# Forest Service Lidar Indexing Project

## Overall goal

The overall goal of the indexing/cataloging portion of the project is to locate estimate the storage requirements for point cloud data that is not also stored as part of the USGS 3DEP data collection. Forest Service has data held on local NAS devices, corporate storage, and external hard drives. For this project, the focus is on corporate storage and local NAS devices. Hopefully, most of the early data has been moved from external hard drives onto one of these storage options. For data that has not, manual cataloging may be necessary to identify and catalog the data. The primary point data file types or LAS and LAZ. Additional formats are zLAS (ESRI’s proprietary compressed point cloud format), Entwine, and COPC (cloud optimized point clouds). COPC presents itself as LAZ but addes some additional information to the file headers to provide spatial index capabilities. Entwine is similar to COPC except data are shattered into lots of small files where each file represents a spatial extent and resolution. The scanning logic used for this indexing does not support Entwine format. However, the python code that can be used to create the same indexes does support Entwine format.

For Region 6 data, information in the folder name indicate is the data are part of a 3DEP acquisition. For other regions, indexes will need to be compared to the 3DEP WESM index to look for spatial overlap.

## Methods

The indexing process was developed using the Region 6 lidar data stored on the T: drive. These data served as the test case and most run times are based on these data. The indexing process consists of four basic steps:

* Recursively scan a parent folder to create a list of all subfolders,
* Look for one or more target file types in each folder,
* For folders with target files, scan all file headers for target file types and create an initial index,
* Postprocess the initial index to generate summary information and correct index problems (mainly CRS-related). Postprocessing varies depending on the folder structure and availability of CRS information.

The recursive scan can be accomplished using the DIR command (DIR /b /s>folderlist.txt). Our CIO contact ran a “special” script for us that took almost an hour. On VDI, the DIR command took less than three minutes. Running the DIR command locally with the T: drive connected via VPN took 6 hours. I also implemented the recursive scan as a first step in a C++ program that looks for target file types. This was used for the Region 6 data and the recursive directory listing took about 6 hours to run on my local computer connected to the T: drive via VPN.

The search for target files in each folder was done using a compiled C++ program that looked for the first file in a folder that matched the target types. I went with C++ because of the availability of the FindFirst() function that does not create a list of all files in a folder prior to checking for a file matching our target types. Running against local files was very fast. Running this locally with T: drive connected via VPN took 12 hours for the Region 6 data. Output from the search is a CSV file with the folder names and a flag indicating whether each target file type was found in the folder.

The scan of file headers was first attempted using python and/or R code running locally with the T: drive connected via VPN. This turned out to be very slow even when using a binary read of the file headers. From experience running the same scan of headers on local drives, the binary reads (implemented in python or R) were much faster compared to using PDAL (in python) or the lidR package (in R). Testing with the binary read function coded in C++ linked to R scripts using local files indicated a performance gain of 200-300 times compared to using a binary read in R or the lidR package. My hope was to run the same R/C++ code on VDI to take advantage of the faster connection to the T: drive. However, after installing Rtools from the software center on VDI (Rtools is needed to compile the C++ code that links to R to do the binary read), I found that permissions/policy would not allow the C++ compiler to run. In the end, I ran the R code from a local machine connected to the T: drive via VPN (ran for about 35 hours for R6 data). While this seems excessive and should be much faster when run on VDI, the index files that were produced allowed me to move forward.

### Post-processing of the indexes varies depending on the data source.

For some of the Region 6 data, there is a projection name embedded in the parent folder name. I used this to guess the CRS for ~60% of the index files. Our target data (non-3DEP) tend to be older so CRS information in point files, if included at all, is encoded as geoTIFF tags which do not provide a readily interpreted CRS “definition”. During the guessing process, I added a field (assignedCRS) to the dataframe associated with the boundingbox and boundary layers. The CRS WKT was added in this field when a CRS was “guessed”. I manually examined maps of all indexes with assigned CRS and only found seven projects/folders with problems. Five projects/folders had incorrect CRS (but don’t know how to figure out the correct CRS) and two had some point tiles with data in a different CRS than other tiles. This was manifested by a boundary polygon that spanned several states or extended into northern British Columbia.

After further thought, I came up with an approach that used PDAL to read a single point file from each index to sort out the CRS from the geoTIFF tags. PDAL has more robust code in this regard but was too slow in my initial tests to be used to read headers for every file. Reading the header for a single file in each folder took a few minutes, at most, so this was doable. The downside with this approach is that it requires a working installation of PDAL. This has not been a problem on my workstation but could be problematic on other FS-images computers. The R code creates the PDAL command line to parse header information and create a json file. The json file is then read in R to extract CRS information and the CRS is assigned to the index. Using this logic and the logic that loks for projection information in the folder name, I was able for get CRS information for 241 out of 278 index files. For folders where the CRS was “guessed” as described in the previous paragraph, I still read the header for a point file and assigned its CRS to the index.

