lasio Documentation

Release 0.31

Kent Inverarity and contributors

May 18, 2023

User guide

Read and write Log ASCII Standard files with Python.

This is a Python 3.7+ package to read and write Log ASCII Standard (LAS) files, used for borehole data such as geophysical, geological, or petrophysical logs. It's compatible with versions 1.2 and 2.0 of the LAS file specification, published by the Canadian Well Logging Society. Support for LAS 3 is being worked on.

lasio is primarily for reading and writing data and metadata to and from LAS files. It is designed to read as many LAS files as possible, including those containing common errors and non-compliant formatting. It can be used directly, but you may want to consider using some other packages, depending on your priorities:

- welly is a Python package that uses lasio for I/O but provides a **lot** more functionality aimed at working with curves, wells, and projects. I would recommend starting there in most cases, to avoid re-inventing the wheel!
- lascheck is focused on checking whether your LAS file meets the specifications.
- lasr is an R package which is designed to read large amounts of data quickly from LAS files; this is a great thing to check out if speed is a priority for you, as lasio is not particularly fast.
- LiDAR surveys are also called "LAS files", but they are quite different and lasio will not help you – check out laspy instead.

lasio stopped supporting Python 2.7 in August 2020. The final version of lasio with Python 2.7 support is version 0.26.

CHAPTER 1

Installation

lasio is written to be compatible with Python 3.7+. The best way to install is using pip.

```
$ pip install lasio
```

This will make sure that the dependency numpy is installed as well.

The final version of lasio with Python 2.7 support is v0.26.

There are some other packages which lasio will use to provide extra functionality if they are installed (pandas, cChardet and/or chardet, and openpyxl). I recommend installing these with:

```
$ pip install "lasio[all]"
```

lasio is now installed.

To upgrade to the latest PyPI version, use:

```
$ pip install --upgrade lasio
```

1.1 Development version

Installing via pip gets the latest release which has been published on PyPI. If you want, you can install the latest changes from GitHub:

```
$ pip install https://github.com/kinverarity1/lasio/archive/master.zip
```

CHAPTER 2

Basic example

```
>>> import lasio
```

You can use *lasio.read()* to open any file or URL. For this tutorial I will use the `lasio.examples` module to load a LAS file which is bundled with lasio:

```
>>> import lasio.examples
>>> las = lasio.examples.open("1001178549.las")
```

The *lasio.read()* function returns a *lasio.LASFile* object. Each of the standard LAS sections can be accessed as an attribute:

```
>>> las.version
[HeaderItem(mnemonic="VERS", unit=, value="2.0", descr="CWLS Log ASCII Standard -V...
↪),
 HeaderItem(mnemonic="WRAP", unit=, value="YES", descr="Multiple lines per depth ...)]
```

Each LAS section is represented as a *lasio.SectionItems* object. The others, for LAS 2.0 files, are present as `las.well`, `las.curves`, and `las.params`; the ~O section is a string accessible at `las.other`.

You can also see the sections printed as an easier-to-read table:

```
>>> print(las.curves)
Mnemonic  Unit    Value         Description
--------  ----    -----         -----------
DEPT      FT      0   1  0  0   1 DEPTH
GSGR      API     31 310  0  0  2 GAMMA RAY
GSTK      API     31 797  0  0  3 ????????
GST       API     99 999 99  0  4 ????????
GSK       PERCNT  31 721  1  0  5 ????????
GSTH      PPM     31 790  0  0  6 THORIUM
GSUR      PPM     31 792  0  0  7 URANIUM
NCNPL     PERCNT  42 890  1  0  8 NEUTRON POROSITY (LIMESTONE)
DLDPL     PERCNT  43 890 10  0  9 DENSITY POROSITY (LIMESTONE)
DLDC      GM/CC   43 356  0  0  10 DENSITY CORRECTION
```

(continues on next page)

(continued from previous page)

```
DLPE      B/E     43 358  0  0  11 PHOTO-ELECTRIC EFFECT
DLDN      GM/CC   43 350  0  0  12 BULK DENSITY
DLCL      INCHES  43 280  0  0  13 CALIPER
DLTN      LBS     43 635  0  0  14 ????????
IDGR      API     7 310  0  0   15 GAMMA RAY
ACCL1     INCHES  60 280  1  0  16 DENSITY CALIPER
ACCL2     INCHES  60 280  2  0  17 NEUTRON CALIPER
ACTC      US/FT   60 520  0  0  18 SONIC INTERVAL TRANSIT TIME (COMPENSATED)
ACAPL     PERCNT  60 890 20  0  19 POROSITY
IDIM      OHMM    7 120 44  0   20 MEDIUM INDUCTION
IDID      OHMM    7 120 46  0   21 DEEP INDUCTION
IDIDC     MMHOS   7 110 46  0   22 INDUCTION (CONDUCTIVITY UNITS)
IDL3      OHMM    7 220  3  0   23 FOCUSSED RESISTIVITY
IDTN      LBS     7 635  0  0   24 ????????
IDSP      MVOLT   7  10  0  0   25 SPONTANEOUS POTENTIAL
MEL1      OHMM    15 250  2  0  26 MICRO INVERSE 1"
ME        OHMM    15 252  2  0  27 MICRO NORMAL 2"
```

The data is present as a `numpy.ndarray` at `las.data`:

```
>>> las.data.shape
(5, 27)
>>> las.data
array([[1.7835000e+03,           nan,           nan,           nan,
                  nan,           nan,           nan,           nan,
                  nan,           nan,           nan,           nan,
                  nan,           nan, 5.0646500e+01, 8.3871000e+00,
        8.4396000e+00, 5.5100000e+01, 5.6900000e-02, 5.6000000e+02,
        1.7500000e+02, 5.0000000e-02, 4.5330000e-01, 1.8930420e+03,
        9.2605000e+01,           nan,           nan],
       [1.7837500e+03,           nan,           nan,           nan,
                  nan,           nan,           nan,           nan,
                  nan,           nan,           nan,           nan,
                  nan,           nan, 4.9676700e+01, 8.3951000e+00,
        8.4460000e+00, 5.4355500e+01, 5.9000000e-02, 5.6000000e+02,
        1.7500000e+02, 5.0000000e-02, 4.5340000e-01, 1.8523320e+03,
        9.2778000e+01,           nan,           nan],
       [1.7840000e+03,           nan,           nan,           nan,
                  nan,           nan,           nan,           nan,
                  nan,           nan,           nan,           nan,
                  nan,           nan, 4.8631300e+01, 8.4052000e+00,
        8.4460000e+00, 5.4444400e+01, 5.8100000e-02, 5.6000000e+02,
        1.7500000e+02, 5.0000000e-02, 4.5370000e-01, 1.8319766e+03,
        9.2948200e+01,           nan,           nan],
       [1.7842500e+03,           nan,           nan,           nan,
                  nan,           nan,           nan,           nan,
                  nan,           nan,           nan,           nan,
                  nan,           nan, 4.7771700e+01, 8.4173000e+00,
        8.4438000e+00, 5.5311100e+01, 5.7700000e-02, 5.6000000e+02,
        1.7500000e+02, 5.0000000e-02, 4.5380000e-01, 1.8319766e+03,
        9.3110300e+01,           nan,           nan],
       [1.7845000e+03,           nan,           nan,           nan,
                  nan,           nan,           nan,           nan,
                  nan,           nan,           nan,           nan,
                  nan,           nan, 4.8114900e+01, 8.4253000e+00,
        8.4460000e+00, 5.6322200e+01, 5.8500000e-02, 5.6000000e+02,
        1.7500000e+02, 5.0000000e-02, 4.5390000e-01, 1.8116211e+03,
```

(continues on next page)

(continued from previous page)

```
        9.3267100e+01,            nan,            nan]])
```

Although it might be easier for you to iterate over the curves:

```
>>> for curve in las.curves:
...     print(curve.mnemonic + ": " + str(curve.data))
DEPT: [1783.5  1783.75 1784.   1784.25 1784.5 ]
GSGR: [nan nan nan nan nan]
GSTK: [nan nan nan nan nan]
GST: [nan nan nan nan nan]
GSK: [nan nan nan nan nan]
GSTH: [nan nan nan nan nan]
GSUR: [nan nan nan nan nan]
NCNPL: [nan nan nan nan nan]
DLDPL: [nan nan nan nan nan]
DLDC: [nan nan nan nan nan]
DLPE: [nan nan nan nan nan]
DLDN: [nan nan nan nan nan]
DLCL: [nan nan nan nan nan]
DLTN: [nan nan nan nan nan]
IDGR: [50.6465 49.6767 48.6313 47.7717 48.1149]
ACCL1: [8.3871 8.3951 8.4052 8.4173 8.4253]
ACCL2: [8.4396 8.446  8.446  8.4438 8.446 ]
ACTC: [55.1    54.3555 54.4444 55.3111 56.3222]
ACAPL: [0.0569 0.059  0.0581 0.0577 0.0585]
IDIM: [560. 560. 560. 560. 560.]
IDID: [175. 175. 175. 175. 175.]
IDIDC: [0.05 0.05 0.05 0.05 0.05]
IDL3: [0.4533 0.4534 0.4537 0.4538 0.4539]
IDTN: [1893.042  1852.332  1831.9766 1831.9766 1811.6211]
IDSP: [92.605  92.778  92.9482 93.1103 93.2671]
MEL1: [nan nan nan nan nan]
ME: [nan nan nan nan nan]
```

The first curve in the LAS file – usually the depth – is present as `las.index`, and curves are also accessible from the LASFile object as items. For example:

```
>>> las.index
array([1783.5 , 1783.75, 1784.  , 1784.25, 1784.5 ])
>>> las["IDTN"]
array([1893.042 , 1852.332 , 1831.9766, 1831.9766, 1811.6211])
```

