Esri Indexed 3d Scene (i3s/i3d)   
Format Specification

Version 1.0.20, 2013-06-07  
Esri R&D Center Zurich

This document specifies the delivery format used for the new ArcGIS 3D Scene Server. The first sections explain the conceptual structure of i3s, while the latter sections provide a detailed implementation-level view.

# Requirements

The Esri Indexed 3d Scene (i3s) format and the corresponding package format (i3d) are specified to fulfill this set of requirements:

1. **User Experience first:** Support a very good user experience – high interactivity, fast display, rendering of visually relevant features first
2. **Scalability:** Support very large scenes, with global extent and a very large number of features (up to 1 billion), as well as very heavy features
3. **Reusability:** Be useable both as the delivery format of the 3d Scene Server and as a format stored in a local file or database
4. **Level of Detail:** Support discrete Level of Detail concepts for generalization of very large/heavy features and for “semantic” Level of Detail approaches
5. **Distribution:** Allow distribution of resources in very large data sets
6. **Merging:** Allow combination/merging with data from other scene data sets
7. **User-controllable symbology:** Support client-side symbology rendering
8. **Extensibility:** Be extensible to support new features (e.g. geometry types) and new platforms (e.g. by allowing definition of different materials/shaders)
9. **Web Friendliness**, i.e. easy to handle and parse by web clients by using JSON and current web standards
10. **Compatibility:** Have a single structure that is useable by all ArcGIS Desktop, Web and native apps, cross platform and cross device usage, map well to GL APIS
11. **Declarative**: limit how much specific knowledge on the client-side is needed for format support (e.g. Index generation method only needs to be known while writing the format)
12. **Follow REST/JSON API best practices:** “Hypertext as the Engine of Application State” – make all resources navigable using hrefs from relevant other resources.

Some of these requirements (especially 8, 9, 10 and 12) are shared with the Khronos glTF[[1]](#footnote-1) format, which is an upcoming standard for transferring 3D content. In this version of i3s, the two formats share the specification of Geometry Typed Arrays.

# The Content – what goes into an i3s file?

The i3s format supports the different types of 2D and 3D content needed for 3D GIS scenes, ranging from 3D feature meshes to 3D point clouds and 2D point/line/polygon features. All content types supported are listed in table 1.

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Name *(example)*** | **Display Type** | **Fields** |
| 1 | Regular point array *(Grid)* | Triangles | No |
| 2 | Pre-Triangulated irregular point array *(TIN)* | Triangles | No |
| 3 | Integrated Mesh *(Acute3D)* | Triangles | Yes |
| 4 | Individual Feature Mesh *(Multipatch)* | Triangles | Yes |
| 5 | Point Cloud *(LAS)* | Points | Yes[[2]](#footnote-2) |
| 6 | Point Features *(GIS data)* | Points/Triangles | Yes |
| 7 | Line Features *(GIS data)* | Lines/Triangles | Yes |
| 8 | Polygon Features *(GIS data)* | Triangles | Yes |

Table 1: i3s Content Types supported in i3s

A single cache can contain data from only one content type, as the different content types typically require different indexing and Level of Details methods to perform best. In many cases their schema also differs substantially. However, a single cache can contain multiple layers that share the same content type. Effectively these layers will share the same index, but they can still be accessed individually. This reduces the number of calls to a 3D Scene Service, local database or the file system that need to be made by the client drastically and furthermore allows reducing the total data volume. In addition, layers in a shared cache can share resources, such as instance geometries.

# The Index Structure

Esri i3s is, as the name implies, an indexed, partitioned 3D Scene format with some similarities to regionated KML[[3]](#footnote-3) or X3D Earth. The purpose of any index is to allow fast access to (blocks of) relevant data. In an Indexed 3D Scene, the spatial extent is split into regions with a roughly equal amount of data in them, and an access data structure – the actual index – allows the client and the server to quickly discover which data the client actually needs. Such a region of a 3D Scene is called a ***Node***. Node creation is capacity driven – the smaller the node capacity is, the smaller the spatial extent of each node will be.