## Results

### Region 6 data

Data for Region 6 is stored on the T: drive in this folder:

:\FS\Reference\RSImagery\ProcessedData\r06\R06\_DRM\_Deliverables\PointCloud

The initial folder scan was done on 3/10/2025. The second scan looking for target file types was completed 3/31/2025 and found 272 folders containing LAS or LAZ files (8 folders had both file types). I did notice some differences in the total number of folders found (~43,000±40) between scans done by CIO and the DIR command so I assume that R6 is actively managing these data and some changes were made. After the scan of file headers, 279 indexes were created (folders with both asset types get two indexes).

Table 1. Summary information for Region 6 point cloud data.

|  |  |
| --- | --- |
| **Index files created:** | 278 |
| **Total data size (Tb):** | 59.4 |
| **Index files identified as 3DEP:** | 52 |
| **3DEP data size (Tb):** | 29.9 |
| **Non-3DEP data size (Tb):** | 29.6 |

### Region 1 data

Data for Region 1 are (will be) stored here:

[\\AFSSXGTACNAS306\Enterprise\_Point\_Cloud\Lidar\R1](file:///\\AFSSXGTACNAS306\Enterprise_Point_Cloud\Lidar\R1)

There is no data on this device/folder (as of 4/8/2025).

### Region 3 data

Data for Region 3 are stored here:

[\\AFSSXGTACNAS306\Enterprise\_Point\_Cloud\Lidar\R3](file:///\\AFSSXGTACNAS306\Enterprise_Point_Cloud\Lidar\R3)

Table 2. Summary information for Region 3 point cloud data.

|  |  |
| --- | --- |
| **Index files created:** | 16 |
| **Total data size (Tb):** | 4.3 |
| **Index files identified as 3DEP:** | NA |
| **3DEP data size (Tb):** | NA |
| **Non-3DEP data size (Tb):** | NA |

As with the R6 data, many projects don’t have CRS information in WKT. The code that reads geoTIFF CRS information using PDAL worked with R3 data for all folders.

### Region 10 data

Data for Region 10 are stored on these NAS devices and on the T: drive

* Z = [\\199.131.101.238\TNFRef](file:///\\199.131.101.238\TNFRef) Z:\RSImagery\Geo\DEM\LiDAR (access denied)
* W = [\\199.131.101.238\CNFRef](file:///\\199.131.101.238\CNFRef) W:\RSImagery\Geo\DEM\LiDAR (access denied)
* T:\FS\Reference\RSImagery\ProcessedData\r10\_tnf\RSImagery\Geo\DEM\LiDAR
* T:\FS\Reference\RSImagery\ProcessedData\r10\_cnf\RSImagery\Geo\DEM\LIDAR (Note different capitalization of “LIDAR” compared to tongass folder)

Some of the R10 data are stored in ESRI’s zLAS format. I don’t have code to read this format so can’t create index files. Code (binary library) is available (https://github.com/Esri/esri-zlas-io-library) but the format is not documented (at least not publicly…not an open source specification). Given more time, I could try to incorporate the library into my C++ scanning program.

**NOTE: The following discussion refers to the zLIDAR format and not zLAS**…I found this resource: <https://jblindsay.github.io/zLidar_spec/section3.html> that has some useful information regarding the zLIDAR format. The zLIDAR file header is the same as the header for LAS/LAZ/COPC except the first four bytes (containing a “signature”) are “ZLDR” instead of “LASF” as in LAS/LAZ/COPC. Changes to the C++ code that reads LAS/LAZ to read zLIDAR are easy (and done), but I have never encountered this format.

As with R6 data, many of the projects do not have CRS information coded as WKT in the LAS files. I used the same code that calls PDAL to sort out the CRS information and assign it to the index files. For the Tongass data, this left 138 index files without CRS information. For the Chugach data, this logic worked to assign a CRS to all index files.

Table 3. Summary information for Region 10 Tongass National Forest point cloud data.

|  |  |
| --- | --- |
| **Index files created:** | 171 |
| **Total data size (Tb):** | 2.7 |
| **Index files identified as 3DEP:** | NA |
| **3DEP data size (Tb):** | NA |
| **Non-3DEP data size (Tb):** | NA |

Table 4. Summary information for Region 10 Chugach National Forest point cloud data.

|  |  |
| --- | --- |
| **Index files created:** | 13 |
| **Total data size (Tb):** | 1.3 |
| **Index files identified as 3DEP:** | NA |
| **3DEP data size (Tb):** | NA |
| **Non-3DEP data size (Tb):** | NA |