CHAPTER 3

Integration with pandas.DataFrame

The *lasio.LASFile.df()* method converts the LAS data to a `pandas.DataFrame`. The first curve in the LAS file is used for the dataframe's index. See below for an example using this LAS file:

```
>>> import lasio.examples
>>> las = lasio.examples.open('6038187_v1.2.las')
>>> df = las.df()
>>> print(df)
          CALI    DFAR  DNEAR     GAMN  NEUT       PR      SP      COND
DEPT
0.05    49.765  4.587  3.382      NaN   NaN      NaN     NaN       NaN
0.10    49.765  4.587  3.382 -2324.28   NaN  115.508 -3.049 -116.998
0.15    49.765  4.587  3.382 -2324.28   NaN  115.508 -3.049 -116.998
0.20    49.765  4.587  3.382 -2324.28   NaN  115.508 -3.049 -116.998
0.25    49.765  4.587  3.382 -2324.28   NaN  115.508 -3.049 -116.998
...        ...    ...    ...       ...   ...      ...     ...       ...
136.40  48.604    NaN    NaN       NaN   NaN      NaN     NaN       NaN
136.45  48.555    NaN    NaN       NaN   NaN      NaN     NaN       NaN
136.50  48.555    NaN    NaN       NaN   NaN      NaN     NaN       NaN
136.55  48.438    NaN    NaN       NaN   NaN      NaN     NaN       NaN
136.60 -56.275    NaN    NaN       NaN   NaN      NaN     NaN       NaN

[2732 rows x 8 columns]
```

If you prefer the DEPT curve not to be set as the `pandas.DataFrame` index, then you can reset the index:

```
>>> df2 = las.df().reset_index()
>>> print(df2)
        DEPT    CALI   DFAR  DNEAR     GAMN  NEUT       PR      SP      COND
0       0.05  49.765  4.587  3.382      NaN   NaN      NaN     NaN       NaN
1       0.10  49.765  4.587  3.382 -2324.28   NaN  115.508 -3.049 -116.998
2       0.15  49.765  4.587  3.382 -2324.28   NaN  115.508 -3.049 -116.998
3       0.20  49.765  4.587  3.382 -2324.28   NaN  115.508 -3.049 -116.998
4       0.25  49.765  4.587  3.382 -2324.28   NaN  115.508 -3.049 -116.998
...      ...     ...    ...    ...       ...   ...      ...     ...       ...
```

(continues on next page)

(continued from previous page)

```
2727  136.40  48.604    NaN    NaN      NaN   NaN      NaN    NaN      NaN
2728  136.45  48.555    NaN    NaN      NaN   NaN      NaN    NaN      NaN
2729  136.50  48.555    NaN    NaN      NaN   NaN      NaN    NaN      NaN
2730  136.55  48.438    NaN    NaN      NaN   NaN      NaN    NaN      NaN
2731  136.60 -56.275    NaN    NaN      NaN   NaN      NaN    NaN      NaN

[2732 rows x 9 columns]
```

But let's continue with `df`, which has DEPT set to the index. There are some summary methods in pandas which are handy for data exploration:

```
>>> df.head(10)
        CALI    DFAR  DNEAR     GAMN  NEUT       PR     SP      COND
DEPT
0.05  49.765  4.587  3.382      NaN   NaN      NaN    NaN       NaN
0.10  49.765  4.587  3.382 -2324.28   NaN  115.508 -3.049 -116.998
0.15  49.765  4.587  3.382 -2324.28   NaN  115.508 -3.049 -116.998
0.20  49.765  4.587  3.382 -2324.28   NaN  115.508 -3.049 -116.998
0.25  49.765  4.587  3.382 -2324.28   NaN  115.508 -3.049 -116.998
0.30  49.765  4.587  3.382 -2324.28   NaN  115.508 -3.049 -116.998
0.35  49.765  4.587  3.382 -2324.28   NaN  115.508 -3.049 -116.998
0.40  49.765  4.587  3.382 -2324.28   NaN  115.508 -3.049 -116.998
0.45  49.765  4.587  3.382 -2324.28   NaN  115.508 -3.049 -116.998
0.50  49.765  4.587  3.382 -2324.28   NaN  115.508 -3.049 -116.998
>>> df.tail(40)
           CALI    DFAR  DNEAR     GAMN   NEUT       PR     SP      COND
DEPT
134.65  100.983  1.563  1.357 -2324.28  158.0  115.508 -3.049  578.643
134.70  100.833  1.570  1.357      NaN    NaN      NaN    NaN  571.233
134.75   93.760  1.582  1.378      NaN    NaN      NaN    NaN  565.552
134.80   88.086  1.561  1.361      NaN    NaN      NaN    NaN  570.490
134.85   86.443  1.516  1.338      NaN    NaN      NaN    NaN  574.937
134.90   79.617  5.989  1.356      NaN    NaN      NaN    NaN  579.137
134.95   65.236  4.587  1.397      NaN    NaN      NaN    NaN       NaN
135.00   55.833  4.587  1.351      NaN    NaN      NaN    NaN       NaN
135.05   49.061  4.587  1.329      NaN    NaN      NaN    NaN       NaN
135.10   49.036    NaN    NaN      NaN    NaN      NaN    NaN       NaN
135.15   49.024    NaN    NaN      NaN    NaN      NaN    NaN       NaN
135.20   49.005    NaN    NaN      NaN    NaN      NaN    NaN       NaN
135.25   48.999    NaN    NaN      NaN    NaN      NaN    NaN       NaN
135.30   48.987    NaN    NaN      NaN    NaN      NaN    NaN       NaN
135.35   48.980    NaN    NaN      NaN    NaN      NaN    NaN       NaN
135.40   48.962    NaN    NaN      NaN    NaN      NaN    NaN       NaN
135.45   48.962    NaN    NaN      NaN    NaN      NaN    NaN       NaN
135.50   48.925    NaN    NaN      NaN    NaN      NaN    NaN       NaN
135.55   48.931    NaN    NaN      NaN    NaN      NaN    NaN       NaN
135.60   48.919    NaN    NaN      NaN    NaN      NaN    NaN       NaN
135.65   48.900    NaN    NaN      NaN    NaN      NaN    NaN       NaN
135.70   48.882    NaN    NaN      NaN    NaN      NaN    NaN       NaN
135.75   48.863    NaN    NaN      NaN    NaN      NaN    NaN       NaN
135.80   48.857    NaN    NaN      NaN    NaN      NaN    NaN       NaN
135.85   48.839    NaN    NaN      NaN    NaN      NaN    NaN       NaN
135.90   48.808    NaN    NaN      NaN    NaN      NaN    NaN       NaN
135.95   48.802    NaN    NaN      NaN    NaN      NaN    NaN       NaN
136.00   48.789    NaN    NaN      NaN    NaN      NaN    NaN       NaN
136.05   48.771    NaN    NaN      NaN    NaN      NaN    NaN       NaN
```

(continues on next page)

(continued from previous page)

```
136.10   48.765    NaN    NaN      NaN    NaN      NaN    NaN      NaN
136.15   48.752    NaN    NaN      NaN    NaN      NaN    NaN      NaN
136.20   48.734    NaN    NaN      NaN    NaN      NaN    NaN      NaN
136.25   48.684    NaN    NaN      NaN    NaN      NaN    NaN      NaN
136.30   48.666    NaN    NaN      NaN    NaN      NaN    NaN      NaN
136.35   48.647    NaN    NaN      NaN    NaN      NaN    NaN      NaN
136.40   48.604    NaN    NaN      NaN    NaN      NaN    NaN      NaN
136.45   48.555    NaN    NaN      NaN    NaN      NaN    NaN      NaN
136.50   48.555    NaN    NaN      NaN    NaN      NaN    NaN      NaN
136.55   48.438    NaN    NaN      NaN    NaN      NaN    NaN      NaN
136.60  -56.275    NaN    NaN      NaN    NaN      NaN    NaN      NaN
>>> df.describe()
              CALI         DFAR        DNEAR         GAMN         NEUT  \
count  2732.000000  2701.000000  2701.000000  2691.000000  2492.000000
mean     97.432002     1.767922     1.729209  -102.330033   441.600013
std      13.939547     0.480333     0.372412   630.106420   370.138208
min     -56.275000     0.725000     0.657001 -2324.280000    81.001800
25%     101.077500     1.526000     1.535000    55.783000   158.002000
50%     101.426000     1.758000     1.785000    74.376900   256.501500
75%     101.582000     1.993000     1.948000    88.326900   680.500250
max     103.380000     5.989000     3.382000   169.672000  1665.990000

                 PR           SP          COND
count   2692.000000  2692.000000  2697.000000
mean   17940.522307    90.393464   478.670791
std    22089.297212    26.725547   753.869866
min      115.508000    -3.049000  -116.998000
25%     2652.470000    93.495500   200.981000
50%     2709.345000    99.994000   266.435000
75%    50499.900000   100.623000   505.530000
max    50499.900000   102.902000  4978.160000
```

Any changes that you make to the DataFrame can be brought back into the LASFile object with *lasio.LASFile.set_data()*.

There's obviously a problem with the GAMN log: -2324.28 is not a valid value. Let's fix that.

```
>>> import numpy as np
>>> df['GAMN'][df['GAMN'] == -2324.28] = np.nan
>>> df.describe()['GAMN']
count    2491.000000
mean       76.068198
std        23.120160
min        13.946000
25%        60.434100
50%        76.700700
75%        90.647500
max       169.672000
Name: GAMN, dtype: float64
```

Let's create a new log with the moving average of the GAMN log, over 1 m. This is easy enough to do with the pandas `pandas.Series.rolling()` method and the LAS file's STEP value:

```
>>> df['GAMN_avg'] = df['GAMN'].rolling(int(1 / las.well.STEP.value), center=True).
↪mean()
```

Now we want to apply this DataFrame `df` back to the `las` LASFile object, and check that it's all there:

