sLASh – a simple LAS reader library in a single header file

Thomas Knudsen, PhD

Danish Geodata Agency

European Lidar Map Forum, 2013

Danish Ministry of the Environment
Danish Geodata Agency

Outline

1. **What?**
   - What is it
   - What does it look like?
2. **Why?**
   - Do we need yet another LAS library?
   - But why a header library?
3. **How? – Design and Usage**
   - Setting and influences
   - Overall Design
4. **(Un)conclusion**
5. **DVD bonus material – digitally remastered director's cut**

Outline

1 **What?**
- What is it
- What does it look like?

2 **Why?**
- Do we need yet another LAS library?
- But why a header library?

3 **How? – Design and Usage**
- Setting and influences
- Overall Design

4 **(Un)conclusion**

5 **DVD bonus material – digitally remastered director's cut**

What is it

- sLASh – `slash.h` – reads **LAS** files
- Header library, written in plain C
- Permissive license (ISC/OpenBSD)
- Very compact

What is it

- sLASh – `slash.h` – reads **LAS** files
- Header library, written in plain C
- Permissive license (ISC/OpenBSD)
- Very compact

```
wc -l slash.h
   1346  slash.h

awk -f lines_of_code.awk  slash.h|wc -l
    577
```

What is it

- sLASh – **slash.h** – reads **LAS** files
- Header library, written in plain C
- Permissive license (ISC/OpenBSD)
- Very compact

```
wc -l slash.h
   1346  slash.h

awk -f lines_of_code.awk  slash.h|wc -l
    577
```

- Data type definitions and self tests ≈200 lines
- Core library functionality **<400 lines**
- Binary **< 7 kB** (gcc, Win64).

What does it look like?

```c
#include "slash.h"

int main (int argc, char **argv) {
    LAS *h;
    double x, y, z;

    h = las_open (argv[1], "rb");
    while  (las_read (h))  {
        x = las_x (h);
        y = las_y (h);
        z = las_z (h);
        printf ("%f %f %f\n", x, y, z);
    }

    las_close (h);
    return 0;
}
```

Outline

1 What?
- What is it
- What does it look like?

2 **Why?**
- Do we need yet another LAS library?
- But why a header library?

3 How? – Design and Usage
- Setting and influences
- Overall Design

4 (Un)conclusion

5 DVD bonus material – digitally remastered director's cut

A relevant question...

LASlib

libLAS

But... **Do we really need another LAS library?**

Do we really need another LAS library?

- **We** may not need it
- but **I did** need it

Long story short...

Do we really need another LAS library?

- **We** may not need it
- but **I did** need it

Long story short...

- Needed to patch some slightly erroneous LAS 1.0 files
- Just a small extra effort to turn the code into a library.

But why a *header* library?

Recently, we did some experiments packaging small re-usable software components as *header-only libraries*

- Including

  ASTA
  : Accumulation of STAtistics

  stack
  : Template library in plain C

  comquat
  : COMplex numbers and QUATernions

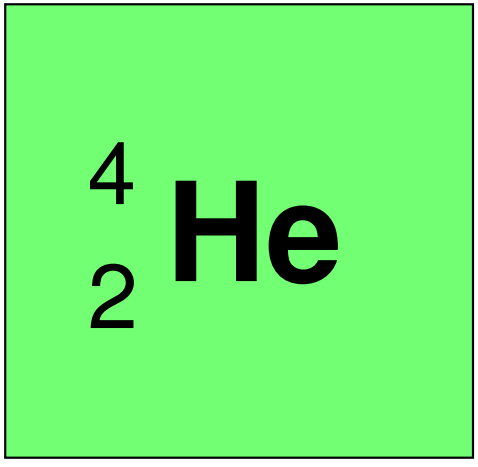
- Obvious idea to package the LAS reader in the style of ASTA, stack and comquat.