All Nodes have an ID that is unique throughout a cache. The ID format used is that of a treekey, i.e. the key directly indicates the position of the node in the tree. Treekeys allow sorting all resources on a single dimension and usually maintaining 2D spatial proximity in the 1D ordering. The root node always gets ID “0”. All further nodes get keys according to the pattern shown in Figure 1. Please note that only 1..9 values are supported; thus, no indexing scheme can be used that assigns more than 9 children per parent.

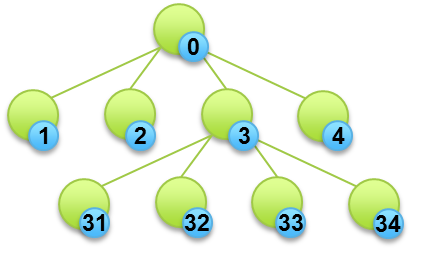
The root node, inner nodes and leaf nodes may carry actual feature data. Inner nodes only carry feature data when features with highly differing spatial sizes are present or when Level of Detail (LoD) feature trees are used.

Figure 1: Tree Keys for a small Quadtree

The i3s format itself is agnostic to the specific indexing scheme used. Methods such as Quadtrees, Octtrees or R-Trees are equally supported. In the format, each node in the index declares all relationships to the parent, siblings and children in the index. Together with information on spatial extents and feature population, these structures allow a client to quickly discover where interesting data resides and to pick the data that is visually representative as well as sufficiently accurate.

# Level of Detail Concept

The Level of Detail concept introduced with this format specification covers several use cases, including splitting up very heavy features such as detailed buildings, very large features (coastlines, rivers, infrastructure), thinning/clustering for optimized visualization and semantic LODs, i.e. the usage of explicit, authored representations to be used for different viewing ranges.

|  |  |  |  |
| --- | --- | --- | --- |
| **Concept** | **Definition** | **Examples** |  |
| *Discrete* | *Multiple representations*, a more detailed one fully replaces a co­arser representation | Image Pyramid, Height map pyramid, Line/Polygon Generalization | http://ggg.udg.edu/skylineEngine/docs/icons/userFriendlyBuildingGraphs.png |
| *Continous* | *Single representation* that is refined continously | SMTerrain, TVTerrain, BitLOD, Progressive Meshes |  |
| *Semantic* | *Independent models* for the same feature | CityGML | http://www.cadmagazine.nl/dir_upload/site/a0799b3c8c7d394b284b80ea5cf40253/101/d2889541_BIMstandaard096-03.gif |

Table 2: LoD concepts support in i3s

Thus the format support for LoD is rather abstract. i3s supports a feature-based Level of Detail approach, i.e. each feature in a node can have higher-detail or lower-detail representations. This feature-based approach has the following properties:

* A Feature can participate in a so-called LoD tree.
* A LoD tree has a single root feature, which has a set of n lodChildren.
* The client receives LoD information with the Node Index Document already, allowing making a choice whether to load a certain resource or not, and also to later identify which features replace which other features.
* lodChildren are guaranteed to be either in the same node or in a direct child node.
* Each Feature that participates in a LoD tree has a rank from 1..d.
* Each Feature that participates in a LoD tree and has a rank > 1 has a rootFeature reference. This reference enables the client to detect which features represent a single object, e.g. for picking purposes.

