Reading REAL `BSCAN` Raw Data Files

A Technical Note for the Raman-shifted Eye-safe Aerosol Lidar (REAL)

Shane D. Mayor
Department of Earth and Environmental Science
California State University, Chico

Draft compiled July 1, 2026

Abstract

This document explains how to read original raw data files from the Raman-shifted Eye-safe Aerosol Lidar (REAL). These files use a custom binary format called `BSCAN`. Each file normally contains one scan, and each record in the file corresponds to one laser pulse. A record consists of a fixed-length 30-word header followed by two profile arrays: one for the parallel-polarization backscatter signal and one for the perpendicular-polarization backscatter signal. All values are stored as big-endian single-precision floating-point numbers. Example reader code is provided in IDL, MATLAB, and Python.

Contents

1 Why the format is called BSCAN

The name `bscan` comes from older radar-display terminology, at least as I understand the history of the format. Before modern digital electronics, radar systems commonly used cathode-ray tube (CRT) displays that were built for particular display geometries. Radar operators and engineers used terms such as PPI, RHI, A-scope, and B-scope to describe different ways of displaying the returned signal.

An A-scope was essentially one-dimensional: signal strength as a function of range for one transmitted pulse or one pointing direction. A B-scope was more image-like. It displayed range along one dimension and another scan coordinate, such as time, pointing angle, or record number, along the other. In other words, a B-scope was a rectangular view of many range profiles placed side by side.

That is also a useful way to picture a REAL `bscan` file. The file contains one complete scan. Each laser pulse contributes one fixed-length record. Each record contains a header followed by two range-resolved backscatter profiles. If those records are stacked in order, the result is a rectangular array: range in one direction and record number, scan angle, or time in the other. The file is binary, but conceptually it is very much like a B-scope image waiting to be read.

2 A short history of the REAL BSCAN format

I learned to store lidar data in this style while working in the Lidar Applications Group at NASA Langley Research Center from 1990 to 1993. During that time, I worked with airborne atmospheric lidar systems, usually differential absorption lidars (DIAL), that pointed either upward or downward from an aircraft. The resulting data geometry was like a vertical atmospheric cross section, sometimes called a curtain file. In that sense, the data really did resemble a B-scope or B-scan format: range along one dimension and aircraft position, time, or record number along the other.

The header was equally important for airborne lidar work because each laser shot had to be associated with aircraft position and attitude information, including latitude, longitude, altitude, pitch, roll, and other navigation or platform variables. This is why those fields still appear in the REAL `bscan` header today, even though the REAL is normally operated as a stationary ground-based system.

In the Lidar Applications Group, we commonly worked with two types of BSCAN files. The *raw* files contained the original, untouched backscatter waveforms. The *reduced* files contained derived products such as range-corrected scattering, ozone DIAL retrievals, or water-vapor DIAL retrievals.

I began saving lidar data in this same style when I went to NCAR in the mid-1990s. I believe I implemented this approach on the NCAR Airborne Infrared Lidar System (NAILS). Later, when I returned to NCAR in 2001 and began development of the REAL, it seemed natural to save lidar data in the same way. And I think my colleagues in the Boulder NOAA lidar group started saving data in the same way and we shared IDL code to read and display BSCAN files.

In the beginning, I remember discussions about storage efficiency. Why save the numbers as floating-point values when quantities such as date, time, and digitizer counts were integers? The answer was simplicity and robustness. It was easier to sacrifice some disk space and save everything as floating point than to keep track of which words were integers and which were floating-point values. There were always going to be some floating-point numbers in each record, and we did not want every reader program to have to remember the mixed data types or slow down to parse them separately. So we saved everything as floating point.

3 Purpose

The purpose of this note is to make the original REAL raw data format understandable to students, collaborators, and future users of the dataset. The `BSCAN` format is simple once its structure is known, but it can be confusing because it is a binary file format rather than a text, NetCDF, HDF, or MATLAB file.