Helios

The entire package of libraries was named after 3 elements

Helios

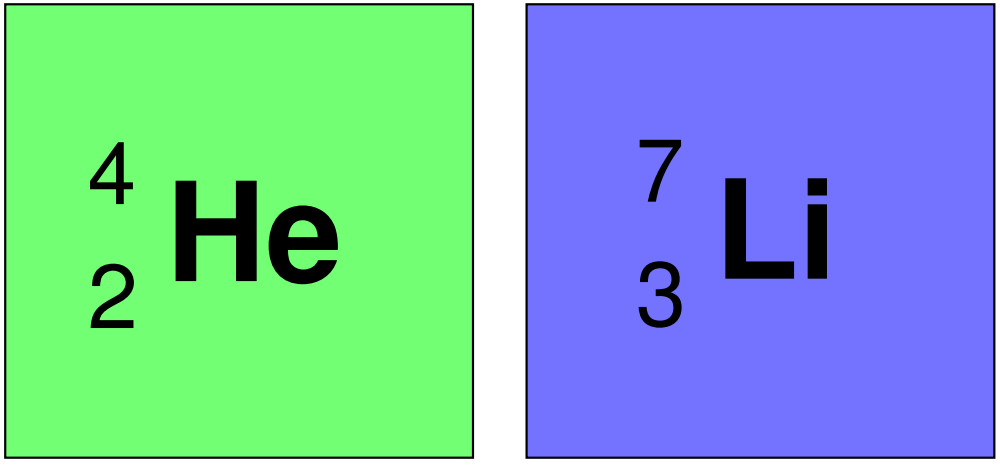
The entire package of libraries was named after 3 elements



Header

Helios

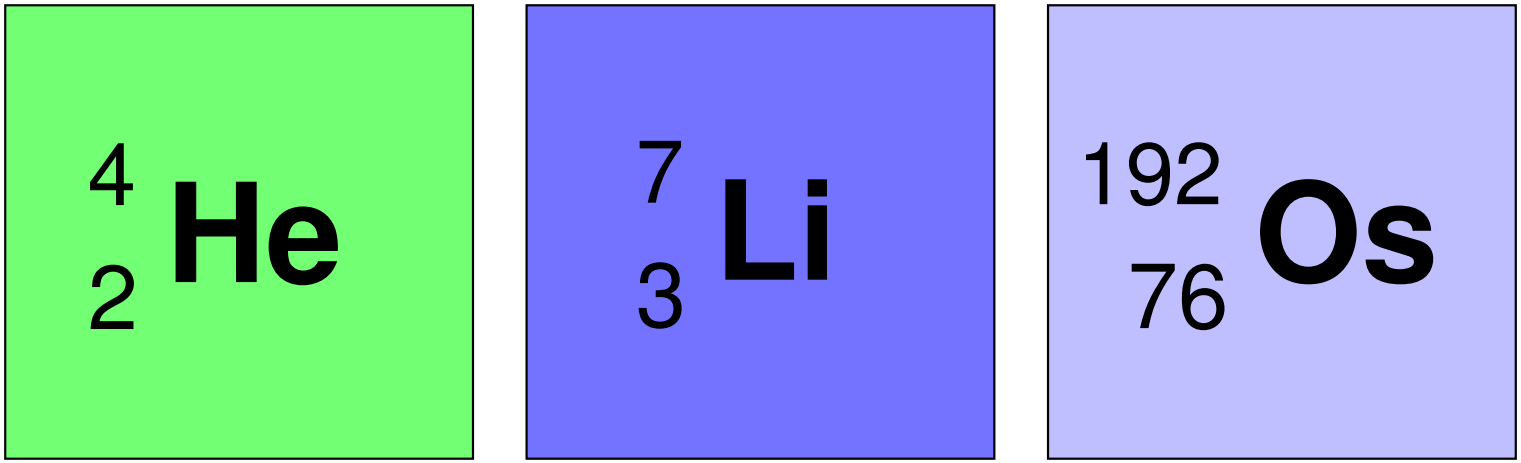
The entire package of libraries was named after 3 elements