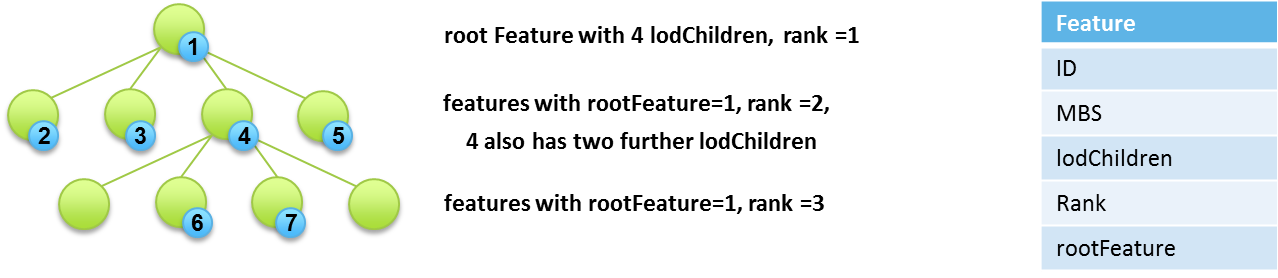
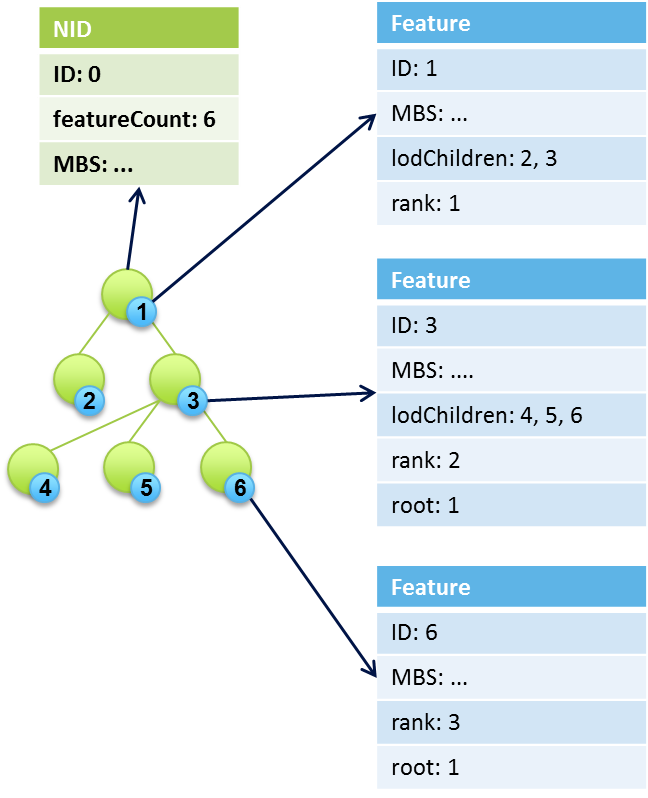


Figure 2: Structure of a Feature LoD tree

In cases where terrain data or integrated meshes are encoded in i3s, it is expected to have a single feature per node, with LoD children in the direct descendants, filling up the entire index with representations. The following figure shows an example of this:

In this example, from root to leaf nodes, each node carries a single feature, for a total count of six nodes and six features. Each of the features that is not a in a root node has a set of lodChildren, with the same set size as the number of node children.

Such data is either created during the cache creation process or predefined by the data provider, as it is the case with Acute3D data.

# Coordinate Reference Systems

i3s layers/stores have to fulfill several in part conflicting requirements when it comes to the selection of spatial reference systems to use:

* Minimal reprojection on client side needed (such as “bending” of large features to the ellipsoid from a projected CRS to an internal geocentric CRS)
* Support very large extent data sets
* Ensure consistency between nodes of a single layer
* Ensure consistency of positions across multiple layers, potentially created from different source projections
* Render easily in Scene and Globe modus
* Support local data with very high positional accuracy
* Support global data sets with high positional accuracy
* Vertex positions have to be meter-based since CityEngine is meter-based

To match these requirements, the following approach is taken for this version of the format:

1. **Metadata**: We use a single, global (bounds -180.0000, -90.0000, 180.0000, 90.0000) Geographic CRS for horizontal location. Elevation and node minimum bounding sphere radius are given in meters. Allowed EPSG codes:
   1. EPSG:4326 (WGS84)
2. **Vertex Positions**: We allow the use of projected CRS, with meter-based x,y,z axes and with a per-node offset (from the center point of the node’s minimum bounding sphere) and using the WGS84 datum. Allowed EPSG codes:
   1. EPSG:32601 to EPSG:32660, EPSG:32701 to EPSG32760 (UTM)
   2. EPSG:3395 (World Mercator) for very large extent datasets (~12° to 360° horizontal extent)

# Structure of format resources

The i3s format contains different components – node index documents (NIDs), feature data, textures, geometry and resources shared across features of a given node. Feature Data, textures, geometry and shared resources are all called resources and are always attached to a node.

