"ordering": [

"ObjectIds"

],

"objectIds": {

"valueType": "UInt32",

"valuesPerElement": 1

}

},

{

"key": "f\_1",

"name": "name",

"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": "TreeFID",

"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\_3",

"name": "description",

"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\_4",

"name": "attribution",

"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

}

}

],

"statisticsInfo": [

{

"key": "f\_1",

"name": "name",

"href": "./statistics/f\_1"

},

{

"key": "f\_2",

"name": "TreeFID",

"href": "./statistics/f\_2"

},

{

"key": "f\_3",

"name": "description",

"href": "./statistics/f\_3"

},

{

"key": "f\_4",

"name": "attribution",

"href": "./statistics/f\_4"

}

]

}

3DNodeIndexDocument

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

All mandatory properties of the 3DNodeIndexDocument shall be implemented in the I3S Point Scene Layer profile.

The following is an example JSON encoding for the 3DNodeIndexDocument class for the Point Scene Layer Profile.

{

"id": "root",

"level": 1,

"version": "ee4fbf04-e882-444e-854d-cd519b68594a",

"mbs": [

-120.235609902853241,

39.1981414865211661,

1895.23079465422779,

446.269165373884221

],

"created": "2014-07-23T00:00:00.000Z",

"expires": "2015-07-23T00:00:00.000Z",

"transform": [

1,

0,

0,

0,

0,

1,

0,

0,

0,

0,

1,

0,

0,

0,

0,

1

],

"lodSelection": [

{

"metricType": "screenSpaceRelative",

"maxError": 0

},

{

"metricType": "distanceRangeFromDefaultCamera",

"maxError": 0

}

],

"featureData": [

{

"href": "./features/0",

"featureRange": [

0,

3

]

}

],

"parentNode": null,

"children": null,

"attributeData": [

{

"href": "./attributes/f\_0/0"

},

{

"href": "./attributes/f\_1/0"

},

{

"href": "./attributes/f\_2/0"

},

{

"href": "./attributes/f\_3/0"

}

]

}

Annex H: I3S Scene Layer 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.

[3dSceneLayer](https://github.com/ArcGIS/i3s-spec-internal/blob/master/profiles/meshpyramids/rules/docs/3dSceneLayerRules.html)

**3dNodeIndexDocument**

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

[3dNodeIndexDocument](https://github.com/ArcGIS/i3s-spec-internal/blob/master/profiles/meshpyramids/rules/docs/3dNodeIndexDocumentRules.html)

**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.

[FeatureData](https://github.com/ArcGIS/i3s-spec-internal/blob/master/profiles/meshpyramids/rules/docs/FeatureDataRules.html)

**SharedResources**

[SharedResources](https://github.com/ArcGIS/i3s-spec-internal/blob/master/profiles/meshpyramids/rules/docs/SharedResourceRules.html)

{

"id": 0,

"version": "3d3c7b51-6336-4893-b484-ad118775bcce",

"name": "Export2",

"href": "./layers/0",

"layerType": "IntegratedMesh",

"ZFactor": 1.0,

"spatialReference": {

"wkid": 4326,

"latestWkid": 4326

},

"alias": "Export2",

"description": "Vricon 3D Surface Model",

"copyrightText": "Limited in accordance with the accompanying Vricon EULA",

"capabilities": [

"View",

"Query"

],

"store": {

"id": "e9ecfade-0d85-4dd7-abb5-a3b0a07b9fd7",

"profile": "meshpyramids",

"resourcePattern": [

"3dNodeIndexDocument",

"SharedResource",

"Geometry",

"Attributes"

],

"rootNode": "./nodes/root",

"version": "1.4",

"extent": [

-106.5054122583675,

38.994677805489189,

-103.99630101552692,

39.996971340614706

],

"indexCRS": "http://www.opengis.net/def/crs/EPSG/0/4326",

"vertexCRS": "http://www.opengis.net/def/crs/EPSG/0/4326",

"nidEncoding": "application/vnd.esri.i3s.json+gzip; version=1.4",

"featureEncoding": "application/vnd.esri.i3s.json+gzip; version=1.4",

"geometryEncoding": "application/octet-stream; version=1.4",

"attributeEncoding": "application/octet-stream; version=1.4",

"textureEncoding": [

"image/jpeg",

"image/vnd-ms.dds"

],

"lodType": "MeshPyramid",

"lodModel": "node-switching",

"defaultGeometrySchema": {

"geometryType": "triangles",

"header": [

{

"property": "vertexCount",

"type": "UInt32"

},

{

"property": "featureCount",

"type": "UInt32"

}

],

"topology": "PerAttributeArray",

"ordering": [

"position",

"normal",

"uv0",

"color"

],

"vertexAttributes": {

"position": {

"valueType": "Float32",

"valuesPerElement": 3

},

"normal": {

"valueType": "Float32",

"valuesPerElement": 3

},

"uv0": {

"valueType": "Float32",

"valuesPerElement": 2

},

"color": {

"valueType": "UInt8",

"valuesPerElement": 4

}

},

"featureAttributeOrder": [

"id",

"faceRange"

],

"featureAttributes": {

"id": {

"valueType": "UInt64",

"valuesPerElement": 1

},

"faceRange": {

"valueType": "UInt32",

"valuesPerElement": 2

}

}

}

}

}

Annex I I3S Point Cloud Scene Layer Profile

The following are profiles of the common classes 3dSceneLayerInfo, attributeInfo, Store, Index, defaultGeometrySchema, vertexAttributes, and AttributeStatistics. Please note that most of these profiles of common and shared classes in the core I3S standard have additional mandatory elements

I3S point cloud Scene Layer (profile of 3dSceneLayerInfo)

This section describes the point cloud scene layer. The following table describes the properties of the point cloud scene layer. The description provides guidance on the mandatory/allowed values for a property (or object) for a given property.

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| **id** | integer | A unique identifying number for the layer. For point cloud scene layer, only a single layer is supported, therefore, id is always 0. |
| **layerType** | string | String indicating the layer type  ***Shall*** be: PointCloud |
| **name** | string | Represents the layer name. |
| alias | string | Represents the alias layer name. |
| desc | string | Description for the layer. |
| copyrightText | string | Copyright information to be displayed with this layer. |
| capabilities | string[] | Capabilities supported by this layer.  Possible values for each array string:   * View: View is supported. * Query: Query is supported. |
| **spatialReference** | ***spatialReference*** | An object containing the WKID or WKT identifying the spatial reference of the layer's geometry. No special point cloud guidance. |
| heightModelInfo | ***heightModelInfo*** | An object containing the vertical coordinate system information. No special point cloud guidance. |
| serviceUpdateTimeStamp | ***serviceUpdateTimeStamp*** | Object to provide time stamp when the I3S service or the source of the service was created or updated. No special point cloud guidance, |
| **store** | [***store***](#_Class_Store) | The storage for the layer. Please reference the point cloud scene layer *store* profile below. |
| **attributeStorageInfo** | [***attributeInfo***](#_attributeStorageInfo) [] | List of attributes included for this layer. Please reference the point cloud scene layer *attributeInfo* profile below. |
| drawingInfo | [***drawingInfo***](#_DrawingInfo) | An object containing drawing information for the point cloud scene layer. |
| elevationInfo | ***elevationInfo*** | An object containing elevation information. For point cloud scene layers, the following rules shall apply   * *mode* is mandatory and *SHALL* be absoluteHeight * *offset* is offset for the point cloud scene layer. The elevation unit is the coordinate systems units. |
| fields | ***Field***[] | For the point cloud scene layer profile, the properties *name* and *type* are mandatory. |

Below is an example JSON encoding of the point cloud scene layer definition. Please note that the “types” *pointCloudSplatAlgorithm* and *pointCloudStretchRenderer* are examples specific to what your clients support. Replace these types with whatever algorithms are appropriate on your development and 3d visualization environment. In the following example, PointCloudStretchRenderer defines the color of each point in a PointCloudLayer based on the value of a numeric attribute. *pointCloudSplatAlgorithm* is an algorithm for rendering points using sizes depending on point density. These are examples only.