The most important facts are:

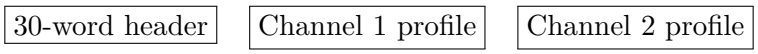
- A `BSCAN` file is a binary file made entirely of 32-bit floating-point numbers.
- The byte order is big-endian.
- In IDL terminology, the values are `real*4`.
- In MATLAB terminology, the values should be read as `'single'` with big-endian byte ordering.
- In Python/NumPy terminology, the values can be read as `'>f4'`.
- Each file usually contains one scan.
- Each record usually corresponds to one laser pulse.
- Each record contains a 30-word header followed by two backscatter profiles.

For M2HATS, a typical scan lasted about 15 seconds and contained about 150 records. The exact number of records should always be computed from the file size and the record length rather than assumed.

4 Overall file structure

A `BSCAN` file is a sequence of fixed-length records:

Each record has the same internal structure:



For REAL, the two profile arrays are the two polarization channels:

- **Channel 1**: parallel-polarization backscatter signal.
- **Channel 2**: perpendicular-polarization backscatter signal.

The profile values are digitizer counts. The recorded signal contains background plus the lidar return signal. The REAL system used a 14-bit, 100 mega-sample per second analog-to-digital converter. For M2HATS, each profile element corresponds to a 1.5 m range increment.

5 Binary representation

All words in the file are single-precision floating-point values:

Quantity	Value
Data type	32-bit floating point
IDL type	`real*4` / `fltarr`
MATLAB type	`single`
NumPy type	`float32`
Byte order	Big-endian
Bytes per word	4

The byte order is important. If the file is read using the wrong byte order, the header values will be nonsense. A quick sanity check is to inspect familiar header fields such as hour, minute, second, month, day, year, azimuth, elevation, range of first profile element, and profile length.

6 Record length

The first 30 words of the file are the header for the first record. Header word 28 in zero-based indexing, or word 29 in MATLAB one-based indexing, gives the number of words in each profile. Let

$$L_p = \text{number of words in one profile.} \tag{1}$$

Then each record contains

$$L_r = 30 + 2L_p \tag{2}$$

floating-point words. Since each word is 4 bytes, the record length in bytes is

$$4L_r = 4(30 + 2L_p). \tag{3}$$

The number of records in a file is therefore

$$N = \frac{\text{file size in bytes}}{4L_r}. \tag{4}$$

In code, it is often written as:

```
prof_len = first_header(29);        % MATLAB one-based indexing
rec_len = fix(2*prof_len + 30);
recs_in_file = floor(filesize/(rec_len*4));
```

or, in Python:

```
prof_len = int(first_header[28])    # Python zero-based indexing
rec_len = int(2*prof_len + 30)
recs_in_file = int(filesize/(rec_len*4))
```

7 Range coordinate

The header contains the range of the first profile element and the range spacing between profile elements. In zero-based indexing:

- `header[20]` is the range of the first element in meters.
- `header[21]` is the range resolution in meters.

In MATLAB one-based indexing, these are:

- `A(21)`: range of the first element in meters.
- `A(22)`: range resolution in meters.

The range array is

$$r_k = r_0 + k\Delta r, \qquad k = 0, 1, 2, \ldots, L_p - 1, \tag{5}$$

where r_0 is the range of the first element and Δr is the range resolution.
For example, in MATLAB:

```
r0 = A(21);                    % range of first element, m
dr = A(22);                    % range spacing, m
r = r0 + (0:prof_len-1)*dr;    % range array, m
```

For M2HATS, typical values are:

Header value	Meaning
`header[20] = -567`	first profile element is at -567 m
`header[21] = 1.5`	1.5 m range spacing
`header[28] = 4200`	4200 words per profile

Negative ranges occur before the laser pulse reaches the atmosphere and are useful for estimating the electronic/background signal.

8 Header words

Table 1 gives the general 30-word header definition using zero-based indexing. This is the natural indexing convention for IDL and Python. MATLAB users should add 1 to each index.

Table 1: General `BSCAN` 30-word header definition using zero-based indexing. MATLAB users should add 1 to each index.