Header Libraries

Helios

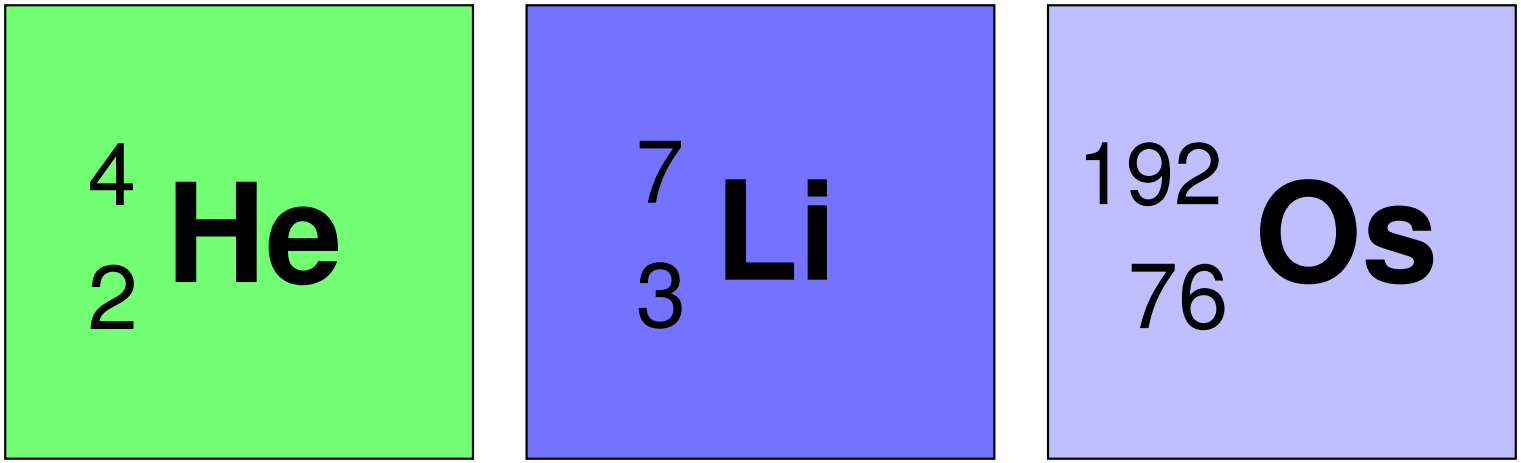
The entire package of libraries was named after 3 elements



Header Libraries on Steroids

Helios

The entire package of libraries was named after 3 elements



Header Libraries on Steroids

It may be *perceived* as *dense and brittle*, but it's *lightweight* and it *keeps you sane* **:-)**

Outline

1 What?
- What is it
- What does it look like?

2 Why?
- Do we need yet another LAS library?
- But why a header library?

3 How? – Design and Usage
- Setting and influences
- Overall Design

4 (Un)conclusion

5 DVD bonus material – digitally remastered director's cut

Our typical software tool setup

LAS processing

LAStools, libLAS, OPALS, TerraSolid

Our typical software tool setup

LAS processing

LAStools, libLAS, OPALS, TerraSolid

Core functionality/library implementation

C (with Python bindings)

Our typical software tool setup

LAS processing

LAStools, libLAS, OPALS, TerraSolid

Core functionality/library implementation

C (with Python bindings)

Graphical User Interfaces

Python/PyQt

Our typical software tool setup

LAS processing
: LAStools, libLAS, OPALS, TerraSolid

Core functionality/library implementation
: C (with Python bindings)

Graphical User Interfaces
: Python/PyQt

Production automatization
: Python, bash, Windows **cmd.exe**

The LAS format

The sLASh design is necessarily shaped by the main structures defined in the LAS format specification:

The LAS format

The sLASh design is necessarily shaped by the main structures defined in the LAS format specification:

Public Header

Version, Spatial extent, Scaling, Record type, etc.