{

"id": 0,

"layerType": "PointCloud",

"name": "Test Data",

"desc": "Nice Test data",

"capabilities": [

"View"

],

"spatialReference": {

"wkid": 4326,

"latestWkid": 4326,

"vcsWkid": 5703,

"latestVcsWkid": 5703

},

"store": {

"id": "",

"profile": "PointCloud",

"version": "2.0",

"extent": [

-122.45945212669568,

38.298060753040346,

-122.43014691292728,

38.303939889306761

],

"index": {

"nodeVersion": 1,

"boundingVolumeType": "obb",

"nodesPerPage": 64,

"lodSelectionMetricType": "density-threshold"

},

"defaultGeometrySchema": {

"geometryType": "points",

"header": [],

"topology": "PerAttributeArray",

"encoding": "lepcc-xyz",

"vertexAttributes": {

"position": {

"valueType": "Float64",

"valuesPerElement": 3

}

},

"ordering": [

"position"

]

}

},

"attributeStorageInfo": [

{

"key": "1",

"name": "ELEVATION",

"encoding": "embedded-elevation"

},

{

"key": "2",

"name": "INTENSITY",

"ordering": [

"attributeValues"

],

"attributeValues": {

"valueType": "UInt16",

"valuesPerElement": 1

},

"encoding": "lepcc-intensity"

},

{

"key": "4",

"name": "RGB",

"ordering": [

"attributeValues"

],

"attributeValues": {

"valueType": "UInt8",

"valuesPerElement": 3

},

"encoding": "lepcc-rgb"

},

{

"key": "8",

"name": "CLASS\_CODE",

"ordering": [

"attributeValues"

],

"attributeValues": {

"valueType": "UInt8",

"valuesPerElement": 1

}

},

{

"key": "16",

"name": "FLAGS",

"ordering": [

"attributeValues"

],

"attributeValues": {

"valueType": "UInt8",

"valuesPerElement": 1

}

},

{

"key": "32",

"name": "RETURNS",

"ordering": [

"attributeValues"

],

"attributeValues": {

"valueType": "UInt8",

"valuesPerElement": 1

}

}

],

"drawingInfo": {

"renderer": {

"pointSizeAlgorithm": {

"type": "pointCloudSplatAlgorithm",

"scaleFactor": 1,

"minSize": 4

},

"pointsPerInch": 25,

"field": "ELEVATION",

"fieldTransformType": "none",

"colorModulation": {

"field": "",

"minValue": 1,

"maxValue": 255

},

"type": "pointCloudStretchRenderer",

"stops": [

{

"value": 23.91416560580215,

"color": [

88,

19,

252,

255

]

},

{

"value": 59.9739474458430379,

"color": [

8,

252,

253,

255

]

},

{

"value": 96.033729285883922,

"color": [

242,

254,

42,

255

]

},

{

"value": 132.093511125924806,

"color": [

255,

43,

24,

255

]

}

]

}

},

"elevationInfo": {

"mode": "absoluteHeight"

},

"heightModelInfo": {

"heightModel": "gravity\_related\_height",

"vertCRS": "NAVD\_1988",

"heightUnit": "meter"

}

}

I3S point cloud scene layer: attributeInfo

List of attributes included for this point cloud scene layer.

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| **key** | string | Represents the attribute key. Key is the same as `id' used in the resource URL to fetch the binary buffers. |
| **name** | string | The attribute name. Must be unique for this layer. |
| ordering | string[] | Mapping between attribute to point. Only 1-to-1 is currently supported.  Possible values for each array string:   * attributeValues |
| encoding | string | Encoding (i.e. compression) for the attribute binary buffer if different from GZIP or no-compression.  Possible values are:   * embedded-elevation: No binary buffer but stats for this pseudo attribute will be available. For example, point.z from the geometry should be used. * lepcc-intensity: LEPCC compression for scaled integral type. * lepcc-rgb: LEPCC color compression for 3-channel RGB 8 bit. |
| attributeValues | [***value***](#_value) | Represents the description for value encoding, for example scalar or vector encoding. |

**Elevation pseudo-attribute**

{

"key": "1",

"name": "ELEVATION",

"encoding": "embedded-elevation"

}

**Example: Color attribute**

{

"key": "4",

"name": "RGB",

"ordering": [

"attributeValues"

],

"attributeValues": {

"valueType": "UInt8",

"valuesPerElement": 3

},

"encoding": "lepcc-rgb"

}

**Example: 8-bit uncompressed/GZIP compressed class-codes**

{

"key": "8",

"name": "CLASS\_CODE",

"ordering": [

"attributeValues"

],

"attributeValues": {

"valueType": "UInt8",

"valuesPerElement": 1

}

}

I3S point cloud scene layer profile: Store

The following table provides information on the properties that describes storage for the PointCloud scene layer. Please note that there are Point Cloud specific profiles of the Index and defaultGeometryScheme classes.

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| id | string | Id for the store. Not currently used by the point cloud scene layer. |
| **profile** | string | Defines the profile type of the scene layer as point cloud scene layer.  Must be:   * PointCloud |
| **version** | string | Point cloud scene layer store version. |
| **extent** | number[4] | 2D extent of the point cloud scene layer in the layers spatial reference units. |
| **index** | ***index*** | Describes the index (i.e. bounding volume tree) of the layer. |
| **defaultGeometrySchema** | [***defaultGeometrySchema***](#_defaultGeometrySchema) | Attribute description as field. |
| geometryEncoding | string | MIME type for the encoding used for the Geometry Resources. For example: application/octet-stream; version=1.6. |
| attributeEncoding | string | MIME type for the encoding used for the Attribute Resources. For example: application/octet-stream; version=1.6. |

*Note: properties in* ***bold*** *are required*

The following is a JSON example.

{

"id": "",

"profile": "PointCloud",

"version": "2.0",

"extent": [

-105.023403,

39.740089,

-105.011746,

39.757051

],

"index": {

"nodeVersion": 1,

"boundingVolumeType": "obb",

"nodesPerPage": 64,

"lodSelectionMetricType": "density-threshold"

},

"defaultGeometrySchema": {

"geometryType": "points",

"header": [],

"topology": "PerAttributeArray",

"encoding": "lepcc-xyz",

"vertexAttributes": {

"position": {

"valueType": "Float64",

"valuesPerElement": 3

}

},

"ordering": [

"position"

]

}

}

I3S point cloud scene layer: Index

The following table describes the properties of the index, such as bounding volume tree, of the Point Cloud layer.

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| **nodeVersion** | integer | The version of the individual nodes format. |
| **nodesPerPage** | integer | The page size describes the number of nodes per paged index document. 64 is currently expected. |
| boundingVolumeType | string | The bounding volume type. Only OBB is currently supported.  Must be:   * obb: Oriented bounding box |
| lodSelectionMetricType | string | Defines how node.lodThreshold should be interpreted  Must be:   * density-threshold: nodes[i].lodThreshold will represent an 'effective' 2D area for the node. This estimation works best when the point cloud scene layer represents a surface and is not volumetric. World space density is defined as Dw = node.pointCount / node.effectiveArea. Ds is Dw converted to screen space. Client would switch LOD when Ds is less/greater than a threshold defined by the client. For example, 0.1 point per pixel square. Note for point cloud scene layer creation: If each point footprint is assumed to be identical (say 0.1x0.1 unit), then the lodThreshold may be computed as number\_of\_points \* point\_footprint for a leaf node and sum( children[i].effective\_area) for inner nodes. |
| href | string |  |

*Note: properties in* ***bold*** *are mandatory.*

{

"nodeVersion": 1,

"boundingVolumeType": "obb",

"nodesPerPage": 64,

"lodSelectionMetricType": "density-threshold"

}

I3S point cloud scene layer: defaultGeometrySchema

The following table describes the properties for class defaultGeometryScheme of the Point Cloud layer.

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| **geometryType** | string | The type of primitive. Only points are supported for point cloud scene layer.  Shall be:   * points |
| header | [] | The header in binary buffers. Currently not supported for point cloud scene layer. |
| **topology** | string | This property is currently \**ignored* for point cloud scene layer since it only contains geometry position without vertex attributes.  Shall be:   * PerAttributeArray |
| **encoding** | string | Only 'lepcc-xyz' compression is currently supported.  Shall be:   * lepcc-xyz |
| ordering | string[] | Currently the geometry contains XYZ only, so vertex attribute must only list 'position'.  Possible values for each array string:   * position: vertex coordinates |
| **vertexAttributes** | [***vertexAttributes***](#_VertexAttributes) | The vertex buffer description. |

*Note: properties in* ***bold*** *are mandatory.*

JSON example for defaultGeometrySchema for point cloud scene layers.

{

"geometryType": "points",

"header": [],

"topology": "PerAttributeArray",

"encoding": "lepcc-xyz",

"vertexAttributes": {

"position": {

"valueType": "Float64",

"valuesPerElement": 3

}

},

"ordering": [

"position"

]

}

I3S point cloud scene layer: vertexAttributes

The following property defines the vertex buffer description.