Index	Meaning
0	Hour, 0–23
1	Minute, 0–59
2	Second, 0.00–59.999
3	GPS altitude, meters above sea level
4	Alternative altitude, such as pressure altitude

Index	Meaning
5	Alternative altitude, such as radar altimeter
6	Lidar beam pointing azimuth, degrees
7	Lidar beam pointing elevation, degrees
8	Platform roll, degrees with respect to earth horizon
9	Platform pitch, degrees with respect to earth horizon
10	Platform yaw or sideslip with respect to platform heading, degrees
11	Month, 1–12
12	Day, 1–31
13	Year, preferably full year such as 2023 rather than 23
14	Pulse repetition frequency of lidar, Hz
15	Number of laser pulses intentionally not used in this record
16	Data type identifier
17	Number of hours to adjust header time to UTC
18	Platform latitude, degrees north
19	Platform longitude, degrees east of Greenwich
20	Range from lidar to first word of recorded profile data, meters
21	Range resolution of lidar data, meters per word
22	Platform identifier
23	Number of words in header; should be 30
24	No-data flag
25	Empty or unused
26	Number of laser shots used to derive this record
27	Number of laser shots intended to use to derive this record
28	Number of words in one profile array
29	Transect number, used for airborne lidar research

9 M2HATS-specific header notes

For M2HATS, several header fields were not filled correctly by the data acquisition system or have uncertain meanings. Table 2 records the current working interpretation. The indices in this table are zero-based.

Table 2: M2HATS-specific notes on the 30-word `BSCAN` header, using zero-based indexing.

Index	M2HATS note
0	Correct: hour
1	Correct: minute
2	Correct: second
3	All zeros
4	Unclear; varies randomly around 0.375–0.378
5	Unclear; oscillates roughly between -0.120 and -0.095
6	Correct: azimuth, often 60–120 degrees during sector scans
7	Correct: elevation, close to zero for near-horizontal scans
8	All zeros
9	All zeros

Index	M2HATS note
10	All zeros
11	Correct: month, 7, 8, or 9 for M2HATS
12	Correct: day of month
13	Correct: year, 2023 for M2HATS
14	All zeros
15	Constant at 967; meaning uncertain
16	Constant at 220; meaning uncertain
17	All zeros
18	Incorrect: contains Chico latitude, not Tonopah/M2HATS latitude
19	Incorrect: contains Chico longitude, not Tonopah/M2HATS longitude
20	Correct: range of first profile element, typically -567 m
21	Correct: range resolution, typically 1.5 m
22	Approximately 9.0001001; meaning uncertain
23	Correct: 30 words per header
24	All zeros
25	All 2; meaning uncertain
26	Unclear; varies roughly between 380 and 440, possibly transmit pulse energy
27	Unclear; varies roughly between 1160 and 1220, possibly pump pulse energy
28	Correct: 4200 words per profile for M2HATS
29	Constant at 1

10 Sanity-check values

For the example file

`REAL.20230818_211736.bscan`

known header values from the first record are:

Zero-based header indices	Values	Meaning
`header[11:13]`	8, 18, 2023	date: 18 August 2023
`header[0:2]`	21, 17, 38.8800	time: 21:17:38.880 UTC

Known values from the last record are:

Zero-based header indices	Values	Meaning
`header[11:13]`	8, 18, 2023	date: 18 August 2023
`header[0:2]`	21, 17, 53.8220	time: 21:17:53.822 UTC

These values are useful for testing whether a reader is handling byte order and record length correctly.