Warning: When using `las.set_data(df)`, don't forget that `df.index` will be used for the first curve of the LAS file.

```
>>> las.set_data(df)
>>> las.curves
[CurveItem(mnemonic="DEPT", unit="M", value="", descr="DEPTH", original_mnemonic="DEPT
↪", data.shape=(2732,)),
 CurveItem(mnemonic="CALI", unit="MM", value="", descr="CALI", original_mnemonic="CALI
↪", data.shape=(2732,)),
 CurveItem(mnemonic="DFAR", unit="G/CM3", value="", descr="DFAR", original_mnemonic=
↪"DFAR", data.shape=(2732,)),
 CurveItem(mnemonic="DNEAR", unit="G/CM3", value="", descr="DNEAR", original_mnemonic=
↪"DNEAR", data.shape=(2732,)),
 CurveItem(mnemonic="GAMN", unit="GAPI", value="", descr="GAMN", original_mnemonic=
↪"GAMN", data.shape=(2732,)),
 CurveItem(mnemonic="NEUT", unit="CPS", value="", descr="NEUT", original_mnemonic=
↪"NEUT", data.shape=(2732,)),
 CurveItem(mnemonic="PR", unit="OHM/M", value="", descr="PR", original_mnemonic="PR",␣
↪data.shape=(2732,)),
 CurveItem(mnemonic="SP", unit="MV", value="", descr="SP", original_mnemonic="SP",␣
↪data.shape=(2732,)),
 CurveItem(mnemonic="COND", unit="MS/M", value="", descr="COND", original_mnemonic=
↪"COND", data.shape=(2732,)),
 CurveItem(mnemonic="GAMN_avg", unit="", value="", descr="", original_mnemonic="GAMN_
↪avg", data.shape=(2732,))]
>>> las.df().describe()
              CALI         DFAR        DNEAR         GAMN         NEUT  \
count  2732.000000  2701.000000  2701.000000  2491.000000  2492.000000
mean     97.432002     1.767922     1.729209    76.068198   441.600013
std      13.939547     0.480333     0.372412    23.120160   370.138208
min     -56.275000     0.725000     0.657001    13.946000    81.001800
25%     101.077500     1.526000     1.535000    60.434100   158.002000
50%     101.426000     1.758000     1.785000    76.700700   256.501500
75%     101.582000     1.993000     1.948000    90.647500   680.500250
max     103.380000     5.989000     3.382000   169.672000  1665.990000

                PR           SP         COND     GAMN_avg
count  2692.000000  2692.000000  2697.000000  2472.000000
mean  17940.522307    90.393464   478.670791    76.326075
std   22089.297212    26.725547   753.869866    18.208038
min     115.508000    -3.049000  -116.998000    24.753655
25%    2652.470000    93.495500   200.981000    64.848379
50%    2709.345000    99.994000   266.435000    77.747517
75%   50499.900000   100.623000   505.530000    88.323376
max   50499.900000   102.902000  4978.160000   120.049300
```

All good, the new curve is in there.

See the pandas documentation for more information!

CHAPTER 4

Header section metadata

4.1 Tutorial

One of the primary motivations in writing lasio was to be able to reliably parse LAS header sections. This is working well for LAS 1.2 and 2.0 files, and partially for LAS 3.0 files.

Note: lasio does not require LAS files to be strictly compliant with the standards, and you should not expect lasio to raise an exception or error for files which are clearly not conforming to the standards. The goal of lasio is to parse metadata and data quietly, not to fail unnecessarily.

```
>>> import lasio.examples
>>> las = lasio.examples.open('6038187_v1.2_short.las')
```

The header sections are stored in the dictionary `las.sections`:

```
>>> type(las.sections)
dict
>>> las.sections.keys()
dict_keys(['Version', 'Well', 'Curves', 'Parameter', 'Other'])
```

These are special names reserved for LAS 1.2 and 2.0 files, as defined by the standard. Non-standard header sections are also allowed but not fully parsed.

LAS file	Read in as	References in `LASFile`
~v or ~V	*lasio.SectionItems*	`LASFile.version` and `LASFile.sections['Version']`
~w or ~W	*lasio.SectionItems*	`LASFile.well` and `LASFile.sections['Well']`
~c or ~C	*lasio.SectionItems*	`LASFile.curves` and `LASFile.sections['Curves']`
~p or ~P	*lasio.SectionItems*	`LASFile.params` and `LASFile.sections['Parameter']`
~o or ~O	`str`	`LASFile.other` and `LASFile.sections['Other']`
~extra section	`str`	`LASFile.sections['extra section']`
~a or ~A	`numpy.ndarray`	`LASFile.data` or each column is in `LASFile.curves[...].data`

For example:

```
>>> las.sections['Version']
[HeaderItem(mnemonic="VERS", unit="", value="2.0", descr="CWLS LOG ASCII STANDA"),
 HeaderItem(mnemonic="WRAP", unit="", value="NO", descr="ONE LINE PER DEPTH STE")]

>>> las.version
[HeaderItem(mnemonic="VERS", unit="", value="2.0", descr="CWLS LOG ASCII STANDA"),
 HeaderItem(mnemonic="WRAP", unit="", value="NO", descr="ONE LINE PER DEPTH STE")]
```

Sections themselves are represented by *lasio.SectionItems* objects. This is a `list` which has been extended to allow you to access the items within by their mnemonic:

```
>>> las.version.VERS
HeaderItem(mnemonic="VERS", unit="", value="2.0", descr="CWLS LOG ASCII STANDA")
>>> las.version['VERS']
HeaderItem(mnemonic="VERS", unit="", value="2.0", descr="CWLS LOG ASCII STANDA")
>>> las.version[0]
HeaderItem(mnemonic="VERS", unit="", value="2.0", descr="CWLS LOG ASCII STANDA")
```

As you can see, either attribute-style or item-style access is fine.

Let's take a look at the next special section, `~W`:

```
>>> las.well
 [HeaderItem(mnemonic="STRT", unit="M", value="0.05", descr="FIRST INDEX VALUE"),
  HeaderItem(mnemonic="STOP", unit="M", value="136.6", descr="LAST INDEX VALUE"),
  HeaderItem(mnemonic="STEP", unit="M", value="0.05", descr="STEP"),
  HeaderItem(mnemonic="NULL", unit="", value="-99999", descr="NULL VALUE"),
  HeaderItem(mnemonic="COMP", unit="", value="", descr="COMP"),
  HeaderItem(mnemonic="WELL", unit="", value="Scorpio E1", descr="WELL"),
  HeaderItem(mnemonic="FLD", unit="", value="", descr=""),
  HeaderItem(mnemonic="LOC", unit="", value="Mt Eba", descr="LOC"),
  HeaderItem(mnemonic="SRVC", unit="", value="", descr=""),
  HeaderItem(mnemonic="CTRY", unit="", value="", descr=""),
  HeaderItem(mnemonic="STAT", unit="", value="SA", descr="STAT"),
  HeaderItem(mnemonic="CNTY", unit="", value="", descr=""),
  HeaderItem(mnemonic="DATE", unit="", value="15/03/2015", descr="DATE"),
  HeaderItem(mnemonic="UWI", unit="", value="6038-187", descr="WUNT")]
```

The CTRY item is blank. We will set it:

```
>>> las.well.CTRY = 'Australia'
>>> las.well.CTRY
HeaderItem(mnemonic="CTRY", unit="", value="Australia", descr="")
```

Notice that *lasio.SectionItems* plays a little trick here. It actually sets the `header_item.value` attribute, instead of replacing the entire *lasio.HeaderItem* object.

You can set any of the attributes directly. Let's take an example from the `~C` section:

```
>>> las.curves
[CurveItem(mnemonic="DEPT", unit="M", value="", descr="DEPTH", original_mnemonic="DEPT
↪", data.shape=(121,)),
 CurveItem(mnemonic="CALI", unit="MM", value="", descr="CALI", original_mnemonic="CALI
↪", data.shape=(121,)),
 CurveItem(mnemonic="DFAR", unit="G/CM3", value="", descr="DFAR", original_mnemonic=
↪"DFAR", data.shape=(121,)),
 CurveItem(mnemonic="DNEAR", unit="G/CM3", value="", descr="DNEAR", original_mnemonic=
↪"DNEAR", data.shape=(121,)),
 CurveItem(mnemonic="GAMN", unit="GAPI", value="", descr="GAMN", original_mnemonic=
↪"GAMN", data.shape=(121,)),
 CurveItem(mnemonic="NEUT", unit="CPS", value="", descr="NEUT", original_mnemonic=
↪"NEUT", data.shape=(121,)),
 CurveItem(mnemonic="PR", unit="OHM/M", value="", descr="PR", original_mnemonic="PR",␣
↪data.shape=(121,)),
 CurveItem(mnemonic="SP", unit="MV", value="", descr="SP", original_mnemonic="SP",␣
↪data.shape=(121,)),
 CurveItem(mnemonic="COND", unit="MS/M", value="", descr="COND", original_mnemonic=
↪"COND", data.shape=(121,))]
>>> las.curves.PR.unit = 'ohmm'
>>> las.curves.PR
CurveItem(mnemonic="PR", unit="ohmm", value="", descr="PR", original_mnemonic="PR",␣
↪data.shape=(121,))
```

Now let's look more closely at how to manipulate and add or remove items from a section.

```
In [195]: las.params
Out[195]:
[HeaderItem(mnemonic="BS", unit="", value="216 mm", descr="BS"),
 HeaderItem(mnemonic="JOBN", unit="", value="", descr="JOBN"),
 HeaderItem(mnemonic="WPMT", unit="", value="", descr="WPMT"),
 HeaderItem(mnemonic="AGL", unit="", value="", descr="AGL"),
 HeaderItem(mnemonic="PURP", unit="", value="Cased hole stratigraphy", descr="P"),
 HeaderItem(mnemonic="X", unit="", value="560160", descr="X"),
 HeaderItem(mnemonic="CSGL", unit="", value="0 m - 135 m", descr="CSGL"),
 HeaderItem(mnemonic="UNIT", unit="", value="", descr="UNIT"),
 HeaderItem(mnemonic="Y", unit="", value="6686430", descr="Y"),
 HeaderItem(mnemonic="TDL", unit="", value="135.2 m", descr="TDL"),
 HeaderItem(mnemonic="PROD", unit="", value="", descr="PROD"),
 HeaderItem(mnemonic="MUD", unit="", value="Water", descr="MUD"),
 HeaderItem(mnemonic="CSGS", unit="", value="100 mm", descr="CSGS"),
 HeaderItem(mnemonic="ENG", unit="", value="", descr="ENG"),
 HeaderItem(mnemonic="STEP", unit="", value="5 cm", descr="STEP"),
 HeaderItem(mnemonic="FLUIDLEVEL", unit="", value="54 m", descr="FluidLevel"),
 HeaderItem(mnemonic="CSGT", unit="", value="PVC", descr="CSGT"),
 HeaderItem(mnemonic="WIT", unit="", value="", descr="WIT"),
 HeaderItem(mnemonic="EREF", unit="", value="", descr="EREF"),
 HeaderItem(mnemonic="PROJ", unit="", value="", descr="PROJ"),
 HeaderItem(mnemonic="ZONE", unit="", value="53J", descr="ZONE"),
```