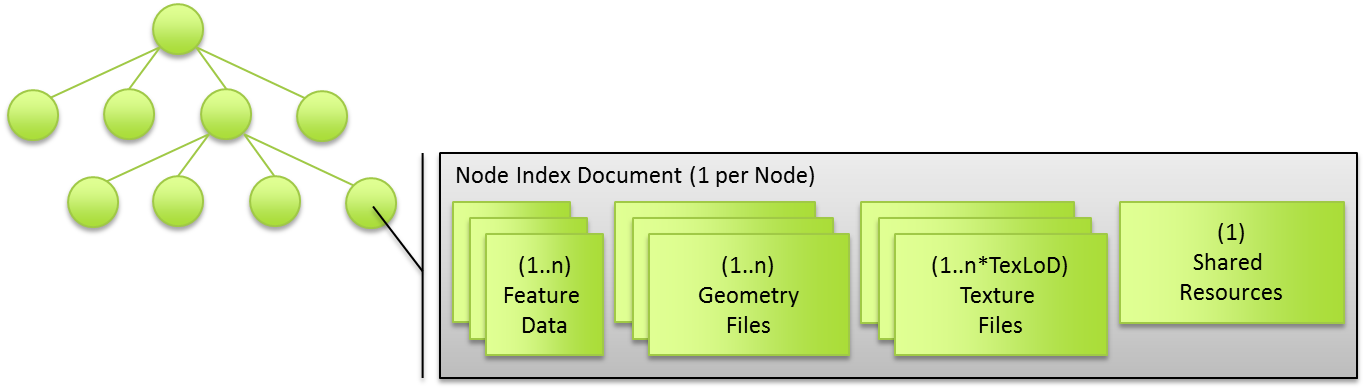
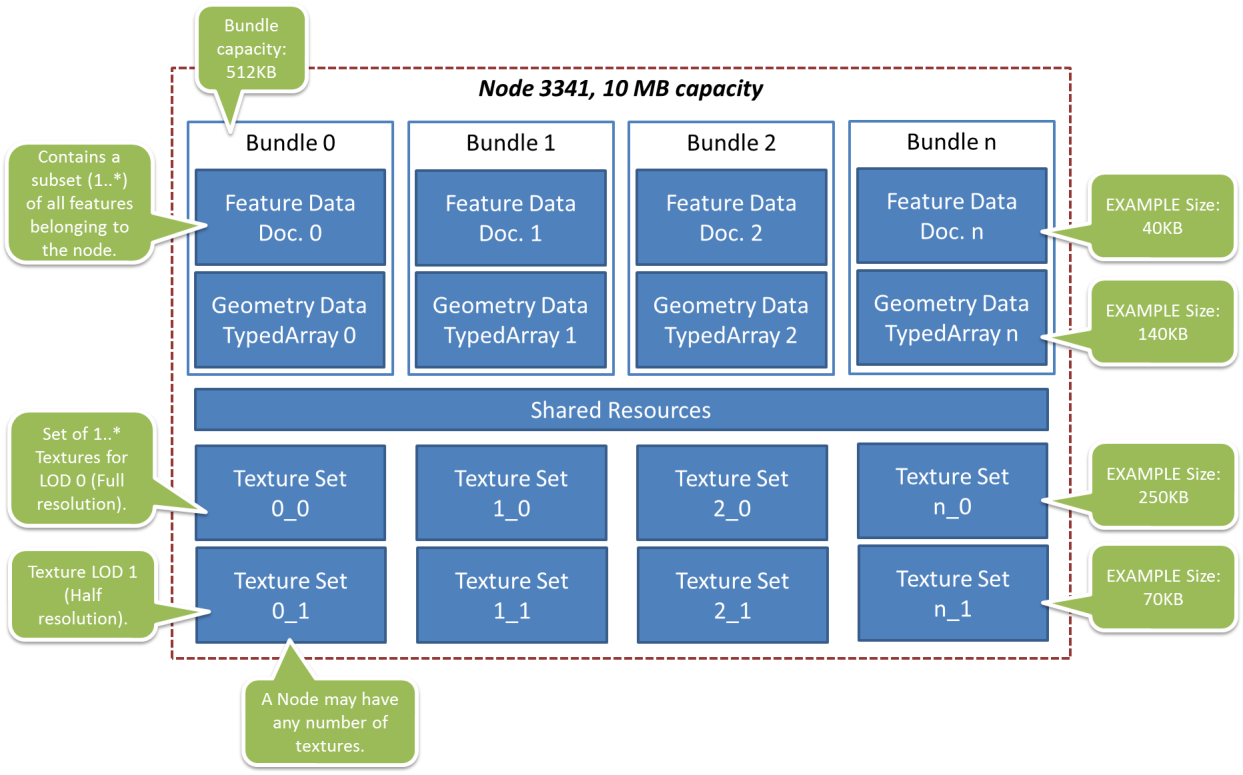