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| position | [***value***](#_value) | Only LEPCC compressed (X,Y,Z) is supported. Decompressed data will be absolute Float64 position. |

Example: vertexAttributes

{

"position": {

"valueType": "Float64",

"valuesPerElement": 3

}

}

I3S point cloud scene layer: Attribute Statistics

This point scene layer class describes and contains statistics about each attribute. Statistics are useful to estimate attribute distribution and range. By convention, statistics are stored by attribute at layers/0/statistics/{attribute\_id}

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| attribute | string | Attribute name. Must match the name specified for this attribute in layer.attributeStorageInfo |
| stats | ***stats*** | Statistics for this attribute |
| labels | ***labels*** | The statistics document may contain labeling information for the attribute values. |

**Example: Elevation statistic (point.z statistics)**

{

"attribute" : "ELEVATION",

"stats" : {

"min" : 1567.597046,

"max" : 1649.043945,

"avg" : 1593.811809,

"stddev" : 12.722517,

"count" : 3799022.000000,

"sum" : 6054926127.557739,

"variance" : 161.862445,

"histogram" : {

"minimum" : 1567.596482,

"maximum" : 1644.937967,

"counts" : [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 3, 123, 1852, 7407, 11776, 15386, 11689, 12569, 10041, 11340, 12370, 18329, 29686, 40210, 44547, 50266, 86603, 102660, 129177, 113065, 97772, 103083, 92726, 74721, 70910, 68750, 65077, 66181, 75049, 69223, 65015, 65122, 54877, 46869, 48223, 47339, 38808, 35533, 35212, 33455, 29682, 33789, 41732, 26137, 23063, 26649, 20234, 15673, 16440, 22573, 23646, 24871, 25511, 25566, 22712, 20494, 19606, 20215, 18483, 17837, 17991, 17078, 19259, 20789, 20905, 18258, 17028, 20344, 20705, 20444, 21581, 19226, 16906, 19515, 15510, 15514, 15741, 15821, 16097, 14391, 14062, 15048, 15715, 15695, 17170, 14275, 12690, 13680, 15699, 15755, 15074, 14364, 16359, 20516, 16407, 12986, 13478, 12766, 12209, 11665, 11931, 13373, 14710, 14430, 14360, 15652, 15713, 14377, 14540, 12478, 12802, 16602, 13237, 12228, 10972, 12456, 13791, 15004, 18845, 15398, 16732, 15076, 13460, 12835, 12615, 13230, 12967, 12310, 13408, 18330, 20433, 21039, 26813, 21120, 12081, 6965, 7830, 9959, 8015, 6127, 6679, 6386, 9682, 11268, 5126, 4643, 5009, 4573, 4029, 4647, 4351, 6812, 4024, 3578, 3810, 4463, 5298, 4970, 2439, 2075, 1624, 1734, 1579, 1596, 1417, 1450, 1391, 1313, 1321, 1970, 3932, 1044, 982, 1158, 871, 622, 739, 513, 526, 378, 388, 500, 1172, 353, 244, 248, 175, 139, 254, 259, 248, 284, 297, 173, 155, 169, 142, 246, 130, 136, 208, 194, 96, 98, 91, 93, 126, 151, 125, 125, 119, 126, 118, 178, 140, 96, 76, 85, 38, 85]

}

}

}

**Example: Intensity statistics**

{

"attribute" : "INTENSITY",

"stats" : {

"min" : 0.000000,

"max" : 256.000000,

"avg" : 0.000000,

"stddev" : 0.000000,

"count" : 3799022.000000,

"sum" : 0.000000,

"variance" : 0.000000,

"histogram" : {

"minimum" : 0.000000,

"maximum" : 256.000000,

"counts" : [143339, 143858, 149770, 164501, 174011, 181377, 192349, 166440, 132127, 110664, 100962, 97616, 97383, 100071, 106157, 114260, 119177, 122104, 121872, 116999, 111086, 102366, 92383, 81642, 71199, 61043, 53223, 47350, 42872, 39193, 38076, 403552]

}

}

}

**Example: RGB color statistics ( please note that histogram is not required here)**

{

"attribute" : "RGB",

"stats" : {

"min" : 0.000000,

"max" : 255.000000,

"avg" : 0.022315,

"stddev" : 0.000000,

"count" : 11397066.000000,

"sum" : 510730851.000000,

"variance" : 0.000000,

"histogram" : {

"minimum" : 0.000000,

"maximum" : 0.000000,

"counts" : []

}

}

}

**Example: Class code statistics with labels**

{

"attribute": "CLASS\_CODE",

"stats": {

"min": 1.0,

"max": 12.0,

"avg": 5.63104,

"stddev": 2.629335,

"count": 3799022.0,

"sum": 21392446.0,

"variance": 6.913403,

"histogram": {

"minimum": 1.0,

"maximum": 12.0,

"counts": [

14,

802764,

681975,

3056,

153,

387412,

4948,

1904257,

9987,

4073,

383

]

},

"mostFrequentValues": [

{

"value": 8.0,

"count": 1904257

},

{

"value": 2.0,

"count": 802764

},

{

"value": 3.0,

"count": 681975

},

{

"value": 6.0,

"count": 387412

},

{

"value": 9.0,

"count": 9987

},

{

"value": 7.0,

"count": 4948

},

{

"value": 10.0,

"count": 4073

},

{

"value": 4.0,

"count": 3056

},

{

"value": 11.0,

"count": 383

},

{

"value": 5.0,

"count": 153

},

{

"value": 1.0,

"count": 14

}

]

},

"labels": {

"labels": [

{

"value": 1.0,

"label": "Unclassified"

},

{

"value": 2.0,

"label": "Ground"

},

{

"value": 3.0,

"label": "Low Vegetation"

},

{

"value": 4.0,

"label": "Medium Vegetation"

},

{

"value": 5.0,

"label": "High Vegetation"

},

{

"value": 6.0,

"label": "Building"

},

{

"value": 7.0,

"label": "Low Point(noise)"

},

{

"value": 8.0,

"label": "Model Key"

},

{

"value": 9.0,

"label": "Water"

},

{

"value": 10.0,

"label": "Rail"

},

{

"value": 11.0,

"label": "Road Surface"

}

]

}

}

**Example: Flags statistics (LIDAR point cloud)**

{

"attribute": "FLAGS",

"stats": {

"min": 0.000000,

"max": 137.000000,

"avg": 0.000000,

"stddev": 0.000000,

"count": 82673752.000000,

"sum": 0.000000,

"variance": 0.000000,

"histogram": {

"minimum": 0.000000,

"maximum": 137.000000,

"counts": [

58138737,

0,

0,

0,

0,

0,

0,

0,

24490375,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

1610,

0,

0,

0,

0,

0,

0,

0,

43030

]

},

"mostFrequentValues": [

{

"value": 0.000000,

"count": 58138737

},

{

"value": 8.000000,

"count": 24490375

},

{

"value": 136.000000,

"count": 43030

},

{

"value": 128.000000,

"count": 1610

}

]

},

"labels": {

"bitfieldLabels": [

{

"bitNumber": 3,

"label": "Overlap"

},

{

"bitNumber": 7,

"label": "Edge"

}

]

}

}

**Example: Returns statistics (LIDAR point cloud)**

{

"attribute": "RETURNS",

"stats": {

"min": 17.0,

"max": 52.0,

"avg": 17.41348,

"stddev": 2.606183,

"count": 3799022.0,

"sum": 66154192.0,

"variance": 6.792191,

"histogram": {

"minimum": 17.0,

"maximum": 52.0,

"counts": [

3704673,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

49581,

43733,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

394,

341,

300

]

},

"mostFrequentValues": [

{

"value": 17.0,

"count": 3704673

},

{

"value": 33.0,

"count": 49581

},

{

"value": 34.0,

"count": 43733

},

{

"value": 49.0,

"count": 394

},

{

"value": 50.0,

"count": 341

},

{

"value": 51.0,

"count": 300

}

]

}

}

I3S point cloud scene layer: nodepage

Nodes represent the spatial index of the data as a bounding-volume hierarchy. To reduce the number of node-index requests required to traverse this index tree, they are organized in *pages* of [layer.index.nodesPerPage](https://github.com/cnreediii/i3s-spec/blob/master/docs/2.0/index.md) nodes.

Children must be **contiguous**, in index range, so they may be located using firstChild and childrenCount fields.