(continues on next page)

(continued from previous page)

```
 HeaderItem(mnemonic="DREF", unit="", value="GL", descr="DREF"),
 HeaderItem(mnemonic="TDD", unit="", value="136 m", descr="TDD")]
```

We want to rename the DREF mnemonic as LMF. We can do so by changing the `header_item.mnemonic` attribute.

```
>>> las.params.DREF.mnemonic = 'LMF'
>>> las.params
[HeaderItem(mnemonic="BS", unit="", value="216 mm", descr="BS"),
 HeaderItem(mnemonic="JOBN", unit="", value="", descr="JOBN"),
 HeaderItem(mnemonic="WPMT", unit="", value="", descr="WPMT"),
 HeaderItem(mnemonic="AGL", unit="", value="", descr="AGL"),
 HeaderItem(mnemonic="PURP", unit="", value="Cased hole stratigraphy", descr="P"),
 HeaderItem(mnemonic="X", unit="", value="560160", descr="X"),
 HeaderItem(mnemonic="CSGL", unit="", value="0 m - 135 m", descr="CSGL"),
 HeaderItem(mnemonic="UNIT", unit="", value="", descr="UNIT"),
 HeaderItem(mnemonic="Y", unit="", value="6686430", descr="Y"),
 HeaderItem(mnemonic="TDL", unit="", value="135.2 m", descr="TDL"),
 HeaderItem(mnemonic="PROD", unit="", value="", descr="PROD"),
 HeaderItem(mnemonic="MUD", unit="", value="Water", descr="MUD"),
 HeaderItem(mnemonic="CSGS", unit="", value="100 mm", descr="CSGS"),
 HeaderItem(mnemonic="ENG", unit="", value="", descr="ENG"),
 HeaderItem(mnemonic="STEP", unit="", value="5 cm", descr="STEP"),
 HeaderItem(mnemonic="FLUIDLEVEL", unit="", value="54 m", descr="FluidLevel"),
 HeaderItem(mnemonic="CSGT", unit="", value="PVC", descr="CSGT"),
 HeaderItem(mnemonic="WIT", unit="", value="", descr="WIT"),
 HeaderItem(mnemonic="EREF", unit="", value="", descr="EREF"),
 HeaderItem(mnemonic="PROJ", unit="", value="", descr="PROJ"),
 HeaderItem(mnemonic="ZONE", unit="", value="53J", descr="ZONE"),
 HeaderItem(mnemonic="LMF", unit="", value="GL", descr="DREF"),
 HeaderItem(mnemonic="TDD", unit="", value="136 m", descr="TDD")]
```

And now we need to add a new mnemonic.

```
>>> las.params.DRILL = lasio.HeaderItem(mnemonic='DRILL', value='John Smith', descr=
↪'Driller on site')
>>> las.params
[HeaderItem(mnemonic="BS", unit="", value="216 mm", descr="BS"),
 HeaderItem(mnemonic="JOBN", unit="", value="", descr="JOBN"),
 HeaderItem(mnemonic="WPMT", unit="", value="", descr="WPMT"),
 HeaderItem(mnemonic="AGL", unit="", value="", descr="AGL"),
 HeaderItem(mnemonic="PURP", unit="", value="Cased hole stratigraphy", descr="P"),
 HeaderItem(mnemonic="X", unit="", value="560160", descr="X"),
 HeaderItem(mnemonic="CSGL", unit="", value="0 m - 135 m", descr="CSGL"),
 HeaderItem(mnemonic="UNIT", unit="", value="", descr="UNIT"),
 HeaderItem(mnemonic="Y", unit="", value="6686430", descr="Y"),
 HeaderItem(mnemonic="TDL", unit="", value="135.2 m", descr="TDL"),
 HeaderItem(mnemonic="PROD", unit="", value="", descr="PROD"),
 HeaderItem(mnemonic="MUD", unit="", value="Water", descr="MUD"),
 HeaderItem(mnemonic="CSGS", unit="", value="100 mm", descr="CSGS"),
 HeaderItem(mnemonic="ENG", unit="", value="", descr="ENG"),
 HeaderItem(mnemonic="STEP", unit="", value="5 cm", descr="STEP"),
 HeaderItem(mnemonic="FLUIDLEVEL", unit="", value="54 m", descr="FluidLevel"),
 HeaderItem(mnemonic="CSGT", unit="", value="PVC", descr="CSGT"),
 HeaderItem(mnemonic="WIT", unit="", value="", descr="WIT"),
 HeaderItem(mnemonic="EREF", unit="", value="", descr="EREF"),
```

(continues on next page)

(continued from previous page)

```
 HeaderItem(mnemonic="PROJ", unit="", value="", descr="PROJ"),
 HeaderItem(mnemonic="ZONE", unit="", value="53J", descr="ZONE"),
 HeaderItem(mnemonic="LMF", unit="", value="GL", descr="DREF"),
 HeaderItem(mnemonic="TDD", unit="", value="136 m", descr="TDD"),
 HeaderItem(mnemonic="DRILL", unit="", value="John Smith", descr="Driller on si")]
```

Bingo.

What if we want to delete or remove an item? You can delete items the same way you would remove an item from a dictionary. Let's remove the item we just added (DRILL):

```
>>> del las.well["DRILL"]
```

There are methods intended for removing curves. Say you want to remove the PR curve:

```
>>> las.delete_curve("PR")
[CurveItem(mnemonic="DEPT", unit="M", value="", descr="DEPTH", original_mnemonic="DEPT
↪", data.shape=(121,)),
 CurveItem(mnemonic="CALI", unit="MM", value="", descr="CALI", original_mnemonic="CALI
↪", data.shape=(121,)),
 CurveItem(mnemonic="DFAR", unit="G/CM3", value="", descr="DFAR", original_mnemonic=
↪"DFAR", data.shape=(121,)),
 CurveItem(mnemonic="DNEAR", unit="G/CM3", value="", descr="DNEAR", original_mnemonic=
↪"DNEAR", data.shape=(121,)),
 CurveItem(mnemonic="GAMN", unit="GAPI", value="", descr="GAMN", original_mnemonic=
↪"GAMN", data.shape=(121,)),
 CurveItem(mnemonic="NEUT", unit="CPS", value="", descr="NEUT", original_mnemonic=
↪"NEUT", data.shape=(121,)),
 CurveItem(mnemonic="SP", unit="MV", value="", descr="SP", original_mnemonic="SP",␣
↪data.shape=(121,)),
 CurveItem(mnemonic="COND", unit="MS/M", value="", descr="COND", original_mnemonic=
↪"COND", data.shape=(121,))]
```

Warning: Common mistake!

A common job is to iterate through the curves and remove all but a few that you are interested in. When doing this, be careful to iterate over a **copy** of the curves section. See example below.

```
>>> keep_curves = ['DEPT', 'DFAR', 'DNEAR']
>>> for curve in las.curves[:]:
...     if curve.mnemonic not in keep_curves:
...         las.delete_curve(curve.mnemonic)
...
>>> las.curves
[CurveItem(mnemonic="DEPT", unit="M", value="", descr="DEPTH", original_mnemonic="DEPT
↪", data.shape=(121,)),
 CurveItem(mnemonic="DFAR", unit="G/CM3", value="", descr="DFAR", original_mnemonic=
↪"DFAR", data.shape=(121,)),
 CurveItem(mnemonic="DNEAR", unit="G/CM3", value="", descr="DNEAR", original_mnemonic=
↪"DNEAR", data.shape=(121,))]
```

Another common task is to retrieve a header item that may or may not be in the file. If you try ordinary item-style access, as is normal in Python, a KeyError exception will be raised if it is missing:

```
>>> permit = las.well['PRMT']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "c:\devapps\kinverarity\projects\lasio\lasio\las_items.py", line 313, in __
↪getitem__
    raise KeyError("%s not in %s" % (key, self.keys()))
KeyError: "PRMT not in ['STRT', 'STOP', 'STEP', 'NULL', 'COMP', 'WELL', 'FLD', 'LOC',
↪'PROV', 'SRVC', 'DATE', 'UWI']"
```

A better pattern is to use the *lasio.SectionItems.get()* method, which allows you to specify a default value in the case of it missing:

```
>>> permit = las.well.get('PRMT', 'unknown')
>>> permit
HeaderItem(mnemonic="PRMT", unit="", value="unknown", descr="")
```

You can use the `add=True` keyword argument if you would like this header item to be added, as well as returned:

```
>>> permit = las.well.get('PRMT', 'unknown', add=True)
>>> las.well
[HeaderItem(mnemonic="STRT", unit="M", value="0.05", descr="FIRST INDEX VALUE"),
HeaderItem(mnemonic="STOP", unit="M", value="136.6", descr="LAST INDEX VALUE"),
HeaderItem(mnemonic="STEP", unit="M", value="0.05", descr="STEP"),
HeaderItem(mnemonic="NULL", unit="", value="-99999", descr="NULL VALUE"),
HeaderItem(mnemonic="COMP", unit="", value="", descr="COMP"),
HeaderItem(mnemonic="WELL", unit="", value="Scorpio E1", descr="WELL"),
HeaderItem(mnemonic="FLD", unit="", value="", descr=""),
HeaderItem(mnemonic="LOC", unit="", value="Mt Eba", descr="LOC"),
HeaderItem(mnemonic="SRVC", unit="", value="", descr=""),
HeaderItem(mnemonic="CTRY", unit="", value="", descr=""),
HeaderItem(mnemonic="STAT", unit="", value="SA", descr="STAT"),
HeaderItem(mnemonic="CNTY", unit="", value="", descr=""),
HeaderItem(mnemonic="DATE", unit="", value="15/03/2015", descr="DATE"),
HeaderItem(mnemonic="UWI", unit="", value="6038-187", descr="WUNT"),
HeaderItem(mnemonic="PRMT", unit="", value="unknown", descr="")]
```

4.2 Handling special cases of header lines

lasio will do its best to read every line from the header section. Some examples follow for unusual formattings:

4.2.1 Comment lines mixed with header lines

lasio will, by default, treat header lines starting with a "#" hash string as a comment line and ignore it. Spaces before the "#" are stripped off before checking for the "#".

To modify which strings indicate comment lines to ignore pass an ignore_comments tuple to `lasio.read()` or `lasio.examples.open()`.