The LAS format

The sLASh design is necessarily shaped by the main structures defined in the LAS format specification:

Public Header

Version, Spatial extent, Scaling, Record type, etc.

Variable Length Records

Georeferencing, generic user defined metadata

The LAS format

The sLASh design is necessarily shaped by the main structures defined in the LAS format specification:

Public Header
: Version, Spatial extent, Scaling, Record type, etc.

Variable Length Records
: Georeferencing, generic user defined metadata

Fixed Length Records
: The actual LiDAR point data

The LAS format

The sLASh design is necessarily shaped by the main structures defined in the LAS format specification:

Public Header
: Version, Spatial extent, Scaling, Record type, etc.

Variable Length Records
: Georeferencing, generic user defined metadata

Fixed Length Records
: The actual LiDAR point data

Extended Variable Length Records
: e.g. Waveform data

The LAS format

The sLASh design is necessarily shaped by the main structures defined in the LAS format specification:

Public Header
: Version, Spatial extent, Scaling, Record type, etc.

Variable Length Records
: Georeferencing, generic user defined metadata

Fixed Length Records
: The actual LiDAR point data

Extended Variable Length Records
: e.g. Waveform data

Five versions (LAS 1.0–1.4), 11 fixed record formats (0–10)

$\Longrightarrow$ Probably not as simple as it could be. . .

Design principles

Less is more

You cannot do everything in 400 lines

Worse is better

Go for simple implementations. Don’t get smart unless necessary: smart means brittle

Design principles

Less is more

You cannot do everything in 400 lines

Worse is better

Go for simple implementations. Don't get smart unless necessary: smart means brittle

Avoid the 2nd system syndrome

Plan ahead but *keep it simple, stupid* (KISS)

Avoid creeping featurism

"you're not gonna need it".

Overall Design

Data types

LAS, NRGB, Waveform Descriptor, LAS record, VLR

Overall Design

Data types

LAS, NRGB, Waveform Descriptor, LAS record, VLR

File access API

las_open, las_seek, las_read, las_close

Overall Design

Data types

LAS, NRGB, Waveform Descriptor, LAS record, VLR

File access API

las_open, las_seek, las_read, las_close

Record access API

las_x, las_y, las_z, las_intensity, las_flag_overlap. . .

Overall Design

Data types

LAS, NRGB, Waveform Descriptor, LAS record, VLR

File access API

las_open, las_seek, las_read, las_close

Record access API

las_x, las_y, las_z, las_intensity, las_flag_overlap. . .

Printing and formatting API

las_header_display, las_record_display, las_vlr_display

Overall Design

Data types
: LAS, NRGB, Waveform Descriptor, LAS record, VLR

File access API
: las_open, las_seek, las_read, las_close

Record access API
: las_x, las_y, las_z, las_intensity, las_flag_overlap. . .

Printing and formatting API
: las_header_display, las_record_display, las_vlr_display

Plumbing
: Record structure LUTs, big endian/little endian, WIN32 large file support.

Back to page 1...

```c
#include "slash.h"

int main (int argc, char **argv) {
    LAS *h;
    double x, y, z;

    h = las_open (argv[1], "rb");
    while  (las_read (h))  {
        x = las_x (h);
        y = las_y (h);
        z = las_z (h);
        printf ("%f %f %f\n", x, y, z);
    }

    las_close (h);
    return 0;
}
```

Features

sLASh will happily. . .

Features

sLASh will happily...

- Access LAS files

Features

sLASh will happily. . .

- Access LAS files
- Read variable length records

Features

sLASh will happily. . .

- Access LAS files
- Read variable length records
- Read fixed length records

Features

sLASh will happily. . .

- Access LAS files
- Read variable length records
- Read fixed length records
- Access/interpret fields and flags of the fixed length records.

Un-features

sLASh will only reluctantly (or not at all)...