**Page number computation example:**

Assuming [layer.store.index.nodesPerPage](https://github.com/cnreediii/i3s-spec/blob/master/docs/2.0/index.md) = 64, then node id = 78 will be in page page\_id = floor( 78 / 64) = 1 (i.e. second page)pc

**Properties**

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| **nodes** | [node](https://github.com/cnreediii/i3s-spec/blob/master/docs/2.0/node.pcsl.md)[] | Array of nodes |

*Note: properties in* ***bold*** *are mandatory.*

**Examples**

**Example: Global scene (WSG84, last page)**

{

"nodes": [

{

"resourceId": 704,

"obb": {

"center": [

-105.01482,

39.747244,

1596.040551

],

"halfSize": [

29.421873,

29.539055,

22.082193

],

"quaternion": [

0.420972,

-0.055513,

-0.118217,

0.897622

]

},

"firstChild": 0,

"childCount": 0,

"vertexCount": 7872,

"lodThreshold": 8979.959961

},

{

"resourceId": 705,

"obb": {

"center": [

-105.014132,

39.747244,

1588.67982

],

"halfSize": [

29.421803,

29.538986,

14.721462

],

"quaternion": [

0.420972,

-0.055509,

-0.118215,

0.897623

]

},

"firstChild": 0,

"childCount": 0,

"vertexCount": 7055,

"lodThreshold": 8047.970215

},

{

"resourceId": 706,

"obb": {

"center": [

-105.01512,

39.747343,

1629.163972

],

"halfSize": [

3.677743,

3.692391,

3.680366

],

"quaternion": [

0.420971,

-0.055515,

-0.118217,

0.897623

]

},

"firstChild": 0,

"childCount": 0,

"vertexCount": 1,

"lodThreshold": 1.140747

},

{

"resourceId": 707,

"obb": {

"center": [

-105.013445,

39.746714,

1584.999455

],

"halfSize": [

29.421768,

29.538958,

11.0411

],

"quaternion": [

0.420977,

-0.055505,

-0.118212,

0.897621

]

},

"firstChild": 0,

"childCount": 0,

"vertexCount": 7567,

"lodThreshold": 8632.032227

},

{

"resourceId": 708,

"obb": {

"center": [

-105.012758,

39.746714,

1584.999455

],

"halfSize": [

29.421768,

29.538958,

11.041096

],

"quaternion": [

0.420978,

-0.055501,

-0.11821,

0.897621

]

},

"firstChild": 0,

"childCount": 0,

"vertexCount": 7168,

"lodThreshold": 8176.874512

},

{

"resourceId": 709,

"obb": {

"center": [

-105.013445,

39.747244,

1584.999456

],

"halfSize": [

29.42177,

29.538956,

11.041099

],

"quaternion": [

0.420973,

-0.055505,

-0.118213,

0.897623

]

},

"firstChild": 0,

"childCount": 0,

"vertexCount": 7532,

"lodThreshold": 8592.106445

},

{

"resourceId": 710,

"obb": {

"center": [

-105.012758,

39.747244,

1581.31909

],

"halfSize": [

29.421768,

29.538956,

14.721463

],

"quaternion": [

0.420973,

-0.0555,

-0.118211,

0.897623

]

},

"firstChild": 0,

"childCount": 0,

"vertexCount": 6710,

"lodThreshold": 7654.412109

},

{

"resourceId": 711,

"obb": {

"center": [

-105.01482,

39.747775,

1592.360184

],

"halfSize": [

29.421841,

29.539022,

18.401829

],

"quaternion": [

0.420968,

-0.055512,

-0.118217,

0.897624

]

},

"firstChild": 0,

"childCount": 0,

"vertexCount": 7044,

"lodThreshold": 8035.421875

},

{

"resourceId": 712,

"obb": {

"center": [

-105.014132,

39.747775,

1584.999455

],

"halfSize": [

29.421772,

29.538952,

11.041098

],

"quaternion": [

0.420968,

-0.055508,

-0.118215,

0.897625

]

},

"firstChild": 0,

"childCount": 0,

"vertexCount": 7080,

"lodThreshold": 8076.48877

},

{

"resourceId": 713,

"obb": {

"center": [

-105.01482,

39.748305,

1599.720916

],

"halfSize": [

29.421906,

29.539085,

25.762558

],

"quaternion": [

0.420964,

-0.055512,

-0.118217,

0.897626

]

},

"firstChild": 0,

"childCount": 0,

"vertexCount": 7355,

"lodThreshold": 8390.194336

},

{

"resourceId": 714,

"obb": {

"center": [

-105.014133,

39.748305,

1599.720915

],

"halfSize": [

29.421902,

29.539082,

25.762558

],

"quaternion": [

0.420964,

-0.055508,

-0.118215,

0.897626

]

},

"firstChild": 0,

"childCount": 0,

"vertexCount": 7358,

"lodThreshold": 8393.616211

},

{

"resourceId": 715,

"obb": {

"center": [

-105.014648,

39.748504,

1629.163965

],

"halfSize": [

7.355486,

7.38478,

3.680366

],

"quaternion": [

0.420962,

-0.055511,

-0.118217,

0.897627

]

},

"firstChild": 0,

"childCount": 0,

"vertexCount": 5,

"lodThreshold": 5.703735

},

{

"resourceId": 716,

"obb": {

"center": [

-105.014218,

39.748139,

1640.205065

],

"halfSize": [

7.355505,

3.692403,

14.72146

],

"quaternion": [

0.420965,

-0.055508,

-0.118215,

0.897626

]

},

"firstChild": 0,

"childCount": 0,

"vertexCount": 71,

"lodThreshold": 80.993034

},

{

"resourceId": 717,

"obb": {

"center": [

-105.013445,

39.747775,

1588.67982

],

"halfSize": [

29.421797,

29.538982,

14.721462

],

"quaternion": [

0.420969,

-0.055504,

-0.118213,

0.897625

]

},

"firstChild": 0,

"childCount": 0,

"vertexCount": 7252,

"lodThreshold": 8272.697266

},

{

"resourceId": 718,

"obb": {

"center": [

-105.012758,

39.747775,

1588.67982

],

"halfSize": [

29.421799,

29.538986,

14.721464

],

"quaternion": [

0.420969,

-0.0555,

-0.118211,

0.897625

]

},

"firstChild": 0,

"childCount": 0,

"vertexCount": 5507,

"lodThreshold": 6282.09375

},

{

"resourceId": 719,

"obb": {

"center": [

-105.013445,

39.748305,

1588.67982

],

"halfSize": [

29.421803,

29.538984,

14.721462

],

"quaternion": [

0.420965,

-0.055504,

-0.118213,

0.897627

]

},

"firstChild": 0,

"childCount": 0,

"vertexCount": 7872,

"lodThreshold": 8979.959961

},

{

"resourceId": 720,

"obb": {

"center": [

-105.012758,

39.748305,

1592.360184

],

"halfSize": [

29.421799,

29.538982,

11.041098

],

"quaternion": [

0.420965,

-0.055499,

-0.118211,

0.897627

]

},

"firstChild": 0,

"childCount": 0,

"vertexCount": 5036,

"lodThreshold": 5744.801758

}

]

}

I3S point cloud scene layer: node

A single bounding volume hierarchy node.

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| **resourceId** | integer | Index of the first child of this node. The resourceID must be used to query node resources, like geometry buffer (XYZ) /nodes//geometry/0 and attribute buffers. One buffer can have one attribute. Available attributes are declared in the SceneLayer document. /nodes//attributes/<attrib\_key>. |
| **firstChild** | integer | Index of the first child of this node. |
| **childCount** | integer | Number of children for this node. Value is 0 if node is a leaf node. |
| vertexCount | integer | Number of points for this node. |
| **obb** | [obb](https://github.com/cnreediii/i3s-spec/blob/master/docs/2.0/obb.cmn.md) | Oriented bounding boxes (OBB) are the only supported bounding volumes. |
| lodThreshold | number | This metric may be used as a threshold to split a parent node into its children. See [layer.store.index.lodSelectionMetricType](https://github.com/cnreediii/i3s-spec/blob/master/docs/2.0/index.pcsl.md) |

*Note: properties in* ***bold*** *are mandatory.*

**Example: Global scene (WSG84)**

{

"resourceId": 704,

"obb": {

"center": [

-105.01482,

39.747244,

1596.040551

],

"halfSize": [

29.421873,

29.539055,

22.082193

],

"quaternion": [

0.420972,

-0.055513,

-0.118217,

0.897622

]

},

"firstChild": 0,

"childCount": 0,

"vertexCount": 7872,

"lodThreshold": 8979.959961

}

I3S point cloud scene layer: stats

Contains statistics about each attribute. Statistics are useful to estimate attribute distribution and range.

**Properties**

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| **min** | number | (Conservative) minimum attribute value for the entire layer. |
| **max** | number | (Conservative) maximum attribute value for the entire layer. |
| **count** | number | Count for the entire layer. |
| sum | number | Sum of the attribute values over the entire layer. |
| avg | number | Representing average or mean value. For example, sum/count. |
| stddev | number | Representing the standard deviation. |
| variance | number | Representing variance. For example, stats.stddev \*stats.stddev. |
| histogram | histogram | Represents the histogram. |
| mostFrequentValues | valuecount[] | An array of most frequently used values within the point cloud scene layer. |

*Note: properties in* ***bold*** *are mandatory.*

I3S point cloud scene layer: histogram

The histogram of the point cloud scene layer. The bin size may be computed as (max-min)/bin count. Please note that stats.histo.min/max is not equivalent to stats.min/max since values smaller than stats.histo.min and greater than stats.histo.max are counted in the first and last bin respectively. The values stats.min and stats.max may be conservative estimates.