Example: `lasio.read(file, ignore_comments=("#", "%MyComment")`

4.2.2 Lines without periods

For example take these lines from a LAS file header section:

```
DRILLED  :12/11/2010
PERM DAT :1
TIME     :14:00:32
HOLE DIA :85.7
```

These lines are missing periods between the mnemonic and colon, e.g. a properly formatted version would be `DRILLED. :12/11/2010`.

However, lasio will parse them silently, and correctly, e.g. for the last line the mnemonic will be `HOLE DIA` and the value will be `85.7`, with the description blank.

4.2.3 Lines with colons in the mnemonic and description

Colons are used as a delimiter, but colons can also occur inside the unit, value, and description fields in a LAS file header. Take this line as an example:

```
TIML.hh:mm 23:15 23-JAN-2001:   Time Logger: At Bottom
```

lasio will parse this correctly such that the unit is `hh:mm`, the value is `23:15 21-JAN-2001`, and the description is `Time Logger:  At Bottom`.

4.2.4 Units containing periods

Similarly, periods are used as delimiters, but can also occur as part of the unit field's value, such as in the case of a unit of tenths of an inch (`.1IN`):

```
TDEP  ..1IN                    :  0.1-in
```

lasio will parse the mnemonic as `TDEP` and the unit as `.1IN`.

If there are two adjoining periods, the same behaviour applies:

```
TDEP..1IN                      :  0.1-in
```

lasio parses this line as having mnemonic `TDEP` and unit `.1IN`.

4.2.5 Special case for units which contain spaces

Normally, any whitespace following the unit in a LAS header line delimits the unit from the value. lasio has a special exception for units which may appear with a space. Currently the only one recognised is `1000 lbf`:

```
HKLA           .1000 lbf                                :(RT)
```

This is parsed as mnemonic `HKLA`, unit `1000 lbf`, and value blank, contrary to the usual behaviour which would result in unit `1000` and value `lbf`.

Please raise a GitHub issue for any other units which should be handled in this way.

4.2.6 Mnemonics which contain a period

As with other LAS file parsers, lasio does not parse mnemonics which contain a period - instead, anything after the period will be parsed as the unit:

```
SP.COND .US/M                          :  EC at 25 deg C
```

results in mnemonic `SP`, unit `COND`, and value `.US/CM`.

Warning: These files are non-conforming, and difficult to anticipate.

4.3 Handling errors silently (`ignore_header_errors=True`)

Sometimes lasio cannot make sense of a header line at all. For example:

```
API       .                                          : API Number     (required if␣
↪CTRY = US)
"# Surface Coords: 1,000' FNL & 2,000' FWL"
LATI      .DEG                                       : Latitude  - see Surface Coords␣
↪comment above
LONG      .DEG                                       : Longitude - see Surface Coords␣
↪comment above
```

The line with `"` causes an exception to be raised by default.

Another example is this ~Param section in a LAS file:

```
~PARAMETER INFORMATION
DEPTH     DT       RHOB     NPHI     SFLU     SFLA      ILM      ILD
```

This isn't a header line, and cannot be parsed as such. It results in a `LASHeaderError` exception being raised:

```
>>> las = lasio.examples.open('dodgy_param_sect.las', ignore_header_errors=False)
```

```
Unable to parse line as LAS header: DEPTH     DT       RHOB     NPHI     SFLU     ␣
↪SFLA      ILM      ILD
Traceback (most recent call last):
File "C:\Users\kinve\code\lasio\lasio\reader.py", line 525, in parse_header_section
    values = read_line(line, section_name=parser.section_name2)
File "C:\Users\kinve\code\lasio\lasio\reader.py", line 711, in read_line
    return read_header_line(*args, **kwargs)
File "C:\Users\kinve\code\lasio\lasio\reader.py", line 780, in read_header_line
    mdict = m.groupdict()
AttributeError: 'NoneType' object has no attribute 'groupdict'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
File "<stdin>", line 1, in <module>
File "C:\Users\kinve\code\lasio\lasio\examples.py", line 46, in open
    return open_local_example(filename, **kwargs)
File "C:\Users\kinve\code\lasio\lasio\examples.py", line 106, in open_local_example
    return LASFile(os.path.join(examples_path, *filename.split("/")), **kwargs)
File "C:\Users\kinve\code\lasio\lasio\las.py", line 84, in __init__
    self.read(file_ref, **read_kwargs)
File "C:\Users\kinve\code\lasio\lasio\las.py", line 222, in read
    mnemonic_case=mnemonic_case,
File "C:\Users\kinve\code\lasio\lasio\las.py", line 142, in add_section
```

(continues on next page)

(continued from previous page)

```
    raw_section, **sect_kws
File "C:\Users\kinve\code\lasio\lasio\reader.py", line 536, in parse_header_section
    raise exceptions.LASHeaderError(message)
lasio.exceptions.LASHeaderError: line 31 (section ~PARAMETER INFORMATION): "DEPTH
↪DT       RHOB     NPHI     SFLU     SFLA      ILM      ILD"
```

However, these can be converted from `LASHeaderError` exceptions into `logger.warning()` calls instead by using `lasio.read(..., ignore_header_errors=True)`:

```
>>> las = lasio.examples.open('dodgy_param_sect.las', ignore_header_errors=True)
Unable to parse line as LAS header: DEPTH     DT       RHOB     NPHI     SFLU
↪SFLA      ILM      ILD
line 31 (section ~PARAMETER INFORMATION): "DEPTH     DT       RHOB     NPHI     SFLU
↪   SFLA      ILM      ILD"
```

Only a warning is issued, and the rest of the LAS file loads OK:

```
>>> las.params
[]
>>> las.curves
[CurveItem(mnemonic="DEPT", unit="M", value="", descr="1  DEPTH", original_mnemonic=
↪"DEPT", data.shape=(3,)),
 CurveItem(mnemonic="DT", unit="US/M", value="", descr="2  SONIC TRANSIT TIME",
↪original_mnemonic="DT", data.shape=(3,)),
 CurveItem(mnemonic="RHOB", unit="K/M3", value="", descr="3  BULK DENSITY", original_
↪mnemonic="RHOB", data.shape=(3,)),
 CurveItem(mnemonic="NPHI", unit="V/V", value="", descr="4   NEUTRON POROSITY",
↪original_mnemonic="NPHI", data.shape=(3,)),
 CurveItem(mnemonic="SFLU", unit="OHMM", value="", descr="5  RXO RESISTIVITY",
↪original_mnemonic="SFLU", data.shape=(3,)),
 CurveItem(mnemonic="SFLA", unit="OHMM", value="", descr="6  SHALLOW RESISTIVITY",
↪original_mnemonic="SFLA", data.shape=(3,)),
 CurveItem(mnemonic="ILM", unit="OHMM", value="", descr="7  MEDIUM RESISTIVITY",
↪original_mnemonic="ILM", data.shape=(3,)),
 CurveItem(mnemonic="ILD", unit="OHMM", value="", descr="8  DEEP RESISTIVITY",
↪original_mnemonic="ILD", data.shape=(3,))
]
```

If you are dealing with "messy" LAS data, it might be good to consider using `ignore_header_errors=True`.

4.4 Handling duplicate mnemonics

Take this LAS file as an example, containing this ~C section:

```
~CURVE INFORMATION
DEPT.M                    :  1  DEPTH
DT  .US/M                 :  2  SONIC TRANSIT TIME
RHOB.K/M3                 :  3  BULK DENSITY
NPHI.V/V                  :  4   NEUTRON POROSITY
RXO.OHMM                  :  5  RXO RESISTIVITY
RES.OHMM                  :  6  SHALLOW RESISTIVITY
RES.OHMM                  :  7  MEDIUM RESISTIVITY
RES.OHMM                  :  8  DEEP RESISTIVITY
```

Notice there are three curves with the mnemonic RES. When we load the file in, lasio distinguishes between these duplicates by appending `:1`, `:2`, and so on, to the duplicated mnemonic:

```
>>> las = lasio.read('tests/examples/mnemonic_duplicate2.las')
>>> las.curves
[CurveItem(mnemonic="DEPT", unit="M", value="", descr="1  DEPTH", original_mnemonic=
↪"DEPT", data.shape=(3,)),
 CurveItem(mnemonic="DT", unit="US/M", value="", descr="2  SONIC TRANSIT TIME",␣
↪original_mnemonic="DT", data.shape=(3,)),
 CurveItem(mnemonic="RHOB", unit="K/M3", value="", descr="3  BULK DENSITY", original_
↪mnemonic="RHOB", data.shape=(3,)),
 CurveItem(mnemonic="NPHI", unit="V/V", value="", descr="4   NEUTRON POROSITY",␣
↪original_mnemonic="NPHI", data.shape=(3,)),
 CurveItem(mnemonic="RXO", unit="OHMM", value="", descr="5  RXO RESISTIVITY",␣
↪original_mnemonic="RXO", data.shape=(3,)),
 CurveItem(mnemonic="RES:1", unit="OHMM", value="", descr="6  SHALLOW RESISTIVITY",␣
↪original_mnemonic="RES", data.shape=(3,)),
 CurveItem(mnemonic="RES:2", unit="OHMM", value="", descr="7  MEDIUM RESISTIVITY",␣
↪original_mnemonic="RES", data.shape=(3,)),
 CurveItem(mnemonic="RES:3", unit="OHMM", value="", descr="8  DEEP RESISTIVITY",␣
↪original_mnemonic="RES", data.shape=(3,))
]
>>> las.curves['RES:2']
CurveItem(mnemonic="RES:2", unit="OHMM", value="", descr="7  MEDIUM RESISTIVITY",␣
↪original_mnemonic="RES", data.shape=(3,))
```