Un-features

sLASh will only reluctantly (or not at all)...

- Interpret variable length records

Un-features

sLASh will only reluctantly (or not at all)...

- Interpret variable length records
- Read compressed **LAZ** files

Un-features

sLASh will only reluctantly (or not at all)...

- Interpret variable length records
- Read compressed **LAZ** files
- Write LAS files

Un-features

sLASh will only reluctantly (or not at all)...

- Interpret variable length records
- Read compressed **LAZ** files
- Write LAS files
- Make coffee
- Provide kitchen sink access
- Read mail
- Emit tweets.

Where to use?

The Use Cases. . .

Where to use?

The Use Cases...

- Probably mostly for lovers of the C language

Where to use?

The Use Cases...

- Probably mostly for lovers of the C language
- Quick-and-dirty hacks

Where to use?

The Use Cases...

- Probably mostly for lovers of the C language
- Quick-and-dirty hacks
- When stream mode reading is sufficient (e.g. building inventories, collecting statistics...)

Where to use?

The Use Cases. . .

- Probably mostly for lovers of the C language
- Quick-and-dirty hacks
- When stream mode reading is sufficient (e.g. building inventories, collecting statistics. . .)
- Memory constrained (e.g. embedded) systems

Where to use? (2)

The Useless Cases...

Where to use? (2)

The Useless Cases...

- When writing LAS files are important

Where to use? (2)

The Useless Cases...

- When writing LAS files are important
- When you need on-the-fly reprojection

Where to use? (2)

The Useless Cases...

- When writing LAS files are important
- When you need on-the-fly reprojection
- When you need spatial indexing (.lax files)
- When you need LAS compression (.laz files)

Where to use? (3)

In brief:

Where to use? (3)

In brief:

- Use LAStools/LASlib/libLAS for the heavy lifting

Where to use? (3)

In brief:

- Use LAStools/LASlib/libLAS for the heavy lifting
- only consider sLASh when filling in where you need specialized functionality

Where to use? (3)

In brief:

- Use LAStools/LASlib/libLAS for the heavy lifting
- only consider sLASh when filling in where you need specialized functionality
- . . . and have fun!

Outline

1 What?
- What is it
- What does it look like?

2 Why?
- Do we need yet another LAS library?
- But why a header library?

3 How? – Design and Usage
- Setting and influences
- Overall Design

4 (Un)conclusion

5 DVD bonus material – digitally remastered director's cut

Summary

- Try sLASh
- It's compact
- It's fast
- It's fun
- It's available from http://bitbucket.org/busstop/helios
- a.k.a. **http://goo.gl/zsi8qz**
- Get it while it's hot!

Outline

1 What?
- What is it
- What does it look like?

2 Why?
- Do we need yet another LAS library?
- But why a header library?

3 How? – Design and Usage
- Setting and influences
- Overall Design

4 (Un)conclusion

5 DVD bonus material – digitally remastered director's cut

lasinfo in 8 lines

```
#include "slash.h"
int main (int argc, char **argv) {
    LAS *h;
    h = las_open (argv[1], "rb");
    las_header_display (stdout, h);
    las_close (h);
    return 0;
}
```

lasinfo in 8 lines

```c
#include "slash.h"
int main (int argc, char **argv) {
    LAS *h;
    h = las_open (argv[1], "rb");
    las_header_display (stdout, h);
    las_close (h);
    return 0;
}
```

Or even shorter:

```c
#include "slash.h"
int main (int argc, char **argv) {
    return las_header_display (stdout,
                    las_open (argv[1], "rb"));
}
```

It's compact

```
> gcc -Os -o slashinfo -I../include slashinfo.c
> strip --strip-all slashinfo.exe
> dir

2013-10-31  13:17                20.992 slashinfo.exe
```

It’s tiny

```
> gcc -Os -x c -c -o slash.o slash.h
> strip --strip-all slash.o
> dir slash.o

2013-11-01  05:48                 6.660 slash.o
```

A LAS reader library in 6.5 kB!