**Properties**

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| **minimum** | number | Minimum attribute value for the entire layer. |
| **maximum** | number | Maximum attribute value for the entire layer. Maximum array size for stats.histo.counts is 256. |
| **counts** | number[:256] | Count for the entire layer. |

*Note: properties in* ***bold*** *are required*

**Example: histogram**

{

"minimum": 1567.596482,

"maximum": 1644.937967,

"counts": [

1,

1,

0,

0,

0,

1,

3,

123,

1852

]

}

**Stats Examples**

**Example: Elevation statistic (point.z statistics)**

{

"min": 1567.597046,

"max": 1649.043945,

"avg": 1593.811809,

"stddev": 12.722517,

"count": 3799022.000000,

"sum": 6054926127.557739,

"variance": 161.862445,

"histogram": {

"minimum": 1567.596482,

"maximum": 1644.937967,

"counts": [

1,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

1,

0,

0,

0,

1,

3,

123,

1852,

7407,

11776,

15386,

11689,

12569,

10041,

11340,

12370,

18329,

29686,

40210,

44547,

50266,

86603,

102660,

129177,

113065,

97772,

103083,

92726,

74721,

70910,

68750,

65077,

66181,

75049,

69223,

65015,

65122,

54877,

46869,

48223,

47339,

38808,

35533,

35212,

33455,

29682,

33789,

41732,

26137,

23063,

26649,

20234,

15673,

16440,

22573,

23646,

24871,

25511,

25566,

22712,

20494,

19606,

20215,

18483,

17837,

17991,

17078,

19259,

20789,

20905,

18258,

17028,

20344,

20705,

20444,

21581,

19226,

16906,

19515,

15510,

15514,

15741,

15821,

16097,

14391,

14062,

15048,

15715,

15695,

17170,

14275,

12690,

13680,

15699,

15755,

15074,

14364,

16359,

20516,

16407,

12986,

13478,

12766,

12209,

11665,

11931,

13373,

14710,

14430,

14360,

15652,

15713,

14377,

14540,

12478,

12802,

16602,

13237,

12228,

10972,

12456,

13791,

15004,

18845,

15398,

16732,

15076,

13460,

12835,

12615,

13230,

12967,

12310,

13408,

18330,

20433,

21039,

26813,

21120,

12081,

6965,

7830,

9959,

8015,

6127,

6679,

6386,

9682,

11268,

5126,

4643,

5009,

4573,

4029,

4647,

4351,

6812,

4024,

3578,

3810,

4463,

5298,

4970,

2439,

2075,

1624,

1734,

1579,

1596,

1417,

1450,

1391,

1313,

1321,

1970,

3932,

1044,

982,

1158,

871,

622,

739,

513,

526,

378,

388,

500,

1172,

353,

244,

248,

175,

139,

254,

259,

248,

284,

297,

173,

155,

169,

142,

246,

130,

136,

208,

194,

96,

98,

91,

93,

126,

151,

125,

125,

119,

126,

118,

178,

140,

96,

76,

85,

38,

85

]

}

}

**Example: Intensity statistics**

{

"min": 0.000000,

"max": 256.000000,

"avg": 0.000000,

"stddev": 0.000000,

"count": 3799022.000000,

"sum": 0.000000,

"variance": 0.000000,

"histogram": {

"minimum": 0.000000,

"maximum": 256.000000,

"counts": [

143339,

143858,

149770,

164501,

174011,

181377,

192349,

166440,

132127,

110664,

100962,

97616,

97383,

100071,

106157,

114260,

119177,

122104,

121872,

116999,

111086,

102366,

92383,

81642,

71199,

61043,

53223,

47350,

42872,

39193,

38076,

403552

]

}

}

**Example: RGB color statistics ( please note that histogram is not required here)**

{

"min": 0.000000,

"max": 255.000000,

"avg": 0.022315,

"stddev": 0.000000,

"count": 11397066.000000,

"sum": 510730851.000000,

"variance": 0.000000,

"histogram": {

"minimum": 0.000000,

"maximum": 0.000000,

"counts": []

}

}

**Example: Class code statistics with labels**

{

"min": 1.0,

"max": 12.0,

"avg": 5.63104,

"stddev": 2.629335,

"count": 3799022.0,

"sum": 21392446.0,

"variance": 6.913403,

"histogram": {

"minimum": 1.0,

"maximum": 12.0,

"counts": [

14,

802764,

681975,

3056,

153,

387412,

4948,

1904257,

9987,

4073,

383

]

},

"mostFrequentValues": [

{

"value": 8.0,

"count": 1904257

},

{

"value": 2.0,

"count": 802764

},

{

"value": 3.0,

"count": 681975

},

{

"value": 6.0,

"count": 387412

},

{

"value": 9.0,

"count": 9987

},

{

"value": 7.0,

"count": 4948

},

{

"value": 10.0,

"count": 4073

},

{

"value": 4.0,

"count": 3056

},

{

"value": 11.0,

"count": 383

},

{

"value": 5.0,

"count": 153

},

{

"value": 1.0,

"count": 14

}

]

}

**Example: Flags statistics (LIDAR point cloud)**

{

"min": 0.000000,

"max": 137.000000,

"avg": 0.000000,

"stddev": 0.000000,

"count": 82673752.000000,

"sum": 0.000000,

"variance": 0.000000,

"histogram": {

"minimum": 0.000000,

"maximum": 137.000000,

"counts": [

58138737,

0,

0,

0,

0,

0,

0,

0,

24490375,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

1610,

0,

0,

0,

0,

0,

0,

0,

43030

]

},

"mostFrequentValues": [

{

"value": 0.000000,

"count": 58138737

},

{

"value": 8.000000,

"count": 24490375

},

{

"value": 136.000000,

"count": 43030

},

{

"value": 128.000000,

"count": 1610

}

]

}

**Example: Returns statistics (LIDAR point cloud)**

{

"min": 17.0,

"max": 52.0,

"avg": 17.41348,

"stddev": 2.606183,

"count": 3799022.0,

"sum": 66154192.0,

"variance": 6.792191,

"histogram": {

"minimum": 17.0,

"maximum": 52.0,

"counts": [

3704673,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

49581,

43733,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

0,

394,

341,

300

]

},

"mostFrequentValues": [

{

"value": 17.0,

"count": 3704673

},

{

"value": 33.0,

"count": 49581

},

{

"value": 34.0,

"count": 43733

},

{

"value": 49.0,

"count": 394

},

{

"value": 50.0,

"count": 341

},

{

"value": 51.0,

"count": 300

}

]

}

I3S point cloud scene layer: labels

Optionally, the statistics document may contain labeling information for the attribute values.

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| labels | [label](https://github.com/cnreediii/i3s-spec/blob/master/docs/2.0/label.pcsl.md)[] | Array of string label/value pairs. Used when attribute represents a set of values. For example, ClassCode. |
| bitfieldLabels | [bitfieldlabel](https://github.com/cnreediii/i3s-spec/blob/master/docs/2.0/bitfieldlabel.pcsl.md)[] | Array of string label/bitNumber pairs. This is useful when the attribute represent a bitfield. For example, FLAGS. |

**Examples**

**Example: Class Code labels (LIDAR data)**

{

"labels": [

{

"value": 1.0,

"label": "Unclassified"

},

{

"value": 2.0,

"label": "Ground"

},

{

"value": 3.0,

"label": "Low Vegetation"

},

{

"value": 4.0,

"label": "Medium Vegetation"

},

{

"value": 5.0,

"label": "High Vegetation"

},

{

"value": 6.0,

"label": "Building"

},

{

"value": 7.0,

"label": "Low Point(noise)"

},

{

"value": 8.0,

"label": "Model Key"

},

{

"value": 9.0,

"label": "Water"

},

{

"value": 10.0,

"label": "Rail"

},

{

"value": 11.0,

"label": "Road Surface"

}

]

}

**Example: Flags labels (LIDAR data)**

{

"bitfieldLabels": [

{

"bitNumber": 3,

"label": "Overlap"

},

{

"bitNumber": 7,

"label": "Edge"

}

]

}

**I3S point cloud scene layer: Statistics Label**

**Properties**

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| **value** | number | Value |
| **label** | string | Label string |

*Note: properties in* ***bold*** *are mandatory.*

**Example: Class Code label**

{

"value": 2,

"label": "Ground"

}

I3S point cloud scene layer: Bit Field Label

Example for LiDAR data:

| **Bit Number** | **Label** | **description** |
| --- | --- | --- |
| 0 | Synthetic | If set then this point was created by a technique other than LIDAR collection such as digitized from a photogrammetric stereo model or by traversing a waveform |
| 1 | Key-Point | If set, this point is considered to be a model key-point and thus generally should not be withheld in a thinning algorithm. |
| 2 | Withheld | If set, this point should not be included in processing (synonymous with Deleted). |
| 3 | Overlap | If set, this point is within the overlap region of two or more swaths or takes. Setting this bit is not mandatory (unless, of course, it is mandated by a particular delivery specification) but allows Classification of overlap points to be preserved. |
| 4 | Scan Channel 0 | Scanner Channel is used to indicate the channel (scanner head) of a multichannel system. Channel 0 is used for single scanner systems |
| 5 | Scan Channel 1 | Scanner Channel is used to indicate the channel (scanner head) of a multichannel system. |
| 6 | Scan Direction | The Scan Direction Flag denotes the direction at which the scanner mirror was traveling at the time of the output pulse. A bit value of 1 is a positive scan direction, and a bit value of 0 is a negative scan direction (where positive scan direction is a scan moving from the left side of the in-track direction to the right side and negative the opposite). |
| 7 | Edge of flight line | The Edge of Flight Line data bit has a value of 1 only when the point is at the end of a scan. It is the last point on a given scan line before it changes direction or the mirror facet changes. Note that this field has no meaning for 360° Field of View scanners (such as Mobile LIDAR scanners) and should not be set |