It remembers the original mnemonic, so when you write the file back out, they come back:

```
>>> from sys import stdout
>>> las.write(stdout)
~Version ------------------------------------------------
VERS. 1.2 : CWLS LOG ASCII STANDARD - VERSION 1.2
WRAP.  NO : ONE LINE PER DEPTH STEP
~Well ---------------------------------------------------
STRT.M         1670.0 :
STOP.M        1669.75 :
STEP.M         -0.125 :
NULL.          -999.25 :
COMP.          COMPANY : # ANY OIL COMPANY LTD.
WELL.             WELL : ANY ET AL OIL WELL #12
FLD .            FIELD : EDAM
LOC .         LOCATION : A9-16-49-20W3M
PROV.         PROVINCE : SASKATCHEWAN
SRVC. SERVICE COMPANY : ANY LOGGING COMPANY LTD.
DATE.         LOG DATE : 25-DEC-1988
UWI .  UNIQUE WELL ID : 100091604920W300
~Curves -------------------------------------------------
DEPT.M     : 1  DEPTH
DT  .US/M  : 2  SONIC TRANSIT TIME
RHOB.K/M3  : 3  BULK DENSITY
NPHI.V/V   : 4   NEUTRON POROSITY
RXO .OHMM  : 5  RXO RESISTIVITY
RES .OHMM  : 6  SHALLOW RESISTIVITY
RES .OHMM  : 7  MEDIUM RESISTIVITY
RES .OHMM  : 8  DEEP RESISTIVITY
~Params -------------------------------------------------
BHT .DEGC   35.5 : BOTTOM HOLE TEMPERATURE
BS  .MM    200.0 : BIT SIZE
```

(continues on next page)

(continued from previous page)

```
FD  .K/M3 1000.0 : FLUID DENSITY
MATR.        0.0 : NEUTRON MATRIX(0=LIME,1=SAND,2=DOLO)
MDEN.     2710.0 : LOGGING MATRIX DENSITY
RMF .OHMM  0.216 : MUD FILTRATE RESISTIVITY
DFD .K/M3 1525.0 : DRILL FLUID DENSITY
~Other ------------------------------------------------
Note: The logging tools became stuck at 625 meters causing the data
between 625 meters and 615 meters to be invalid.
~ASCII -------------------------------------------------
    1670     123.45       2550       0.45     123.45     123.45      110.2      105.6
    1669.9     123.45       2550       0.45     123.45     123.45      110.2      105.
↪6
    1669.8     123.45       2550       0.45     123.45     123.45      110.2      105.
↪6
```

Note that the same approach is taken for duplicate mnemonics elsewhere in the header, including the ~Well and ~Parameter sections. So, for example, if you have a file which erroneously contains two lines with the UWI mnemonic, then attempting to access `las.well['UWI']` will fail with a KeyError exception, as lasio does not know which of the two mnemonics present should be returned.

4.4.1 Normalising mnemonic case

If there is a mix of upper and lower case characters in the mnemonics, by default lasio will convert all mnemonics to uppercase to avoid problems with producing the :1, :2, :3, and so on. There is a keyword argument which will preserve the original formatting if that is what you prefer.

```
>>> las = lasio.read('tests/examples/mnemonic_case.las')
>>> las.curves
[CurveItem(mnemonic="DEPT", unit="M", value="", descr="1  DEPTH", original_mnemonic=
↪"DEPT", data.shape=(3,)),
 CurveItem(mnemonic="SFLU:1", unit="K/M3", value="", descr="3  BULK DENSITY",␣
↪original_mnemonic="SFLU", data.shape=(3,)),
 CurveItem(mnemonic="NPHI", unit="V/V", value="", descr="4   NEUTRON POROSITY",␣
↪original_mnemonic="NPHI", data.shape=(3,)),
 CurveItem(mnemonic="SFLU:2", unit="OHMM", value="", descr="5  RXO RESISTIVITY",␣
↪original_mnemonic="SFLU", data.shape=(3,)),
 CurveItem(mnemonic="SFLU:3", unit="OHMM", value="", descr="6  SHALLOW RESISTIVITY",␣
↪original_mnemonic="SFLU", data.shape=(3,)),
 CurveItem(mnemonic="SFLU:4", unit="OHMM", value="", descr="7  MEDIUM RESISTIVITY",␣
↪original_mnemonic="SFLU", data.shape=(3,)),
 CurveItem(mnemonic="SFLU:5", unit="OHMM", value="", descr="8  DEEP RESISTIVITY",␣
↪original_mnemonic="SFLU", data.shape=(3,))
]
>>> las = lasio.read('tests/examples/mnemonic_case.las', mnemonic_case='preserve')
>>> las.curves
[CurveItem(mnemonic="Dept", unit="M", value="", descr="1  DEPTH", original_mnemonic=
↪"Dept", data.shape=(3,)),
 CurveItem(mnemonic="Sflu", unit="K/M3", value="", descr="3  BULK DENSITY", original_
↪mnemonic="Sflu", data.shape=(3,)),
 CurveItem(mnemonic="NPHI", unit="V/V", value="", descr="4   NEUTRON POROSITY",␣
↪original_mnemonic="NPHI", data.shape=(3,)),
 CurveItem(mnemonic="SFLU:1", unit="OHMM", value="", descr="5  RXO RESISTIVITY",␣
↪original_mnemonic="SFLU", data.shape=(3,)),
 CurveItem(mnemonic="SFLU:2", unit="OHMM", value="", descr="6  SHALLOW RESISTIVITY",␣
↪original_mnemonic="SFLU", data.shape=(3,)),
```

(continues on next page)

(continued from previous page)

```
 CurveItem(mnemonic="sflu", unit="OHMM", value="", descr="7  MEDIUM RESISTIVITY",␣
↪original_mnemonic="sflu", data.shape=(3,)),
 CurveItem(mnemonic="SfLu", unit="OHMM", value="", descr="8  DEEP RESISTIVITY",␣
↪original_mnemonic="SfLu", data.shape=(3,))
]
```

CHAPTER 5

Data section

5.1 Handling text, dates, timestamps, or any non-numeric characters

By default, lasio will attempt to convert each column of the data section into floating-point numbers. If that fails, as it will for non-numeric characters, then the column will be returned as text (`str`). The behavour can be controlled by specifing the data type as either `int`, `float` or `str` per column using the `dtypes` keyword argument to *`lasio.LASFile.read()`*.

See the example `data_characters.las`:

```
~A TIME        DATE        DEPT ARC_GR_UNC_RT
00:00:00 01-Jan-20  1500.2435         126.56
00:00:01 01-Jan-20  1500.3519         126.56
```

```
>>> import lasio.examples
>>> las = lasio.examples.open("data_characters.las")
>>> las["TIME"]
array(['00:00:00', '00:00:01'], dtype='<U32')
>>> las["DATE"]
array(['01-Jan-20', '01-Jan-20'], dtype='<U32')
>>> las["DEPT"]
array([1500.2435, 1500.3519])
>>> las["ARC_GR_UNC_RT"]
array([126.56, 126.56])
>>> las.df().reset_index().info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2 entries, 0 to 1
Data columns (total 4 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   TIME           2 non-null      object
 1   DATE           2 non-null      object
 2   DEPT           2 non-null      float64
 3   ARC_GR_UNC_RT  2 non-null      float64
```

(continues on next page)

(continued from previous page)

```
dtypes: float64(2), object(2)
memory usage: 192.0+ bytes
```

lasio doesn't yet understand dates and timestamps natively, but you can do these conversions with pandas:

```
>>> las["DATE_DT"] = pd.to_datetime(las["DATE"]).values
```

5.2 Repeated/duplicate curve mnemonics

LAS files don't always have unique mnemonics for each curve, but that makes it difficult to retrieve curves by their mnemonic! lasio handles this by appending `:1`, `:2`, etc. to the end of repeat/duplicate mnemonics. For an example, see a LAS file with this ~C section, with "SFLU" duplicated:

```
~CURVE INFORMATION
#MNEM.UNIT      API CODE      CURVE DESCRIPTION
#---------    -------------   ------------------------------
DEPT.M                      :  1  DEPTH
DT  .US/M                      :  2  SONIC TRANSIT TIME
RHOB.K/M3                   :  3  BULK DENSITY
NPHI.V/V                    :  4   NEUTRON POROSITY
SFLU.OHMM                   :  5  RXO RESISTIVITY
SFLU.OHMM                   :  6  SHALLOW RESISTIVITY
ILM .OHMM                   :  7  MEDIUM RESISTIVITY
ILD .OHMM                   :  8  DEEP RESISTIVITY
```

This is represented in the following way:

```
>>> import lasio.examples
>>> las = lasio.examples.open("mnemonic_duplicate.las")
>>> print(las.curves)
Mnemonic  Unit  Value  Description
--------  ----  -----  -----------
DEPT      M         1  DEPTH
DT        US/M      2  SONIC TRANSIT TIME
RHOB      K/M3      3  BULK DENSITY
NPHI      V/V       4   NEUTRON POROSITY
SFLU:1    OHMM      5  RXO RESISTIVITY
SFLU:2    OHMM      6  SHALLOW RESISTIVITY
ILM       OHMM      7  MEDIUM RESISTIVITY
ILD       OHMM      8  DEEP RESISTIVITY
>>> las["SFLU:1"]
array([123.45, 123.45, 123.45])
>>> las["SFLU:2"]
array([125.45, 125.45, 125.45])
```

Note that the actual mnemonic is not present, to avoid ambiguity about which curve would be expected to be returned:

```
>>> las["SFLU"]
```

```
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
File "C:\devapps\kinverarity\projects\lasio\lasio\las.py", line 661, in __getitem__
    raise KeyError("{} not found in curves ({})".format(key, curve_mnemonics))
KeyError: "SFLU not found in curves (['DEPT', 'DT', 'RHOB', 'NPHI', 'SFLU:1', 'SFLU:2
↪', 'ILM', 'ILD'])"
```

(continues on next page)

(continued from previous page)

Note also that lasio remembers the original mnemonic so that on writing the file out, the original mnemonics are replicated:

```
>>> import sys
>>> las.write(sys.stdout)
...
~Curve Information ----------------------------------------
DEPT.M     : 1  DEPTH
DT  .US/M  : 2  SONIC TRANSIT TIME
RHOB.K/M3  : 3  BULK DENSITY
NPHI.V/V   : 4   NEUTRON POROSITY
SFLU.OHMM  : 5  RXO RESISTIVITY
SFLU.OHMM  : 6  SHALLOW RESISTIVITY
ILM .OHMM  : 7  MEDIUM RESISTIVITY
ILD .OHMM  : 8  DEEP RESISTIVITY
...
```

5.3 Ignoring commented-out lines

Sometimes data sections have comment line inside them. By default lasio will ignore any lines starting with the "#" character within the data section. You can control this using the `remove_data_line_filter='#'` argument to *`lasio.LASFile.read()`*.

