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

Version 1.0.13, 2013-04-22  
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 caches
6. **Merging:** Allow combination/merging with data from other scene caches
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.

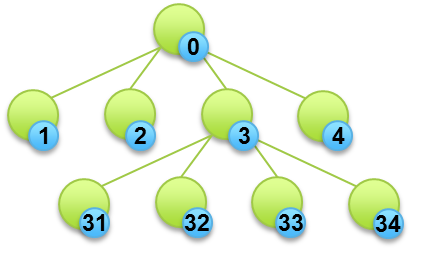
The root node, inner nodes and leaf nodes may carry actual feature data. Inner nodes only carry feature data when features with highly differing spatial sizes are present or when Level of Detail (LoD) feature trees are used.

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.

# 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.

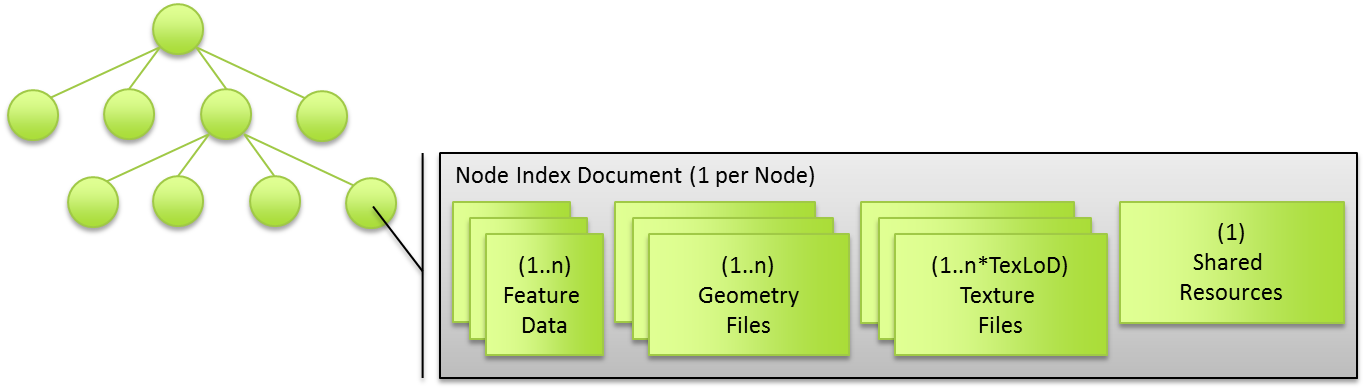


Figure 2: Structure of a single Node and its attached Resources

Per node, there is exactly one Node Index Document and one Shared Resource. Feature Data, Geometry files and Texture files are separated into blocks for optimal network transfer. 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 block size, usually in the range of 64kB to 512kB). There are always an equal number ***n*** of Feature Data and Geometry blocks, and each set contains the corresponding data elements to be able to render a complete feature. For Textures, a number of Texture LoD steps (***TexLoD***) can be set during i3s generation, where each added level will have half the resolution of the texture level of detail before. The number of Texture resources created is then equal ***n\*TexLoD***.

# 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:

[FIXME]

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.

## 3dSceneCacheInfo.js

The 3dSceneCacheInfo file is a JSON file that describes the content and encoding a single cache. A single cache can contain multiple layers, which all need to be of the same geometry content type.

The 3dSceneCacheInfo has the following structure:

[FIXME]

## 3dSceneLayerInfo.js

The 3dSceneLayerInfo file is a JSON file that describes the properties of a single layer in a cache, 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:

[FIXME]

## 3dNodeIndexDocument.js

The 3dNodeIndexDocument JSON file describes a single index node within a cache, 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 3: Logical format of the 3dNodeIndexDocument

### Class Node

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| id | long |  |
| level | int |  |
| href | string |  |
| version | UUID |  |
| srs | string |  |
| mbs | double[4] |  |
| precision | double |  |
| created | timestamp |  |
| expires | timestamp |  |

Table 2: Attributes of the Class Node within the NodeIndexDocument

### Class NodeReference

### Class Resource

### Class Transform

### Class Feature

### Blocked Nodes

Inner Nodes can also be “blocked”, i.e. grouped together, as a simple JSON array containing multiple Node Index Documents.

## Features.js

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

Features have the following structure:



## SharedResources.js

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

## Textures.bin

The Textures file is a binary resource that contains one or multiple images that are used as textures of features in the cache. 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 caches benefit from this type of storage.

## Bundled Indexed 3d Scenes

A cache 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)