## Index format

A separate index is created for each folder and each file type (LAS, LAZ, COPC, or EPT). Names for index files are created using the folder name without the root folder. Slashes in the names (indicating subfolders) are replaced with “\_][\_” and the file type is appended to the name. For example, the index for LAS files in a folder named “2022\_LidarProject\PointData\LAZ” would be named “2022\_LidarProject\_][\_PointData\_][\_LAZ\_LAS.gpkg”. Index files are created by reading headers for all point data files in a folder and there is a separate index created for each type of point data. Index files are written in geopackage format with three layers:

* Boundingbox: overall bounding box for the tiles in the folder
* Boundary: polygon(s) that are the union of the tile bounding boxes. This will generally delineate areas covered by data but some portions of the tile bounding boxes may not actually contain point data.
* Assets: individual tile bounding boxes

Coordinate reference system (CRS) information is extracted from the point files and assigned to the index. If point files do not contain a CRS, a field (hasCRS) in the boundary and boundingbox layers is set to indicate that no CRS was found and the CRS WKT field (crs) is blank. If the CRS could be “guessed” based on the folder of file names, the corresponding CRS WKT is put in the assignedCRS field. For the Region 6 data, many folder names contain a string that identifies the CRS. However, the name is insufficient to fully guess the CRS. The name generally identifies the projection but not the horizontal datum.

The boundary and boundingbox layers share the same attributes (Table 2). The CRS of the index should be the same CRS described in the crs field or the assignedCRS field (when crs field is empty). For assets (Table 3), crs should match crs for the boundary and boundingbox layers. However, it is possible for some assets to have CRS information that is different from the overall CRS for the index. Only the first asset was examined when assigning the CRS for the index. If the first asset has no CRS information or CRS information that is different from other assets, there could be mixed CRSs for assets, or an incorrect CRS assigned to the index.

Entwine data is structured differently from simple point files. With entwine there is a json file in the parent directly that contains information describing the layout of individual tiles corresponding to octree cells in the data structure. For the scan as implemented in Python, the json file is read to get general information and the header for the root volume tile (ept-data/0-0-0-0.laz) is read to get point cloud header information. Headers for individual assets (nodes) are not read or included in the index. The R implementation with C++ header reading does not currently support entwine data.

Table 5. Field names, data types, and descriptions for boundary and boundingbox layers in point index geopackage files.

|  |  |  |
| --- | --- | --- |
| **Field name** | **Type** | **Description** |
| base | Character | Folder name. Does not include training slash. |
| pattern | Character | Search pattern (regex) for target files (“\.las”, “\.laz”, “\.copc”, “ept.json”) |
| assettype | Character | Asset type. Either “points” or “ept” |
| assetcount | Integer | Number of point tiles in folder for LAS/LAZ/COPC. For ept, should be 1. |
| assetsize | Long integer | Total size in bytes for all assets in folder. |
| totalpointcound | Long integer | Total number of points for all assets in folder. |
| hasCRS | Logical | Logical value indicating that CRS was found in assets. TRUE means CRS information was found and should be in the crs field. FALSE means that CRS was not found in assets. |
| minx | Double | Minimum X value for bounding box containing all assets. |
| miny | Double | Minimum Y value for bounding box containing all assets. |
| maxx | Double | Maximum X value for bounding box containing all assets. |
| maxy | Double | Maximum Y value for bounding box containing all assets. |
| crs | Character | WKT string representing CRS information found in assets. May be a simple projection or compound CRS. When this string is NOT empty, hasCRS should be TRUE. Conversely, when hasCRS is TRUE, this string should contain information. |
| assignedCRS | Character | CRS WKT assigned to index. Blank if assets contained CRS or no CRS was assigned. The assignment could have been done using information in the folder name (as with some R6 data), WKT in the point files read with the C++ function or information from the geoTIFF tags in point files read using PDAL. |

Table 6. Field names, data types, and descriptions for assets layer in point index geopackage files.

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| filespec | Character | Full path to the asset file. |
| filesize | Long integer | Size, in bytes, of the asset. |
| pointcount | Long integer | Number of points in asset. |
| compressed | Logical | Logical value indicating file is compressed. |
| copc | Logical | Logical value indicating file is COPC format. |
| creation\_day | Integer | Creation day of year[[1]](#footnote-1). |
| creation\_year | Integer | Creation year1. |
| point\_record\_format | Integer | Point record format1. |
| point\_record\_length | Integer | Point record length1. |
| major\_version | Integer | LAS format major version1. |
| minor\_version | Integer | LAS format minor version1. |
| minx | Double | Minimum X value for bounding box containing all assets. |
| miny | Double | Minimum Y value for bounding box containing all assets. |
| minz | Double | Minimum Z value for bounding box containing all assets. |
| maxx | Double | Maximum X value for bounding box containing all assets. |
| maxy | Double | Maximum Y value for bounding box containing all assets. |
| maxz | Double | Maximum Z value for bounding box containing all assets. |
| crs | character | WKT string representing CRS information found in assets. May be a simple projection or compound CRS. |

1. Refer to LAS standard documentation for details. [↑](#footnote-ref-1)