It's fast

```
Read 392826 points.

(bla bla bla...)

********** timings for sLASh **********
Min:  0.0644 s
Max:  0.0657 s
Mean: 0.0650 s

.
```

Python bindings and timings by Simon L. Kokkendorf

It's fast

```
Read 392826 points.

(bla bla bla...)

********** timings for libLAS **********
Min:  8.4942 s
Max:  8.5881 s
Mean: 8.5480 s

(8.548 / 0.065 = 131.5)
```

Python bindings and timings by Simon L. Kokkendorf

What does it look like?

```c
#include "slash.h"

int main (int argc, char **argv) {
    LAS *h;
    double x, y, z;

    h = las_open (argv[1], "rb");
    while  (las_read (h))  {
        x = las_x (h);
        y = las_y (h);
        z = las_z (h);
        printf ("%f %f %f\n", x, y, z);
    }

    las_close (h);
    return 0;
}
```

. . . compared to LASlib

```
#include "lasreader.hpp"
#include "laswriter.hpp"

int main(int argc, char *argv[]) {
  LASreadOpener lasreadopener;
  lasreadopener.set_file_name("original.las");
  LASreader* lasreader = lasreadopener.open();

  LASwriteOpener laswriteopener;
  laswriteopener.set_file_name("compressed.laz");
  LASwriter* laswriter = laswriteopener.open(&lasreader->header);

  while (lasreader->read_point())
      laswriter->write_point(&lasreader->point);

  laswriter->close();
  delete laswriter;
  lasreader->close();
  delete lasreader;
  return 0;
}
```

Data types

```
struct lasheader;
typedef struct lasheader LAS;

struct las_nrgb;
typedef struct las_nrgb    LAS_NRGB;

struct las_wf_desc;
typedef struct las_wf_desc LAS_WAVEFORM_DESCRIPTOR;

struct lasrecord;
typedef struct lasrecord LAS_RECORD;

struct lasvlr;
typedef struct lasvlr    LAS_VLR;
```

Main API

```
LAS   *las_open (const char *filename, const char *mode);

void   las_close (LAS *h);

int    las_seek (LAS *h, size_t pos, int whence);

size_t las_read (LAS *h);
```

Record access API

```
double        las_x (const LAS *h);
double        las_y (const LAS *h);
double        las_z (const LAS *h);
double        las_gps_time (const LAS *h);
double        las_intensity (const LAS *h);

unsigned int las_class (const LAS *h);
unsigned int las_class_flags (const LAS *h);
unsigned int las_flag_synthetic (const LAS *h);
unsigned int las_flag_key_point (const LAS *h);
unsigned int las_flag_withheld (const LAS *h);
unsigned int las_flag_overlap (const LAS *h);

unsigned int las_return_number (const LAS *h);
unsigned int las_number_of_returns (const LAS *h);
unsigned long long las_record_number (const LAS *h);

double        las_scan_angle_rank (const LAS *h);
int           las_point_source_id (const LAS *h);
int           las_scanner_channel (const LAS *h);
unsigned int las_scan_direction (const LAS *h);
unsigned int las_edge_of_flight_line (const LAS *h);

LAS_WAVEFORM_DESCRIPTOR las_waveform_descriptor (const LAS *h);
LAS_NRGB las_colour (const LAS *h);
```

Variable length records API

```
LAS_VLR *las_vlr_read (LAS *h, int type) ;
void las_vlr_free (LAS_VLR *self) ;
```

Printing and formatting API

```
struct tm yd2dmy(int y, int d) ;
void las_record_display (FILE *f, const LAS *h);
void las_header_display (FILE *f, const LAS *h);
void las_vlr_display (LAS_VLR *self, FILE *stream);
void las_vlr_display_all (LAS *h, FILE *stream);
```