Properties

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| **bitNumber** | integer | Bit number (0 is LSB) |
| **label** | string | Label string |

*Note: properties in* ***bold*** *are required*

**Example: Flag label**

{

"bitNumber": 3,

"label": "Overlap"

}

Annex J: Examples of 3dSceneLayerInfo

**Example: 3D Scene Layer info for 3D object scene layer**

{"id":0,"version":"F3950549-438B-4B31-A1AB-710067EECD6A","name":"Pipes","serviceUpdateTimeStamp":

{"lastUpdate":1539204480000},"href":"./layers/0","layerType":"3DObject","spatialReference":

{"wkid":103130,"latestWkid":6551},"ZFactor":0.30480060960121924,"alias":"Pipes","description":"Pipes","copyrightText":"","capabilities":

["View","Query"],"cachedDrawingInfo":{"color":true},"drawingInfo":{"renderer":{"type":"simple","symbol":

{"type":"MeshSymbol3D","symbolLayers":[{"type":"Fill","material":{"color":

[255,170,0],"transparency":0,"colorMixMode":"multiply"},"edges":{"type":"solid","color":

[0,0,0],"size":1,"transparency":0,"extensionLength":0}}]}}},"popupInfo":{"title":"{DocName}","mediaInfos":[],"fieldInfos":

[{"fieldName":"OBJECTID\_1","visible":true,"isEditable":false,"label":"OBJECTID\_1"},

{"fieldName":"Category","visible":true,"isEditable":true,"label":"Category"},

{"fieldName":"Family","visible":true,"isEditable":true,"label":"Family"},

{"fieldName":"FamilyType","visible":true,"isEditable":true,"label":"FamilyType"}

{"fieldName":"ObjectId","visible":true,"isEditable":true,"label":"ObjectId"},

{"fieldName":"BldgLevel","visible":true,"isEditable":true,"label":"BldgLevel"}

{"fieldName":"BldgLevel\_Elev","visible":true,"isEditable":true,"label":"BldgLevel\_Elev"}

{"fieldName":"BldgLevel\_IsBuildingStory","visible":true,"isEditable":true,"label":"BldgLevel\_IsBuildingStory"}

{"fieldName":"BldgLevel\_RoomOffset","visible":true,"isEditable":true,"label":"BldgLevel\_RoomOffset"}

{"fieldName":"CreatedPhase","visible":true,"isEditable":true,"label":"CreatedPhase"},

{"fieldName":"DemolishedPhase","visible":true,"isEditable":true,"label":"DemolishedPhase"}

{"fieldName":"ElementType","visible":true,"isEditable":true,"label":"ElementType"}

{"fieldName":"Discipline","visible":true,"isEditable":true,"label":"Discipline"}

{"fieldName":"Function","visible":true,"isEditable":true,"label":"Function"}

{"fieldName":"DocPath","visible":true,"isEditable":true,"label":"DocPath"}

{"fieldName":"DocVer","visible":true,"isEditable":true,"label":"DocVer"}

{"fieldName":"DocUpdate","visible":true,"isEditable":true,"label":"DocUpdate"},

{"fieldName":"Transparency","visible":true,"isEditable":true,"label":"Transparency"}

{"fieldName":"BaseCategory","visible":true,"isEditable":true,"label":"BaseCategory"}

{"fieldName":"AssemblyCode","visible":true,"isEditable":true,"label":"AssemblyCode"}

{"fieldName":"AssemblyDesc","visible":true,"isEditable":true,"label":"AssemblyDesc"}

{"fieldName":"OmniClass","visible":true,"isEditable":true,"label":"OmniClass"}

{"fieldName":"OmniClassDescription","visible":true,"isEditable":true,"label":"OmniClassDescription"}

{"fieldName":"Mark","visible":true,"isEditable":true,"label":"Mark"}

{"fieldName":"Typ\_Mark","visible":true,"isEditable":true,"label":"Typ\_Mark"}

{"fieldName":"DocName","visible":true,"isEditable":true,"label":"DocName"}

{"fieldName":"WidthDiameter","visible":true,"isEditable":true,"label":"WidthDiameter"}

{"fieldName":"SystemId","visible":true,"isEditable":true,"label":"SystemId"}

{"fieldName":"SystemName","visible":true,"isEditable":true,"label":"SystemName"},

{"fieldName":"SystemType","visible":true,"isEditable":true,"label":"SystemType"}

{"fieldName":"SystemClass","visible":true,"isEditable":true,"label":"SystemClass"}

{"fieldName":"CalculatedSize","visible":true,"isEditable":true,"label":"CalculatedSize"}

{"fieldName":"Comments","visible":true,"isEditable":true,"label":"Comments"}

{"fieldName":"Flow","visible":true,"isEditable":true,"label":"Flow"}],"popupElements":[{"fieldInfos"

[{"fieldName":"OBJECTID\_1","visible":true,"isEditable":false,"label":"OBJECTID\_1"}

{"fieldName":"Category","visible":true,"isEditable":true,"label":"Category"}

{"fieldName":"Family","visible":true,"isEditable":true,"label":"Family"}

{"fieldName":"FamilyType","visible":true,"isEditable":true,"label":"FamilyType"}

{"fieldName":"ObjectId","visible":true,"isEditable":true,"label":"ObjectId"}

{"fieldName":"BldgLevel","visible":true,"isEditable":true,"label":"BldgLevel"}

{"fieldName":"BldgLevel\_Elev","visible":true,"isEditable":true,"label":"BldgLevel\_Elev"},

{"fieldName":"BldgLevel\_IsBuildingStory","visible":true,"isEditable":true,"label":"BldgLevel\_IsBuildingStory"}

{"fieldName":"BldgLevel\_RoomOffset","visible":true,"isEditable":true,"label":"BldgLevel\_RoomOffset"}

{"fieldName":"CreatedPhase","visible":true,"isEditable":true,"label":"CreatedPhase"}

{"fieldName":"DemolishedPhase","visible":true,"isEditable":true,"label":"DemolishedPhase"}

{"fieldName":"ElementType","visible":true,"isEditable":true,"label":"ElementType"}

{"fieldName":"Discipline","visible":true,"isEditable":true,"label":"Discipline"}

{"fieldName":"Function","visible":true,"isEditable":true,"label":"Function"}

{"fieldName":"DocPath","visible":true,"isEditable":true,"label":"DocPath"}

{"fieldName":"DocVer","visible":true,"isEditable":true,"label":"DocVer"}

{"fieldName":"DocUpdate","visible":true,"isEditable":true,"label":"DocUpdate"}

{"fieldName":"Transparency","visible":true,"isEditable":true,"label":"Transparency"}

{"fieldName":"BaseCategory","visible":true,"isEditable":true,"label":"BaseCategory"}

{"fieldName":"AssemblyCode","visible":true,"isEditable":true,"label":"AssemblyCode"}

{"fieldName":"AssemblyDesc","visible":true,"isEditable":true,"label":"AssemblyDesc"}

{"fieldName":"OmniClass","visible":true,"isEditable":true,"label":"OmniClass"}

{"fieldName":"OmniClassDescription","visible":true,"isEditable":true,"label":"OmniClassDescription"}

{"fieldName":"Mark","visible":true,"isEditable":true,"label":"Mark"}

{"fieldName":"Typ\_Mark","visible":true,"isEditable":true,"label":"Typ\_Mark"}

{"fieldName":"DocName","visible":true,"isEditable":true,"label":"DocName"}

{"fieldName":"WidthDiameter","visible":true,"isEditable":true,"label":"WidthDiameter"},

{"fieldName":"SystemId","visible":true,"isEditable":true,"label":"SystemId"}

{"fieldName":"SystemName","visible":true,"isEditable":true,"label":"SystemName"}

{"fieldName":"SystemType","visible":true,"isEditable":true,"label":"SystemType"}

{"fieldName":"SystemClass","visible":true,"isEditable":true,"label":"SystemClass"},

{"fieldName":"CalculatedSize","visible":true,"isEditable":true,"label":"CalculatedSize"}

{"fieldName":"Comments","visible":true,"isEditable":true,"label":"Comments"}

{"fieldName":"Flow","visible":true,"isEditable":true,"label":"Flow"}],"type":"fields"}],"expressionInfos":