5.4 Ignoring the data section

Lasio can ignore the data section by setting ignore_data to true: `lasio.read(file, ignore_date=True)`

This will completely skip reading the data section and the returned object will just contain the header metadata section.

A quick way to see the expected column names is: `lasio.read(file, ignore_data=True).keys()`

To re-run without ignore_data: `lasio.read(file).keys()`

If this returns a different set of columns then there may be a data parsing error. In this case, if incorrect parsing causes lasio to create extra columns they will be named 'UKNOWN:1', 'UNKNOWN:2', 'UNKNOWN:<n>'... This can usually be fixed by tuning lasio.read()'s read_policy or null_policy options.

5.5 Handling errors with read_policy and null_policy

lasio has a flexible way of handling "errors" in the ~ASCII data section to accommodate how strict or flexible you want to be. The two main tools are `read_policy` and `null_policy`. These are optional arguments to *`lasio.LASFile.read()`*. Each defaults to common options which can be overridden either by other pre-set options or by a list of specific options. These policy settings are configured in `lasio/defaults.py`.

By default, `lasio.read(f)` runs as if explicitly set to `lasio.read(f, read_policy='default', null_policy='common')`.

5.5.1 Examples of policy override syntax

Change only read_policy with one of the builtin policy sets: `lasio.read(f, read_policy='comma-delimiter')`

Change only null_policy with one of the builtin policy sets: `lasio.read(f, null_policy='aggressive')`

Change both read_policy and null_policy with builtin policies: `lasio.read(f, read_policy='comma-delimiter', null_policy='none')`

Change read_policy with specific policies (found in defaults.py): `lasio.read(f, read_policy=["comma-decimal-mark", "run-on(.)"])`

Change null_policy with your own hard-coded options: `lasio.read(f, null_policy=["9999.25", "999.25", "NA", "INF", "IO", "IND"])`

5.5.2 Example errors

Here are some examples of errors.

- Files could contain a variety of indicators for an invalid data point other than that defined by the NULL line in the LAS header (usually -999.25).
- Fixed-width columns could run into each other:

```
7686.500    64.932     0.123     0.395    12.403   156.271    10.649    -0.005   193.
↪223   327.902    -0.023     4.491     2.074    29.652
7686.000    67.354     0.140     0.415     9.207  4648.011    10.609    -0.004  3778.
↪709  1893.751    -0.048     4.513     2.041   291.910
7685.500    69.004     0.151     0.412     7.020101130.188    10.560    -0.004 60000.
↪000  2901.317    -0.047     4.492     2.046   310.119
7685.000    68.809     0.150     0.411     7.330109508.961    10.424    -0.005 60000.
↪000  2846.619    -0.042     4.538     2.049   376.968
7684.500    68.633     0.149     0.402     7.345116238.453    10.515    -0.005 60000.
↪000  2290.275    -0.051     4.543     2.063   404.972
7684.000    68.008     0.144     0.386     7.682  4182.679    10.515    -0.004  3085.
↪681  1545.842    -0.046     4.484     2.089   438.195
```

- Odd text such as `(null)`:

```
8090.00         -999.25         -999.25         -999.25               0
↪0               0               0               0              0               0
↪            0
8091.000          0.70          337.70          (null)              0
↪0               0               0               0              0               0
↪            0
8092.000        -999.25         -999.25         -999.25               0
↪0               0               0               0              0               0
↪          0
```

5.5.3 Handling run-on errors

lasio detects and handles these problems by default using `lasio.read(f, read_policy='default')`. For example a file with this data section:

```
~A
    7686.000    67.354     0.140     0.415     9.207  4648.011    10.609
```

(continues on next page)

(continued from previous page)

```
    7685.500    69.004    0.151    0.412    7.020101130.188    10.560
    7685.000    68.809    0.150    0.411    7.330-19508.961    10.424
    7684.500    68.633    0.149    0.402    7.345116238.453    10.515
    7684.000    68.008    0.144    0.386    7.682  4182.679    10.515
```

is loaded by default as the following:

```
>>> import lasio.examples
>>> las = lasio.examples.open('null_policy_runon.las')
>>> las.data
array([[7686.0, 67.354, 0.14, 0.415, 9.207, 4648.011, 10.609],
       [7685.5, 69.004, 0.151, 0.412, nan, nan, 10.56],
       [7685.0, 68.809, 0.15, 0.411, 7.33, -19508.961, 10.424],
       [7684.5, 68.633, 0.149, 0.402, nan, nan, 10.515],
       [7684.0, 68.008, 0.144, 0.386, 7.682, 4182.679, 10.515]])
```

5.5.4 Handling invalid data indicators automatically

These are detected by lasio to a degree which you can control with the null_policy keyword argument.

You can specify a policy of 'none', 'strict', 'common', 'aggressive', or 'all'. These policies all include a subset of pre-defined substitutions. Or you can give your own list of substitutions. Here is the list of predefined policies and substitutions from *lasio.defaults*.

Policies that you can pick with e.g. `null_policy='common'`:

```
NULL_POLICIES = {
    'none': [],
    'strict': ['NULL', ],
    'common': ['NULL', '(null)', '-',
               '9999.25', '999.25', 'NA', 'INF', 'IO', 'IND'],
    'aggressive': ['NULL', '(null)', '--',
                   '9999.25', '999.25', 'NA', 'INF', 'IO', 'IND',
                   '999', '999.99', '9999', '9999.99' '2147483647', '32767',
                   '-0.0', ],
    'all': ['NULL', '(null)', '-',
            '9999.25', '999.25', 'NA', 'INF', 'IO', 'IND',
            '999', '999.99', '9999', '9999.99' '2147483647', '32767', '-0.0',
            'numbers-only', ],
    'numbers-only': ['numbers-only', ]
    }
```

Or substitutions you could specify with e.g. `null_policy=['NULL', '999.25', 'INF']`:

```
NULL_SUBS = {
    'NULL': [None, ],                          # special case to be handled
    '999.25': [-999.25, 999.25],
    '9999.25': [-9999.25, 9999.25],
    '999.99': [-999.99, 999.99],
    '9999.99': [-9999.99, 9999.99],
    '999': [-999, 999],
    '9999': [-9999, 9999],
    '2147483647': [-2147483647, 2147483647],
    '32767': [-32767, 32767],
    'NA': [(re.compile(r'(#N/A)[ ]'), ' NaN '),
```

(continues on next page)

(continued from previous page)

```
            (re.compile(r'[ ](#N/A)'), ' NaN '), ],
    'INF': [(re.compile(r'(-?1\.#INF)[ ]'), ' NaN '),
            (re.compile(r'[ ](-?1\.#INF)'), ' NaN '), ],
    'IO': [(re.compile(r'(-?1\.#IO)[ ]'), ' NaN '),
            (re.compile(r'[ ](-?1\.#IO)'), ' NaN '), ],
    'IND': [(re.compile(r'(-?1\.#IND)[ ]'), ' NaN '),
            (re.compile(r'[ ](-?1\.#IND)'), ' NaN '), ],
    '-0.0': [(re.compile(r'(-?0\.0+)[ ]'), ' NaN '),
             (re.compile(r'[ ](-?0\.0+)'), ' NaN '), ],
    'numbers-only': [(re.compile(r'([^ 0-9.\-+]+)[ ]'), ' NaN '),
                     (re.compile(r'[ ]([^ 0-9.\-+]+)'), ' NaN '), ],
    }
```

You can also specify substitutions directly. E.g. for a file with this data section:

```
~A  DEPTH     DT        RHOB      NPHI      SFLU      SFLA       ILM       ILD
1670.000    9998  2550.000    0.450  123.450  123.450  110.200  105.600
1669.875    9999  2550.000    0.450  123.450  123.450  110.200  105.600
1669.750   10000       ERR    0.450  123.450  -999.25  110.200  105.600
```

By default, it will read all data as a string due to the presence of "ERR":

```
>>> las = lasio.examples.open('null_policy_ERR.las')
>>> las.data
array([['1670.0', '9998.0', '2550.0', '0.45', '123.45', '123.45',
        '110.2', '105.6'],
       ['1669.875', '9999.0', '2550.0', '0.45', '123.45', '123.45',
        '110.2', '105.6'],
       ['1669.75', '10000.0', 'ERR', '0.45', '123.45', '-999.25',
        '110.2', '105.6']], dtype='<U32')
```

We can fix it by using an explicit NULL policy.

```
>>> las = lasio.examples.open('null_policy_ERR.las', null_policy=[('ERR', ' NaN ')])
>>> las.data
array([[ 1.670000e+03,  9.998000e+03,  2.550000e+03,  4.500000e-01,
         1.234500e+02,  1.234500e+02,  1.102000e+02,  1.056000e+02],
       [ 1.669875e+03,  9.999000e+03,  2.550000e+03,  4.500000e-01,
         1.234500e+02,  1.234500e+02,  1.102000e+02,  1.056000e+02],
       [ 1.669750e+03,  1.000000e+04,           nan,  4.500000e-01,
         1.234500e+02, -9.992500e+02,  1.102000e+02,  1.056000e+02]])
```

See `tests/test_null_policy.py` (link) for some examples.

CHAPTER 6

Writing LAS files

Any LASFile object can be written to a new LAS file using the *lasio.LASFile.write()* method.