Figure 3: Structure of a single Node and its attached Resources

Per node, there is exactly one Node Index Document and one Shared Resource. Feature Data, Geometry files and Texture files are split into bundles for optimal network transfer and client-side reactivity. This allows balancing between index size, feature splitting (with a relatively large node capacity between 1MB and 10MB) and optimal network usage (with a smaller bundle size, usually in the range of 64kB to 512kB). There are always an equal number ***n*** of Feature Data and Geometry resources, and each set contains the corresponding data elements to be able to render a complete feature. Textures are not tightly coupled to bundle due to the fact that they can also be in the node as part of a shared resource that bubbled up. For each texture in the Node (***m***), the number of Texture resources created is then equal ***m\*Texture LoD Steps***. The following figure illustrates an example set of bundles within a node:



# Definition of resources

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

## 3dSceneServiceInfo.js

The 3dSceneServiceInfo file is a JSON file that describes the capability and data sets offered by an instance of a 3dSceneService instance.

The 3dSceneServiceInfo has the following structure:



Figure 4: Logical format of the 3dSceneServiceInfo document

This file is not generated by the authoring tools and is not part of a Bundled i3s file. It is generated solely by the 3D Scene Server for each service instance. Its description is contained here only for reference.

### Class 3dSceneServiceInfo

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

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| serviceName | string | The type of the service; always 3dSceneService. |
| serviceVersion | String | The version of the service protocol/REST endpoint. |
| supportedBindings | String[1..\*] | the list of bindings, should we ever need to add new bindings in addition to the REST binding initially supported |
| supportedOperations | String[1..3] | Supported profiles of the service from the choice {"Base", "Dynamic", "Editing"}. |

Table 3: Attributes of the Class 3dSceneServiceInfo within the 3dSceneServiceInfo document

### Class Layer

In the 3dSceneServiceInfo document, the Layer object provides a reference so that clients know which Layers are served by a given service.

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| id | long | The ID of this layer, unique within a 3dSceneService. |
| name | String | The version of the service protovol/REST endpoint. |
| alias | String[1..\*] | the list of bindings, should we ever need to add new bindings in addition to the REST binding initially supported |
| href | String[1..3] | The relative URL to the Layer resource giving full information on the Layer's schema and drawing info |

Table 4: Attributes of the Class Layer within the 3dSceneServiceInfo document

### Class Store

While Layers are the user-visible entry point to the 3dSceneServer resources (for web scene authoring and viewing), internally the service uses so-called stores. A store can contain 1..\* many layers, which will share a common index and set of nodes, as well as resources. 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.

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| id | UUID | A Store ID, unique across a SceneServer. Enables the client to discover which layers a part of a common store, if any. |
| rootNode | URL | relative URL to root node resource. |
| version | String | Format version of this resource; used here again if this store hasn't been served by a 3D Scene Server. |
| geometryType | String | The geometry type of this store; selected from {FeatureMesh, IntegratedMesh, RasterTerrain, TINTerrain, Point, Line, Polygon, PointCloud} |
| extent | Double[4] | The spatial extent of this store, in the horizontal indexCRS |
| indexCRS | URL | The horizontal CRS used for all minimum bounding spheres (mbs) in this store, identified by a OGC URL. |
| positionCRS | URL | The horizontal CRS used for all "vertex positions" in this store, identified by a OGC URL. |
| nidEncoding | MIMEType | MIME type for the encoding used for the Node Index Documents; format: application/vnd.esri.i3s.json+gzip; version=1.0 |
| featureEncoding | MIMEType | MIME type for the encoding used for the Feature Data Resources |
| geometryEncoding | MIMEType | MIME type for the encoding used for the Geometry Resources |
| textureEncoding | MIMEType | MIME type for the encoding used for the Texture Resources |
| indexingScheme | String | Indexing Scheme used; selected from {esriRTree, QuadTree, AGOLTilingScheme} |
| featureOrdering | String[1..3] | Ordered list of keywords indicating the ordering scheme applied to sort features within a node; selected from {ID, Prominence, Layer} |