[]},"disablePopup":false,"store":{"id":"7586A65E-5A4A-4FAA-B279-1C97F7C0208B","profile":"meshpyramids","resourcePattern"

["3dNodeIndexDocument","Attributes","SharedResource","Geometry"],"rootNode":"./nodes/root","version":"1.6","extent"

[1816831.76067100465,731679.422988593578,1816950.00551325083,731840.359674587846],"indexCRS":"http://www.opengis.net/def/crs

/EPSG/0/6551","vertexCRS":"http://www.opengis.net/def/crs/EPSG/0/6551","normalReferenceFrame":"vertex-reference

frame","nidEncoding":"application/vnd.esri.i3s.json+gzip; version=1.6","featureEncoding":"application/vnd.esri.i3s.json+gzip;

version=1.6","geometryEncoding":"application/octet-stream; version=1.6","attributeEncoding":"application/octet-stream;

version=1.6","lodType":"MeshPyramid","lodModel":"node-switching","defaultGeometrySchema":{"geometryType":"triangles","header"

[{"property":"vertexCount","type":"UInt32"},

{"property":"featureCount","type":"UInt32"}],"topology":"PerAttributeArray","ordering"

["position","normal","uv0","color"],"vertexAttributes":{"position":{"valueType":"Float32","valuesPerElement":3},"normal"

{"valueType":"Float32","valuesPerElement":3},"uv0":{"valueType":"Float32","valuesPerElement":2},"color"

{"valueType":"UInt8","valuesPerElement":4}},"featureAttributeOrder":["id","faceRange"],"featureAttributes":{"id"

{"valueType":"UInt64","valuesPerElement":1},"faceRange":{"valueType":"UInt32","valuesPerElement":2}}},"textureEncoding"

["image/jpeg","image/vnd-ms.dds"]},"fields":[{"name":"OBJECTID\_1","type":"esriFieldTypeOID","alias":"OBJECTID\_1"},

{"name":"Category","type":"esriFieldTypeString","alias":"Category"}

{"name":"Family","type":"esriFieldTypeString","alias":"Family"}

{"name":"FamilyType","type":"esriFieldTypeString","alias":"FamilyType"}

{"name":"ObjectId","type":"esriFieldTypeString","alias":"ObjectId"}

{"name":"BldgLevel","type":"esriFieldTypeInteger","alias":"BldgLevel"}

{"name":"BldgLevel\_Elev","type":"esriFieldTypeDouble","alias":"BldgLevel\_Elev"}

{"name":"BldgLevel\_IsBuildingStory","type":"esriFieldTypeSmallInteger","alias":"BldgLevel\_IsBuildingStory"}

{"name":"BldgLevel\_RoomOffset","type":"esriFieldTypeDouble","alias":"BldgLevel\_RoomOffset"}

{"name":"CreatedPhase","type":"esriFieldTypeInteger","alias":"CreatedPhase"}

{"name":"DemolishedPhase","type":"esriFieldTypeInteger","alias":"DemolishedPhase"}

{"name":"ElementType","type":"esriFieldTypeString","alias":"ElementType"}

{"name":"Discipline","type":"esriFieldTypeString","alias":"Discipline"}

{"name":"Function","type":"esriFieldTypeInteger","alias":"Function"}

{"name":"DocPath","type":"esriFieldTypeString","alias":"DocPath"}

{"name":"DocVer","type":"esriFieldTypeString","alias":"DocVer"}

{"name":"DocUpdate","type":"esriFieldTypeDate","alias":"DocUpdate"}

{"name":"Transparency","type":"esriFieldTypeDouble","alias":"Transparency"}

{"name":"BaseCategory","type":"esriFieldTypeString","alias":"BaseCategory"}

{"name":"AssemblyCode","type":"esriFieldTypeString","alias":"AssemblyCode"}

{"name":"AssemblyDesc","type":"esriFieldTypeString","alias":"AssemblyDesc"},

{"name":"OmniClass","type":"esriFieldTypeString","alias":"OmniClass"}

{"name":"OmniClassDescription","type":"esriFieldTypeString","alias":"OmniClassDescription"}

{"name":"Mark","type":"esriFieldTypeString","alias":"Mark"},{"name":"Typ\_Mark","type":"esriFieldTypeString","alias":"Typ\_Mark"}

{"name":"DocName","type":"esriFieldTypeString","alias":"DocName"}

{"name":"WidthDiameter","type":"esriFieldTypeDouble","alias":"WidthDiameter"}

{"name":"SystemId","type":"esriFieldTypeString","alias":"SystemId"}

{"name":"SystemName","type":"esriFieldTypeString","alias":"SystemName"}

{"name":"SystemType","type":"esriFieldTypeString","alias":"SystemType"},

{"name":"SystemClass","type":"esriFieldTypeInteger","alias":"SystemClass"}

{"name":"CalculatedSize","type":"esriFieldTypeString","alias":"CalculatedSize"}

{"name":"Comments","type":"esriFieldTypeString","alias":"Comments"}

{"name":"Flow","type":"esriFieldTypeDouble","alias":"Flow"}],"attributeStorageInfo":[{"key":"f\_0","name":"OBJECTID\_1","header"

[{"property":"count","valueType":"UInt32"}],"ordering":["ObjectIds"],"objectIds":{"valueType":"UInt32","valuesPerElement":1}}

{"key":"f\_1","name":"Category","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":"Family","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\_3","name":"FamilyType","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\_4","name":"ObjectId","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\_5","name":"BldgLevel","header"

[{"property":"count","valueType":"UInt32"}],"ordering":["attributeValues"],"attributeValues"

{"valueType":"Int32","valuesPerElement":1}},{"key":"f\_6","name":"BldgLevel\_Elev","header"

[{"property":"count","valueType":"UInt32"}],"ordering":["attributeValues"],"attributeValues"

{"valueType":"Float64","valuesPerElement":1}},{"key":"f\_7","name":"BldgLevel\_IsBuildingStory","header"

[{"property":"count","valueType":"UInt32"}],"ordering":["attributeValues"],"attributeValues"

{"valueType":"Int16","valuesPerElement":1}},{"key":"f\_8","name":"BldgLevel\_RoomOffset","header"

[{"property":"count","valueType":"UInt32"}],"ordering":["attributeValues"],"attributeValues"

{"valueType":"Float64","valuesPerElement":1}},{"key":"f\_9","name":"CreatedPhase","header"

[{"property":"count","valueType":"UInt32"}],"ordering":["attributeValues"],"attributeValues"

{"valueType":"Int32","valuesPerElement":1}},{"key":"f\_10","name":"DemolishedPhase","header"

[{"property":"count","valueType":"UInt32"}],"ordering":["attributeValues"],"attributeValues"

{"valueType":"Int32","valuesPerElement":1}},{"key":"f\_11","name":"ElementType","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\_12","name":"Discipline","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\_13","name":"Function","header"

[{"property":"count","valueType":"UInt32"}],"ordering":["attributeValues"],"attributeValues"

{"valueType":"Int32","valuesPerElement":1}},{"key":"f\_14","name":"DocPath","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\_15","name":"DocVer","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\_16","name":"DocUpdate","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\_17","name":"Transparency","header"

[{"property":"count","valueType":"UInt32"}],"ordering":["attributeValues"],"attributeValues"

{"valueType":"Float64","valuesPerElement":1}},{"key":"f\_18","name":"BaseCategory","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\_19","name":"AssemblyCode","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\_20","name":"AssemblyDesc","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\_21","name":"OmniClass","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\_22","name":"OmniClassDescription","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\_23","name":"Mark","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\_24","name":"Typ\_Mark","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\_25","name":"DocName","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\_26","name":"WidthDiameter","header"

[{"property":"count","valueType":"UInt32"}],"ordering":["attributeValues"],"attributeValues"

{"valueType":"Float64","valuesPerElement":1}},{"key":"f\_27","name":"SystemId","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\_28","name":"SystemName","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\_29","name":"SystemType","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\_30","name":"SystemClass","header"

[{"property":"count","valueType":"UInt32"}],"ordering":["attributeValues"],"attributeValues"

{"valueType":"Int32","valuesPerElement":1}},{"key":"f\_31","name":"CalculatedSize","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\_32","name":"Comments","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\_33","name":"Flow","header"

[{"property":"count","valueType":"UInt32"}],"ordering":["attributeValues"],"attributeValues":

{"valueType":"Float64","valuesPerElement":1}}],"statisticsInfo":[{"key":"f\_1","name":"Category","href":"./statistics/f\_1"}

{"key":"f\_2","name":"Family","href":"./statistics/f\_2"},{"key":"f\_3","name":"FamilyType","href":"./statistics/f\_3"}

{"key":"f\_4","name":"ObjectId","href":"./statistics/f\_4"},{"key":"f\_5","name":"BldgLevel","href":"./statistics/f\_5"}

