August 25th 2011

Requested Addendum for the LAS 1.4 specification

Hello,

the LAS format allows to store n "extra bytes" for every LAS point simply by specifying a point size that is n bytes larger thank minimally required by the point type in the LAS header. These extra bytes can be used to store user specific data. However, when doing so one has to remember what those extra bytes mean as they do not get described anywhere and it is impossible for other users to get any understanding what those "extra bytes" may mean (other than with an accompanying "meta_data.txt" file).

I have an academic partner (OPALS at TU Vienna) that would like to switch their workflow to LAS as they like the simplicity, efficiency, and wide-spead acceptance of the format. However, they occasionally need to store "extra bytes" per point such as the laser pulse width of a return and want those "extra bytes" documented.

Also, I am consulting with a joint working group of the German state land use and mapping agencies. All state mapping agency workflows in Germany are currently ASCII based but they consider switching to LAS. They are also interested in having the ability to occasional add another field to LAS and have it described somehow so it can be shared across agencies and softwares.

Hence, I am proposing to add an official, yet strictly optional, "Extra Bytes" LASF_Spec VLR to the LAS 1.4 specification that describes those "extra bytes". It does not interfere with the current spec any more than the existing "Histogram" LASF_Spec or "Text area description" LASF_Spec.

As an added benefit, an "Extra Bytes" LASF_Spec is a first step towards a self-describing LAS format. Adding the "Extra Bytes" LASF_Spec to LAS provides us a with a no-cost yet real-world "test-run" for self-describing content in LAS that is completely risk-free as the added feature does not affect anyone who does not use extra bytes. It being implemented and used via LAStools, OPALS, (maybe libLAS?), ... will give us user feedback and insights for designing LAS 2.0.

Below I describe the proposed "Extra Bytes" LASF_Spec 4 along the current LASF_Spec 0 - 3.

Regards,

Martin.

Defined Variable Length Records:

Classification lookup: (optional)

User ID: LASF_Spec
Record ID: 0
Record Length after Header: 255 recs X 16 byte struct len (should it not be 256?)

```
struct CLASSIFICATION
{
        unsigned char ClassNumber;
        char Description[15];
};
```

Header lookup for flight-lines:
(Removed with Version 1.1 - Point Source ID in combination with Source ID provides the new scheme for directly encoding flight line number. Thus variable Record ID 1 now becomes reserved for future use.)

User ID: LASF_Spec
Record ID: 1

Histogram: (optional)

User ID: LASF_Spec
Record ID: 2

Text area description: (optional)

User ID: LASF_Spec
Record ID: 3

Extra Bytes: (optional)

User ID: LASF_Spec
Record ID: 4
Record Length after Header: n records x 92 bytes

This (optional) record is only needed for LAS files that contain user-defined “extra bytes” for every LAS point. This happens whenever the point record size is set to a larger value than required by the point type. For example, when a LAS file containing point type 1 has a point record size of 32 instead of 28 then there are 4 extra bytes per point. The “Extra Bytes” VLR contains a simple description of type and meaning of these extra bytes so they can be useful to other people by - optionally - exposing these “extra bytes” semantics via the LAS reader. The additional 4 bytes, for example, could be a floating point value that specifies the pulse width. In this case there would only be a single EXTRA_BYTES struct in the payload of this VLR.

The bit mask options specifies whether the min and max range of the value have been set (i.e. are meaningful), whether the scale and/or offset values are set with which the extrabytes are then to be multiplied and translated to reach the actual value, and whether there is a special value that should be interpreted as no_data. By default all bits are zero which is supposed to mean that the values in the corresponding fields are to be disregarded (and will probably be set to zero).
If the selected data_type is less than 8 bytes, the no_data_value, min, and max field should be upcast into 8-byte storage. For any float these 8 bytes would be a double, for any unsigned char, unsigned short, or unsigned long they would be an unsigned long long and for any char, short, or long, they would be a long long.

```
struct EXTRA_BYTES
{
        unsigned char       reserved[2];        // 2 bytes
        unsigned char       data_type;          // 1 byte
        unsigned char       options;            // 1 byte
        char                name[16];           // 16 bytes
        anytype             no_data_value;      // 8 bytes
        anytype             min;                // 8 bytes
        anytype             max;                // 8 bytes
        double              scale;              // 8 bytes
        double              offset;             // 8 bytes
        char                description[32];    // 32 bytes
};                                              // total of 92 bytes
```

Table 1 - Values for Extra Bytes Data Types

Value	*Meaning*	*Size*
0	unsigned char	1 byte
1	char	1 byte
2	unsigned short	2 bytes
3	short	2 bytes
4	unsigned long	4 bytes
5	long	4 bytes
6	unsigned long long	8 bytes
7	long long	8 bytes
8	float	4 bytes
9	double	8 bytes

Table 2 - Option Bit Field Encoding

Bit	*Field Name*	*Description*
0	no_data	If set the no_data value is relevant.
1	min	If set the min value is relevant
2	max	If set the max value is relevant
3	scale	If set each extra byte value should be multiplied by the scale value
4	offset	If set each extra byte value should be translated by the offset value