Table 5: Attributes of the Class Store within the 3dSceneServiceInfo document

## 3dSceneLayerInfo.js

The 3dSceneLayerInfo file is a JSON file that describes the properties of a single layer in a store, including the default symbology to use. It shares the definition of this default symbology with the web scene item JSON file. The 3dSceneLayerInfo has the following structure:



Figure 5: Logical format of the 3dSceneLayerInfo document

### Class 3dSceneLayerInfo

The 3dSceneLayerInfo is the major object in the 3dSceneLayerInfo document. There is always exactly one 3dSceneLayerInfo object in the document, which describes a Layer.

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| id | long | Unique numeric ID of the Layer. |
| name | string | The name of this layer. |
| alias | string | The display alias to be used for this layer. |
| description | string | Description string for this layer. |
| copyrightText | string | An array of four doubles, corresponding to x, y, z and radius of the minimum bounding sphere of a node. |
| pickable | boolean | True if the layer is supposed to be pickable. |
| capabilities | String[1..3] | Capabilities from the Set {View, Query, Edit} that are possible on this layer. If not served by a 3DSceneServer (e.g. exported by CE), "View" only. |

Table 3: Attributes of the Class Node within the NodeIndexDocument

### Class Store

For the Store class, please refer to the description provided with the 3dSceneServiceInfo document.

### Class Field

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

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| name | string | The name of the field. |
| type | string | The type of the field, from this enum: {esriFieldTypeBlob, esriFieldTypeDate, esriFieldTypeDouble, esriFieldTypeGeometry, esriFieldTypeGlobalID, esriFieldTypeGUID, esriFieldTypeInteger, esriFieldTypeOID, esriFieldTypeRaster, esriFieldTypeSingle, esriFieldTypeSmallInteger, esriFieldTypeString, esriFieldTypeXML} |
| alias | string | The display alias to be used for this field. |

Table 7: Attributes of the Class Node within the NodeIndexDocument

### Class DrawingInfo

DrawingInfo and the associated classes contain the default symbology for this Layer. This part matches the Cartographic Information Model as used by the other Esri REST services.

## 3dNodeIndexDocument.js

The 3dNodeIndexDocument JSON file describes a single index node within a store, with links to other nodes (children, sibling, and parent), metadata such as its spatial extent and optionally a list of features that the node contains. While this makes a relatively heavy node document, it also means clients have a rich set of information to use to further decide which data to retrieve. The features list already provides sufficient data for simple visualization by rendering the centroids as point features or the Minimum Bounding Sphere as spheres.

The 3dNodeIndexDocument has the following structure:



Figure 6: Logical format of the 3dNodeIndexDocument

### Class Node

The Node is the major object in the NID. There is always exactly one Node object in a NID.

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| id | long | Tree Key ID, unique within the store. ID 0 is always the root node. |
| level | int | Explicit level of this node within the index tree. |
| href | string | The relative URL to the resource from which this NID comes. |
| version | UUID | The version (store update session ID) of this node. |
| mbs | double[4] | An array of four doubles, corresponding to x, y, z and radius of the minimum bounding sphere of a node. |
| precision | double | The "epsilon" value for a node; i.e. the maximum error introduced through generalization of features, relative to the diameter of the MBS of this node. For all nodes/stores without LOD features, this value can be 0.0. |
| created | timestamp | Creation date of this node in UTC. |
| expires | timestamp | Expiration date of this node in UTC. |
| transform | Double[16] | 3D (4x4) transformation matrix expressed as a linear array of 16 values. |

