3D Tiles Archive Format 1.1

The 3D Tiles Archive format 1.1 is based on the zip file format as defined by the ISO/IEC 21320-1:2015 specification, see https://www.iso.org/standard/60101.html, with the addition of the Zstandard compression method defined in “APPNOTE - .Zip File Format Specification” version 6.3.9, https://pkware.cachefly.net/webdocs/APPNOTE/APPNOTE-6.3.9.TXT. The Zstandard compression format itself is defined in IETF RFC 8474, https://tools.ietf.org/html/rfc8478.

Informative note: a valid 3D Tiles Archive Format 1.0 file is also valid according to v1.1 if it has a valid index file as described below.

A 3D Tiles archive must be a valid zip file according to the ISO/IEC 21320-1:2015 specification, with the addition of the Zstandard compression method as defined in “APPNOTE - .Zip File Format Specification” version 6.3.9, noting that it must use compression method ID 93 to indicate Zstandard was used.

For optimal read performance, files in the archive should be stored without compression, however, for a good trade of read performance and file size, use the Zstandard compression method. For best compatibility with legacy software, choose the standard deflate compression method, noting that this is the slowest of the three methods. Compression may be used on some or all files, except for the index file, which must be stored uncompressed.

All Local File Headers in the zip file must have compressed file size, uncompressed file size and crc32 set in the header. Note: this restricts the maximum file size for files inside the archive to 4 GB.

The archive must contain on the root level a file named “tileset.json”, that is a valid 3D Tiles tileset file, see https://github.com/CesiumGS/3d-tiles/tree/master/specification.

All references made in files inside the archive must be relative, and internal to the archive. This includes uri’s inside 3D Tiles tilesets as well as any references made in glTF documents in the archive.

The archive must use the `*.3tz` file extension.

The archive must contain a valid index file and it must be stored uncompressed, to improve load and read performance when the archive contains a very large number of files. The index file must be named `@3dtilesIndex1@` and must be the last file in the archive (read: it must be the last entry in the zip’s *Central Directory* [see Zip File Format Specification]).

The `@3dtilesIndex1@` file in the archive must not have an associated file comment in the Zip *Central Directory* (see Zip File Format Specification).

To generate the `@3dtilesIndex1@` index file follow these steps.

1. For each file in the archive (excluding the index file itself if present), find the offset to the corresponding Zip Local File Header (see Zip File Format Specification).
2. Normalize each file path by
    a. Replacing backslashes with forward slashes
    b. Dropping any leading forward slashes (e.g /tileset.json => tileset.json)
3. Compute the MD5 128-bit hash for the normalized file path.
4. Create an index entry with the hash (16 bytes) and the corresponding offset (8 bytes) for a total of 24 bytes.
5. Insert the index entry into the index, sorted in ascending order by the 128-bit MD5 hash interpreted as two little-endian 64-bit unsigned integers using the following pseudocode:

```
function md5_less_than(md5 hashA, md5 hashB) {
  hiA = hashA.readUInt64LE(0)
  hiB = hashB.readUInt64LE(0)
  if hiA == hiB then
     lowA = hashA.readUInt64LE(8)
     lowB = hashB.readUInt64LE(8)
      return lowA < lowB
  else
      return hiA < hiB
}
```

6. Write the index to the `@3dtilesIndex1@` binary file using
    a. Little endian order
    b. No padding
    c. No header

To find a specific file in the archive using the `@3dtilesIndex1@` index:

1. Normalize the given file path using the steps above.
2. Compute the MD5 hash for the normalized file path.
3. Search the index to find an entry with a matching MD5 hash. Since the index is sorted in ascending order based on the MD5 hash (see above), it's recommended to use a binary search algorithm to quickly find the given index entry.
4. If an index entry was found, read the Zip Local File Header (see Zip File Format Specification) at the entry offset.
5. Verify that the found Local File Header is the correct one, since there may have been a hash collision.
    a. Compare the Local File Header filename with the normalized file path.
    b. If that's a match, continue to step 6.

    c. Otherwise, find all index entries that have the same MD5 hash (they will be adjacent in the index table), then re-run steps 4 and 5 on them (excluding step 5c).
6. Return the associated decompressed file contents (see Zip File Format Specification).

Efficient reading of the 3D Tiles Archive Format

Readers of 3dtiles archive format are suggested to not use an off-the-shelf zip archive reader for opening the archive, but instead to first scan the archive a few hundred bytes from the end to find the last entry in the Central Directory. If the filename of the last entry matches the known 3dtiles archive index filename as defined by this specification, attempt to load the file contents by using the information found in the Central Directory entry. Then use the index to quickly find files inside the archive, as described above.

It’s still possible to read the 3D Tiles Archive as a standard zip archive, noting that it might be very slow in comparison.

An alternative approach is to extract the index from the archive once using any standard zip software, and then use it for reading the archive.