11 IDL example

The following IDL example reads the first 30 words, determines the full record length, computes the number of records, constructs a range array, and then loops through all records.

```
pro read_bscan
path = '/Volumes/data/'
fn = 'REAL.20230818_211736.bscan'

openr,lun1,path+fn,/get_lun
record = assoc(lun1,fltarr(30))
first_header = record(0)
stats = fstat(lun1)
close,lun1
free_lun,lun1

byteorder,first_header,/Lswap

prof_len = first_header(28)
rec_len = fix(2*prof_len + 30)
recs_in_file = long(stats.size/(rec_len*4l))

zero_range = fix(-first_header(20)/first_header(21))
range_index = findgen(prof_len)-zero_range
range = range_index*first_header(21)

openr,lun1,path+fn,/get_lun
record = assoc(lun1,fltarr(rec_len))

for i = 0, recs_in_file-1 do begin
    z = record(i)
    byteorder,z,/Lswap

    header = z(0:29)
    channel1 = z(30:30+prof_len-1)
    channel2 = z(30+prof_len:30+prof_len*2-1)
endfor

close,lun1
free_lun,lun1
stop
end
```

12 MATLAB example

This MATLAB example is intentionally simple. It extracts the core reading logic from a larger analysis script. The important point is the use of `fopen(fn,'r','b')`, which tells MATLAB to read the file as big-endian.

```
% Simple MATLAB reader for REAL BSCAN files.

fn = 'REAL.20230818_211736.bscan';
```

```
% Read the first 30 words to determine the file structure.
fileID = fopen(fn,'r','b');          % 'b' means big-endian
first_header = fread(fileID,30,'single');
fseek(fileID,0,'eof');
filesize = ftell(fileID);
fclose(fileID);

% MATLAB uses one-based indexing.
prof_len = first_header(29);
rec_len = fix(2*prof_len + 30);
recs_in_file = floor(filesize/(rec_len*4));

% Construct the range array.
r0 = first_header(21);
dr = first_header(22);
r = r0 + (0:prof_len-1)*dr;

% Allocate arrays if desired.
headers = nan(recs_in_file,30);

% Read each complete record.
fileID = fopen(fn,'r','b');

for i = 1:recs_in_file
    record = fread(fileID,rec_len,'single');

    if numel(record) < rec_len
        break
    end

    header = record(1:30);
    channel1 = record(31:30+prof_len);
    channel2 = record(31+prof_len:30+2*prof_len);

    headers(i,:) = header;

    % Example header values:
    year = header(14);
    month = header(12);
    day = header(13);
    hour = header(1);
    minute = header(2);
    second = header(3);
    azimuth = header(7);
    elevation = header(8);
end

fclose(fileID);
```

To seek directly to record `irec`, use:

```
fid = fopen(fn,'r','b');
offset = (irec-1)*rec_len*4;
fseek(fid,offset,'bof');
record = fread(fid,rec_len,'single');
```

```
fclose(fid);

header = record(1:30);
channel1 = record(31:30+prof_len);
channel2 = record(31+prof_len:30+2*prof_len);
```

13 Python example

The cleanest way to read a `BSCAN` file in Python is to use NumPy's big-endian single-precision data type, `'>f4'`. This avoids manual byte swapping.

```
from pathlib import Path
import numpy as np


def read_bscan(filename):
    """Read a REAL BSCAN file.

    Parameters
    ----------
    filename : str or pathlib.Path
        Path to a .bscan file.

    Returns
    -------
    headers : ndarray, shape (n_records, 30)
        Header values for each record.
    channel1 : ndarray, shape (n_records, prof_len)
        Parallel-polarization backscatter profiles.
    channel2 : ndarray, shape (n_records, prof_len)
        Perpendicular-polarization backscatter profiles.
    ranges : ndarray, shape (prof_len,)
        Range coordinate in meters.
    """

    filename = Path(filename)
    filesize = filename.stat().st_size

    # Read the first 30 big-endian float32 values.
    first_header = np.fromfile(filename, dtype=">f4", count=30)

    prof_len = int(first_header[28])
    rec_len = int(2*prof_len + 30)
    recs_in_file = int(filesize // (rec_len*4))

    # Read the entire file as big-endian float32 and reshape into records.
    data = np.fromfile(filename, dtype=">f4", count=recs_in_file*rec_len)
    records = data.reshape((recs_in_file, rec_len))

    headers = records[:, 0:30]
    channel1 = records[:, 30:30+prof_len]
    channel2 = records[:, 30+prof_len:30+2*prof_len]
```

```
    r0 = first_header[20]
    dr = first_header[21]
    ranges = r0 + np.arange(prof_len)*dr

    return headers, channel1, channel2, ranges


if __name__ == "__main__":
    headers, ch1, ch2, ranges = read_bscan("REAL.20230818_211736.bscan")

    print("records:", headers.shape[0])
    print("profile length:", ch1.shape[1])
    print("first date:", headers[0, 11:14])
    print("first time:", headers[0, 0:3])
    print("last date:", headers[-1, 11:14])
    print("last time:", headers[-1, 0:3])
```