Low level portability functions

```
void memcpy_swapping (void *dest, const void *src, size_t offset
long long get_signed_16 (const void *buf, size_t offset);
long long get_signed_32 (const void *buf, size_t offset);
long long get_signed_64 (const void *buf, size_t offset);
unsigned long long get_unsigned_16 (const void *buf, size_t offs
unsigned long long get_unsigned_32 (const void *buf, size_t offs
unsigned long long get_unsigned_64 (const void *buf, size_t offs
float get_float (const void *buf, size_t offset);
double get_double (const void *buf, size_t offset);
```

LAS (1)

```
/* LAS file header straightforwardly implemented from the LAS 1.0--1.4 specs */
struct lasheader {
    char                    signature[8];     /* LASF */
    unsigned short          file_source_id;
    unsigned short          global_encoding;

    unsigned long           project_id_1;
    unsigned short          project_id_2;
    unsigned short          project_id_3;
    unsigned char           project_id_4[8];

    unsigned char           version_major;
    unsigned char           version_minor;

    char                    system_id[32];
    char                    generated_by[32];

    unsigned short          file_creation_day_of_year;
    unsigned short          file_creation_year;

    unsigned short          header_size;
    unsigned long           offset_to_point_data;
    unsigned long           number_of_variable_length_records;

    unsigned char           point_data_format;
    unsigned short          point_data_record_length;
    unsigned long long      number_of_point_records;
    unsigned long long      number_of_points_by_return[15];
```

LAS (2)

```
    double                      x_scale;
    double                      y_scale;
    double                      z_scale;
    double                      x_offset;
    double                      y_offset;
    double                      z_offset;

    double                      x_max;
    double                      x_min;
    double                      y_max;
    double                      y_min;
    double                      z_max;
    double                      z_min;

    unsigned long long          offset_to_waveform_data_packet_record;
    unsigned long long          start_of_extended_vlrs;
    unsigned long long          number_of_extended_vlrs;

    /* additional fields for internal use by sLASh */
    size_t  next_record;
    FILE   *f;
    char mode[256];
    size_t class_histogram[256];

    /* number of decimals recommended (computed from scale factors by las_open)*/
    int nx, ny, nz;
    unsigned char raw[8192];
    unsigned char record[1024];
};
```

Colour and Waveforms

```
/* Colour information - types 2, 3, 5, 7 (rgb),
                            and 8, 10 (nrgb)  */
struct las_nrgb {double n, r, g, b;};


/* Waveform information - types 4, 5, 9, 10 */
struct las_wf_desc {
    unsigned char      descriptor_index;
    float              return_point_location;
    float              x_t, y_t, z_t;
    unsigned long long offset_to_data;
    unsigned long long packet_size;
};
```

LAS record – in preparation for writing

```
struct lasrecord {
    /* The common (unpacked) subset for all record types */
    double x, y, z;
    double intensity;

    /* Flags and narrow data fields from bytes 14-15 */
    unsigned int return_number, number_of_returns;
    unsigned int scanner_channel, scan_direction, edge_of_flight_line;
    /* Classification flags (overlap: types 6-10 only) */
    unsigned int synthetic, key_point, withheld, overlap;

    unsigned int classification; /* "class" is reserved under C++, hence "classificat

    double scan_angle;
    unsigned char user_data;
    unsigned int  point_source_id;

    /* Record types 0 and 2 omits the GPS time */
    double gps_time;

    /* Colour information - types 2, 3, 5, 7 (rgb), and 8, 10 (nrgb) */
    LAS_NRGB colour;

    /* Waveform information - types 4, 5, 9, 10*/
    LAS_WAVEFORM_DESCRIPTOR waveform;
};
```

Variable length records

```
struct lasvlr {
    unsigned long long reserved;
    char user_id[16];
    unsigned long long record_id;
    unsigned long long payload_size;
    char description[32];
    /* -------------------- */
    fpos_t pos;
    int type; /* vlr: 0, evlr: 1 */
    unsigned char *payload;
};
```