Table 8: Attributes of the Class Node within the NodeIndexDocument

### Class NodeReference

A NodeReference is a pointer to another node – the parent, a child or a neighbor. NodeReferences contain a relative URL pointing to the referenced NID, as well as a set of metainformation 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 | long | Tree Key ID of the referenced node. |
| mbs | double[4] | An array of four doubles, corresponding to x, y, z and radius of the minimum bounding sphere of the referenced node. |
| href | string | The relative URL to the referenced node resource. |
| version | UUID | Version (store update session ID) of the referenced node. |
| featureCount | int | Number of features in the referenced node and its descendants. |

Table 9: Attributes of the Class NodeReference within the NodeIndexDocument

### Class Resource

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

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| href | string | The relative URL to the referenced resource. |
| layerContent | string[1..\*] | The list of layer names that indicates which layer features in the bundle belongs to. The client can use this information to selectively download bundles. |

Table 10: Attributes of the Class Resource within the NodeIndexDocument

### Class Feature

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 |
| id | long | A ID of the Feature, unique within the store (important to note when using Features from multiple stores!) |
| mbs | double[4] | An array of four doubles, corresponding to x, y, z and radius of the minimum bounding sphere of the referenced node. |
| lodChildren | long[0..\*] | IDs of Features in a lower LOD level, which can replace this feature when loaded. |
| rank | int | The LOD level of this feature. Only required for Features that participate in a LOD tree and are not root features of that LOD tree. |
| root | long | The 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. |

Table 11: Attributes of the Class Feature within the NodeIndexDocument

## FeatureData.js

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

Features have the following structure:



Figure 7: Logical format of the FeatureData document

### Class Feature

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

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| id | long | Feature ID, unique within the store. |
| positionOffset | double[2] | The x,y offset used by all vertex positions in this feature. Add these values to the feature geometry vertex positions to get absolute projected coordinates in the positionCRS. |
| Mbb | double[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 positionCRS, 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. |

Table 12: Attributes of the Class NodeReference within the NodeIndexDocument

### Class FeatureAttribute

A FeatureAttribute is a field carrying a value.

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| name | String | The name of the attribute. |
| value | String | The value of the attribute. |

Table 13: Attributes of the Class FeatureAttribute within the NodeIndexDocument

### Class Geometry

Objects of this type include the Geometry definition with material references. Note that one Feature can have multiple geometries, which can consist of multiple parts.

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| type | String | The type denotes whether the following geometry is defined by using array buffer views (ArrayBufferView), as a reference to a shared Resource (SharedResourceReference) or embedded (Embedded). |
| transformation | double[16] | 3D (4x4) transformation matrix expressed as a linear array of 16 values. |

Table 14: Attributes of the Class FeatureAttribute within the NodeIndexDocument

### Class GeometryAttribute

Each GeometryAttribute object is an accessor, i.e. a view, into an arraybuffer.

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| byteOffset | Long | The starting byte position where the required bytes begin |
| count | long | The number of elements. Multiply by number of bytes used for valueType to know how many bytes need to be read. |
| valueType | String | The element type, either UInt8, UInt16, Int16, Int32, Int64 or Float32, Float64 |
| valuesPerElement | short | The number of values need to make a valid element (such as 3 for a xyz position) |

Table 15: Attributes of the Class FeatureAttribute within the NodeIndexDocument

## SharedResources.js

Shared resources are models or textures that can be shared among features within the same store. They are stored as a JSON file entirely, comparable to the encoding used for geometry and textures in a 3ws 2.2 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.

### Class MaterialDefinition

MaterialDefinitions describe how a Feature or a set of Features is to be rendered. This includes which textures to use, which shading and which mapping.