14 From raw counts to high-pass median-filtered backscatter

Reading the file is only the first step. Many REAL analyses use a processed signal rather than the raw digitizer counts. A common processing path is:

1. Estimate the background from the negative-range portion of each profile.
2. Subtract the background from each channel.
3. Add the two polarization channels, if total backscatter signal is desired.
4. Multiply by range squared to compensate for the inverse-square spreading of the lidar return.
5. Convert to decibels.
6. Apply a rolling high-pass median filter in the radial dimension.

In MATLAB-like notation:

```
bg_idx = find(r < 0);

B1 = mean(C1(bg_idx));
B2 = mean(C2(bg_idx));

C1_bs = C1 - B1;
C2_bs = C2 - B2;

p = C1_bs + C2_bs;
beta = p .* (r.^2);

offset = 500;
beta_dB = 10*log10(max(beta + offset, 1e-5));

win = 333;   % 333 samples * 1.5 m/sample is about 500 m
med = movmedian(beta_dB, win, 'Endpoints','shrink');
hpmf = beta_dB - med;
```

This processing is not part of the BSCAN format itself. It is included here to show how the raw profile arrays are commonly converted into images or radial signals used in REAL aerosol-structure analyses.

15 Common pitfalls

15.1 Using the wrong byte order

The file must be read as big-endian. In MATLAB, use:

```
fopen(fn,'r','b')
```

In Python, use:

```
np.fromfile(filename, dtype=">f4")
```

If the byte order is wrong, the date, time, azimuth, elevation, range spacing, and profile length will not make physical sense.

15.2 Confusing zero-based and one-based indexing

IDL and Python use zero-based indexing. MATLAB uses one-based indexing. Therefore:

Quantity	IDL/Python index	MATLAB index
Hour	0	1
Minute	1	2
Second	2	3
Azimuth	6	7
Elevation	7	8
Month	11	12
Day	12	13
Year	13	14
First range	20	21
Range spacing	21	22
Profile length	28	29

15.3 Assuming the number of records

Do not assume that every scan contains exactly 150 records. Compute the number of records from the file size and record length.

15.4 Forgetting that each record has two profiles

The REAL BSCAN records contain two profile arrays after the header. The record length is therefore `30 + 2*prof_len`, not `30 + prof_len`.

15.5 Treating M2HATS latitude and longitude as valid

For M2HATS files, header words 18 and 19 contain Chico latitude and longitude rather than the Tonopah/M2HATS location. Do not use those fields for M2HATS geolocation.

16 Minimal checklist for a new reader

A new `BSCAN` reader should be tested by verifying that it can:

1. Read the first 30 words as big-endian single-precision floats.
2. Recover a sensible profile length from header word 28, or MATLAB word 29.
3. Compute a record length of `30 + 2*prof_len`.
4. Compute the number of records from the file size.
5. Reshape or iterate through the file record by record.
6. Extract each 30-word header.
7. Extract the two profile arrays.
8. Recover plausible date, time, azimuth, and elevation values.
9. For the example file, recover the first and last record date/time values listed in this document.

Acknowledgments

The REAL `BSCAN` format descends from earlier experimental atmospheric lidar systems and was later adopted for use by REAL. The examples in this note are based on working IDL, MATLAB, and Python reader code used at California State University, Chico.