{"key":"f\_6","name":"BldgLevel\_Elev","href":"./statistics/f\_6"},{"key":"f\_7","name":"BldgLevel\_IsBuildingStory","href":"

/statistics/f\_7"},{"key":"f\_8","name":"BldgLevel\_RoomOffset","href":"./statistics/f\_8"}

{"key":"f\_9","name":"CreatedPhase","href":"./statistics/f\_9"},{"key":"f\_10","name":"DemolishedPhase","href":"./statistic

/f\_10"},{"key":"f\_11","name":"ElementType","href":"./statistics/f\_11"},{"key":"f\_12","name":"Discipline","href":"./statistic

/f\_12"},{"key":"f\_13","name":"Function","href":"./statistics/f\_13"},{"key":"f\_14","name":"DocPath","href":"./statistics/f\_14"}

{"key":"f\_15","name":"DocVer","href":"./statistics/f\_15"},{"key":"f\_16","name":"DocUpdate","href":"./statistics/f\_16"}

{"key":"f\_17","name":"Transparency","href":"./statistics/f\_17"},{"key":"f\_18","name":"BaseCategory","href":"./statistics/f\_18"}

{"key":"f\_19","name":"AssemblyCode","href":"./statistics/f\_19"},{"key":"f\_20","name":"AssemblyDesc","href":"./statistics/f\_20"}

{"key":"f\_21","name":"OmniClass","href":"./statistics/f\_21"},{"key":"f\_22","name":"OmniClassDescription","href":"./statistic

/f\_22"},{"key":"f\_23","name":"Mark","href":"./statistics/f\_23"},{"key":"f\_24","name":"Typ\_Mark","href":"./statistics/f\_24"}

{"key":"f\_25","name":"DocName","href":"./statistics/f\_25"},{"key":"f\_26","name":"WidthDiameter","href":"./statistics/f\_26"}

{"key":"f\_27","name":"SystemId","href":"./statistics/f\_27"},{"key":"f\_28","name":"SystemName","href":"./statistics/f\_28"}

{"key":"f\_29","name":"SystemType","href":"./statistics/f\_29"},{"key":"f\_30","name":"SystemClass","href":"./statistics/f\_30"}

{"key":"f\_31","name":"CalculatedSize","href":"./statistics/f\_31"},{"key":"f\_32","name":"Comments","href":"./statistics/f\_32"}

{"key":"f\_33","name":"Flow","href":"./statistics/f\_33"}]}

**Example: 3D Scene Layer info for integrated mesh scene layer**

{

"id": 0,

"version": "3d3c7b51-6336-4893-b484-ad118775bcce",

"name": "Export2",

"href": "./layers/0",

"layerType": "IntegratedMesh",

"ZFactor": 1.0,

"spatialReference": {

"wkid": 4326,

"latestWkid": 4326

},

"alias": “Export2",

"description": "Vricon 3D Surface Model",

"copyrightText": "Limited in accordance with the accompanying Vricon EULA",

"capabilities": [

"View",

"Query"

],

"store": {

"id": "e9ecfade-0d85-4dd7-abb5-a3b0a07b9fd7",

"profile": "meshpyramids",

"resourcePattern": [

"3dNodeIndexDocument",

"SharedResource",

"Geometry",

"Attributes"

],

"rootNode": "./nodes/root",

"version": "1.4",

"extent": [

-106.5054122583675,

38.994677805489189,

-103.99630101552692,

39.996971340614706

],

"indexCRS": "http://www.opengis.net/def/crs/EPSG/0/4326",

"vertexCRS": "http://www.opengis.net/def/crs/EPSG/0/4326",

"nidEncoding": "application/vnd.esri.i3s.json+gzip; version=1.4",

"featureEncoding": "application/vnd.esri.i3s.json+gzip; version=1.4",

"geometryEncoding": "application/octet-stream; version=1.4",

"attributeEncoding": "application/octet-stream; version=1.4",

"textureEncoding": [

"image/jpeg",

"image/vnd-ms.dds"

],

"lodType": "MeshPyramid",

"lodModel": "node-switching",

"defaultGeometrySchema": {

"geometryType": "triangles",

"header": [

{

"property": "vertexCount",

"type": "UInt32"

},

{

"property": "featureCount",

"type": "UInt32"

}

],

"topology": "PerAttributeArray",

"ordering": [

"position",

"normal",

"uv0",

"color"

],

"vertexAttributes": {

"position": {

"valueType": "Float32",

"valuesPerElement": 3

},

"normal": {

"valueType": "Float32",

"valuesPerElement": 3

},

"uv0": {

"valueType": "Float32",

"valuesPerElement": 2

},

"color": {

"valueType": "UInt8",

"valuesPerElement": 4

}

},

"featureAttributeOrder": [

"id",

"faceRange"

],

"featureAttributes": {

"id": {

"valueType": "UInt64",

"valuesPerElement": 1

},

"faceRange": {

"valueType": "UInt32",

"valuesPerElement": 2

}

}

}

}

}

1. https://creativecommons.org/licenses/by-sa/4.0/ [↑](#footnote-ref-1)
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> [↑](#footnote-ref-2)
3. The specification for accessing I3S resources as Scene Service REST endpoints are described in Annex E (informative). [↑](#footnote-ref-3)
4. <https://en.wikipedia.org/wiki/3D_modeling> (February 7, 2017) [↑](#footnote-ref-4)
5. <https://en.wikipedia.org/wiki/Bounding_sphere> (February 12, 2017) [↑](#footnote-ref-5)
6. <http://mathworld.wolfram.com/NormalVector.html> (March 3, 2017) [↑](#footnote-ref-6)
7. An Exact Algorithm for Finding Minimum Oriented Bounding Boxes. <http://clb.demon.fi/minobb/minobb.html> (June 1, 2015) [↑](#footnote-ref-7)
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. [↑](#footnote-ref-8)
9. <https://en.wikipedia.org/wiki/Texture_atlas> (February 19, 2017) [↑](#footnote-ref-9)
10. <https://en.wikipedia.org/wiki/Texture_mapping> (February 19, 2017) [↑](#footnote-ref-10)
11. <http://www.rozengain.com/blog/2007/08/26/uv-coordinate-basics/> (February 19, 2017) [↑](#footnote-ref-11)
12. <https://en.wikipedia.org/wiki/UV_mapping> (February 9, 2017) [↑](#footnote-ref-12)
13. <https://en.wikipedia.org/wiki/Vertex_(geometry)#Vertices_in_computer_graphics> (February 9, 2017) [↑](#footnote-ref-13)
14. NOTE: The JSON/HTML examples for these profiles are currently in the public Esri Git repo but will be replicated in an OGC repository if and when this candidate community standard is approved and published on the OGC website. [↑](#footnote-ref-14)
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] [↑](#footnote-ref-15)
16. http://portal.opengeospatial.org/files/?artifact\_id=13227 [↑](#footnote-ref-16)
17. <http://docs.opengeospatial.org/is/12-063r5/12-063r5.html> [↑](#footnote-ref-17)
18. The text in this paragraph is extracted verbatim from 12-063r5. [↑](#footnote-ref-18)
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. [↑](#footnote-ref-19)
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". [↑](#footnote-ref-20)
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= [↑](#footnote-ref-21)
22. https://en.wikipedia.org/wiki/Quadtree [↑](#footnote-ref-22)
23. https://en.wikipedia.org/wiki/Octree [↑](#footnote-ref-23)
24. https://en.wikipedia.org/wiki/R-tree [↑](#footnote-ref-24)
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. [↑](#footnote-ref-25)
26. JavaScript: ArrayBufferView is an abstract type that is the base for the following types: [DataView](http://www.javascripture.com/DataView), [Float32Array](http://www.javascripture.com/Float32Array), [Float64Array](http://www.javascripture.com/Float64Array), [Int8Array](http://www.javascripture.com/Int8Array), [Int16Array](http://www.javascripture.com/Int16Array), [Int32Array](http://www.javascripture.com/Int32Array), [Uint8Array](http://www.javascripture.com/Uint8Array), [Uint8ClampedArray](http://www.javascripture.com/Uint8ClampedArray), [Uint16Array](http://www.javascripture.com/Uint16Array), [Uint32Array](http://www.javascripture.com/Uint32Array). [↑](#footnote-ref-26)
27. https://en.wikipedia.org/wiki/S3\_Texture\_Compression [↑](#footnote-ref-27)
28. <https://en.wikipedia.org/wiki/UV_mapping> for more information. [↑](#footnote-ref-28)
29. 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. [↑](#footnote-ref-29)
30. <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 [↑](#footnote-ref-30)
31. https://en.wikipedia.org/wiki/Zip\_(file\_format) [↑](#footnote-ref-31)
32. https://en.wikipedia.org/wiki/Gzip [↑](#footnote-ref-32)