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| vertexColors | Boolean | Indicates whether this Material use Vertex Colors. |
| transparency | Float | Indicates whether this Material uses Transparency. |
| reflectivity | Float | Indicates reflectivity of this Material. |
| shininess | Float | Indicates shininess of this Material. |
| ambient | String | Hex-encoded ambient color of this Material. |
| diffuse | String | Hex-encoded diffuse color of this Material. |
| specular | String | Hex-encoded specular color of this Material. |
| renderMode | String | Rendering mode, any one of {textured, solid, untextured, wireframe} |
| type | String | Shader type, such as “Lambert” |

Table 16: Attributes of the Class MaterialDefinition within the SharedResources document

### Class Map

A Map is a set of images, with some parameters specific to the texture/uv mapping to geometries.

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| encoding | MIMEtype | The encoding/content type that is used by all images in this map |
| wrap | String[2] | UV wrapping modes, from {none, repeat, mirror} |
| atlas | Boolean | true if the Map represents a texture atlas. |
| uvSet | String | The name of the UV set to be used as texture coordinates. |

Table 17: Attributes of the Class Map within the SharedResources document

### 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.

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| id | Long | A unique ID for each image. Generated using the BuildID function that is documented in the spec. |
| size | int | 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 | The href to the image. The resource ID is identical to the id. |
| byteOffset | Long | The byte offset of this image in the block in which this texture image resides. |
| length | Long | The length in bytes of this image. |

Table 18: Attributes of the Class Image within the SharedResources document

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)<<44;

   UInt64 l\_h = ((UInt64)h)<<32;

   UInt64 id64 = l\_al + l\_l + l\_w + l\_h + (UInt64)id;

   return id64;

}

### Class ShaderDefinition

ShaderDefinitions are, in version 1.0 of the i3s specification, foreseen as an optional feature to provide API-dependent shader programs with a layer. Their support requires changes to the Scene Server Clients.

### Class SymbolDefinition

For SymbolDefinitions, the same model is used as in the Web Scene Item specification for ArcGIS online. Please refer to that specification when it becomes available.

## Textures.bin

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

## Geometry.bin

The binary geometry attribute file follows the Khronos Typed Array[[4]](#footnote-4) specification in the Editor’s Draft version of 10th April 2013. Citing the overview of that spec:

“*This specification defines an ArrayBuffer type, representing a generic fixed-length binary buffer. It is not possible to manipulate the contents of an ArrayBuffer directly. 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).”*

# Persistence

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

## Exploded Files layout

All resources are stored as individual files on the file system, using a folder structure aligned with the index structure to keep the number of folder and files per folder on a manageable level.

This part of the specification is currently not maintained as it has been superseded by CouchDB and Bundled i3s storage.

## Storage in CouchDB

The 3d Scene Server stores i3s resources in a document-oriented database. After testing, CouchDB was selected as being suitable. Especially large stores benefit from this type of storage.

## Bundled Indexed 3d Scenes

A store can also be delivered as a single file. Such a file takes all the resources and their attachments and stores them as entries in a MIME/multipart file. This format was selected because it is well-known, there is robust support for all languages and clients (browsers included) and it is also used as a ‘dump’ file format for CouchDB. Consequently, a *Bundled Indexed 3D Scene* can be loaded to CouchDB (the scene service storage backend) easily.

## Storage in IndexedDB

*IndexedDB* is a Key-value document store available in many current browsers, such as Firefox, Chrome and Internet Explorer[[5]](#footnote-5). IndexedDB offers a method of storing data client-side and allows indexed database queries against JSON documents.

In IndexedDB, each resource is stored separately, using partial URLs as key as described here:

# Client Access Pattern

This section describes how a client wanting to consume i3s is expected to load and handle resources from the format.

1. Please refer to <https://github.com/KhronosGroup/glTF/> for the current state of the specification. glTF is currently work-in-progress. [↑](#footnote-ref-1)
2. Fields in Point clouds are limited as follows: Per point: numeric only, per cloud: all types allowed [↑](#footnote-ref-2)
3. See <http://code.google.com/p/regionator/wiki/Welcome> for more information. [↑](#footnote-ref-3)
4. Please refer to <http://www.khronos.org/registry/typedarray/specs/latest/> for the latest version of this specification. [↑](#footnote-ref-4)
5. Please refer to <http://caniuse.com/#feat=indexeddb> for the current state of support. [↑](#footnote-ref-5)