6.1 Converting between v1.2 and v2.0

Take this sample LAS 2.0 file:

```
~VERSION INFORMATION
 VERS.                          2.0 :   CWLS LOG ASCII STANDARD -VERSION 2.0
 WRAP.                          NO  :   ONE LINE PER DEPTH STEP
~WELL INFORMATION
#MNEM.UNIT              DATA                        DESCRIPTION
#----- -----            ----------                 -------------------------
STRT    .M              1670.0000                  :START DEPTH
STOP    .M              1660.0000                  :STOP DEPTH
STEP    .M              -0.1250                    :STEP
NULL    .               -999.25                    :NULL VALUE
COMP    .       ANY OIL COMPANY INC.               :COMPANY
WELL    .       AAAAA_2             :WELL
FLD     .       WILDCAT                            :FIELD
LOC     .       12-34-12-34W5M                     :LOCATION
PROV    .       ALBERTA                            :PROVINCE
SRVC    .       ANY LOGGING COMPANY INC.           :SERVICE COMPANY
DATE    .       13-DEC-86                          :LOG DATE
UWI     .       100123401234W500                   :UNIQUE WELL ID
~CURVE INFORMATION
#MNEM.UNIT              API CODES                     CURVE DESCRIPTION
#------------------     ------------                 -------------------------
 DEPT   .M                                         :  1  DEPTH
 DT     .US/M           60 520 32 00               :  2  SONIC TRANSIT TIME
 RHOB   .K/M3           45 350 01 00               :  3  BULK DENSITY
 NPHI   .V/V            42 890 00 00               :  4  NEUTRON POROSITY
 SFLU   .OHMM           07 220 04 00               :  5  SHALLOW RESISTIVITY
```

(continues on next page)

(continued from previous page)

```
 SFLA    .OHMM           07 222 01 00             :  6  SHALLOW RESISTIVITY
 ILM     .OHMM           07 120 44 00             :  7  MEDIUM RESISTIVITY
 ILD     .OHMM           07 120 46 00             :  8  DEEP RESISTIVITY
~PARAMETER INFORMATION
#MNEM.UNIT              VALUE             DESCRIPTION
#--------------     ----------------      ------------------------------------------
 MUD     .               GEL CHEM        :   MUD TYPE
 BHT     .DEGC           35.5000         :   BOTTOM HOLE TEMPERATURE
 BS      .MM             200.0000        :   BIT SIZE
 FD      .K/M3           1000.0000       :   FLUID DENSITY
 MATR    .               SAND            :   NEUTRON MATRIX
 MDEN    .               2710.0000       :   LOGGING MATRIX DENSITY
 RMF     .OHMM           0.2160          :   MUD FILTRATE RESISTIVITY
 DFD     .K/M3           1525.0000       :   DRILL FLUID DENSITY
~OTHER
     Note: The logging tools became stuck at 625 metres causing the data
     between 625 metres and 615 metres to be invalid.
~A  DEPTH     DT    RHOB        NPHI   SFLU    SFLA      ILM      ILD
1670.000   123.450 2550.000    0.450  123.450  123.450  110.200  105.600
1669.875   123.450 2550.000    0.450  123.450  123.450  110.200  105.600
1669.750   123.450 2550.000    0.450  123.450  123.450  110.200  105.600
```

And we can use lasio to convert it to LAS 1.2:

```
>>> las = lasio.examples.open("2.0/sample_2.0.las")
>>> las.write('example-as-v1.2.las', version=1.2)
```

```
~Version ------------------------------------------------
VERS. 1.2 : CWLS LOG ASCII STANDARD - VERSION 1.2
WRAP.  NO : ONE LINE PER DEPTH STEP
~Well ---------------------------------------------------
STRT.M         1670.0 : START DEPTH
STOP.M        1669.75 : STOP DEPTH
STEP.M         -0.125 : STEP
NULL.         -999.25 : NULL VALUE
COMP.         COMPANY : ANY OIL COMPANY INC.
WELL.            WELL : AAAAA_2
FLD .           FIELD : WILDCAT
LOC .        LOCATION : 12-34-12-34W5M
PROV.        PROVINCE : ALBERTA
SRVC. SERVICE COMPANY : ANY LOGGING COMPANY INC.
DATE.        LOG DATE : 13-DEC-86
UWI .  UNIQUE WELL ID : 100123401234W500
~Curves -------------------------------------------------
DEPT.M                 : 1  DEPTH
DT  .US/M 60 520 32 00 : 2  SONIC TRANSIT TIME
RHOB.K/M3 45 350 01 00 : 3  BULK DENSITY
NPHI.V/V  42 890 00 00 : 4  NEUTRON POROSITY
SFLU.OHMM 07 220 04 00 : 5  SHALLOW RESISTIVITY
SFLA.OHMM 07 222 01 00 : 6  SHALLOW RESISTIVITY
ILM .OHMM 07 120 44 00 : 7  MEDIUM RESISTIVITY
ILD .OHMM 07 120 46 00 : 8  DEEP RESISTIVITY
~Params -------------------------------------------------
MUD .   GEL CHEM : MUD TYPE
BHT .DEGC   35.5 : BOTTOM HOLE TEMPERATURE
```

(continues on next page)

(continued from previous page)

```
BS  .MM    200.0 : BIT SIZE
FD  .K/M3 1000.0 : FLUID DENSITY
MATR.       SAND : NEUTRON MATRIX
MDEN.     2710.0 : LOGGING MATRIX DENSITY
RMF .OHMM  0.216 : MUD FILTRATE RESISTIVITY
DFD .K/M3 1525.0 : DRILL FLUID DENSITY
~Other ------------------------------------------------
Note: The logging tools became stuck at 625 metres causing the data
between 625 metres and 615 metres to be invalid.
~ASCII -------------------------------------------------
       1670     123.45       2550       0.45     123.45     123.45      110.2
↪105.6
     1669.9     123.45       2550       0.45     123.45     123.45      110.2
↪105.6
     1669.8     123.45       2550       0.45     123.45     123.45      110.2
↪105.6
```

6.2 Converting between wrapped/unwrapped

Here is an example using this file to convert a wrapped data section to unwrapped.

```
~Version Information
 VERS.                 1.20:   CWLS log ASCII Standard -VERSION 1.20
 WRAP.                  YES:   Multiple lines per depth step
~Well Information
#MNEM.UNIT       Data Type    Information
#---------    -------------   ------------------------------
 STRT.M            910.000:
 STOP.M            901.000:
 STEP.M            -0.1250:
 NULL.           -999.2500:   Null value
 COMP.             COMPANY:   ANY OIL COMPANY INC.
 WELL.                WELL:   ANY ET AL XX-XX-XX-XX
 FLD .               FIELD:   WILDCAT
 LOC .            LOCATION:   XX-XX-XX-XXW3M
 PROV.            PROVINCE:   SASKATCHEWAN
 SRVC.     SERVICE COMPANY:   ANY LOGGING COMPANY INC.
 SON .     SERVICE ORDER :   142085
 DATE.            LOG DATE:   13-DEC-86
 UWI .      UNIQUE WELL ID:
~Curve Information
#MNEM.UNIT      API CODE      Curve Description
#---------    -------------   ------------------------------
 DEPT.M                      :    Depth
 DT  .US/M                   :  1 Sonic Travel Time
 RHOB.K/M                    :  2 Density-Bulk Density
 NPHI.V/V                    :  3 Porosity -Neutron
 RX0 .OHMM                   :  4 Resistivity -Rxo
 RESS.OHMM                   :  5 Resistivity -Shallow
 RESM.OHMM                   :  6 Resistivity -Medium
 RESD.OHMM                   :  7 Resistivity -Deep
 SP  .MV                     :  8 Spon. Potential
 GR  .GAPI                   :  9 Gamma Ray
 CALI.MM                     : 10 Caliper
```

(continues on next page)

(continued from previous page)

```
 DRHO.K/M3                      : 11 Delta-Rho
 EATT.DBM                       : 12 EPT Attenuation
 TPL .NS/M                      : 13 TP -EPT
 PEF .                          : 14 PhotoElectric Factor
 FFI .V/V                       : 15 Porosity -NML FFI
 DCAL.MM                        : 16 Caliper-Differential
 RHGF.K/M3                      : 17 Density-Formation
 RHGA.K/M3                      : 18 Density-Apparent
 SPBL.MV                        : 19 Baselined SP
 GRC .GAPI                      : 20 Gamma Ray BHC
 PHIA.V/V                       : 21 Porosity -Apparent
 PHID.V/V                       : 22 Porosity -Density
 PHIE.V/V                       : 23 Porosity -Effective
 PHIN.V/V                       : 24 Porosity -Neut BHC
 PHIC.V/V                       : 25 Porosity -Total HCC
 R0  .OHMM                      : 26 Ro
 RWA .OHMM                      : 27 Rfa
 SW   .                         : 28 Sw -Effective
 MSI .                          : 29 Sh Idx -Min
 BVW .                          : 30 BVW
 FGAS.                          : 31 Flag -Gas Index
 PIDX.                          : 32 Prod Idx
 FBH .                          : 33 Flag -Bad Hole
 FHCC.                          : 34 Flag -HC Correction
 LSWB.                          : 35 Flag -Limit SWB
~A Log data section
910.000000
  -999.2500  2692.7075     0.3140    19.4086    19.4086    13.1709    12.2681
    -1.5010    96.5306   204.7177    30.5822  -999.2500  -999.2500     3.2515
  -999.2500     4.7177  3025.0264  3025.0264    -1.5010    93.1378     0.1641
     0.0101     0.1641     0.3140     0.1641    11.1397     0.3304     0.9529
     0.0000     0.1564     0.0000    11.1397     0.0000     0.0000     0.0000
909.875000
  -999.2500  2712.6460     0.2886    23.3987    23.3987    13.6129    12.4744
    -1.4720    90.2803   203.1093    18.7566  -999.2500  -999.2500     3.7058
  -999.2500     3.1093  3004.6050  3004.6050    -1.4720    86.9078     0.1456
    -0.0015     0.1456     0.2886     0.1456    14.1428     0.2646     1.0000
     0.0000     0.1456     0.0000    14.1428     0.0000     0.0000     0.0000
909.750000
  -999.2500  2692.8137     0.2730    22.5909    22.5909    13.6821    12.6146
    -1.4804    89.8492   201.9287     3.1551  -999.2500  -999.2500     4.3124
  -999.2500     1.9287  2976.4451  2976.4451    -1.4804    86.3465     0.1435
     0.0101     0.1435     0.2730     0.1435    14.5674     0.2598     1.0000
     0.0000     0.1435     0.0000    14.5674     0.0000     0.0000     0.0000
909.625000
  -999.2500  2644.3650     0.2765    18.4831    18.4831    13.4159    12.6900
    -1.5010    93.3999   201.5826    -6.5861  -999.2500  -999.2500     4.3822
  -999.2500     1.5826  2955.3528  2955.3528    -1.5010    89.7142     0.1590
     0.0384     0.1590     0.2765     0.1590    11.8600     0.3210     0.9667
     0.0000     0.1538     0.0000    11.8600     0.0000     0.0000     0.0000
909.500000
  -999.2500  2586.2822     0.2996    13.9187    13.9187    12.9195    12.7016
    -1.4916    98.1214   201.7126    -4.5574  -999.2500  -999.2500     3.5967
  -999.2500     1.7126  2953.5940  2953.5940    -1.4916    94.2670     0.1880
     0.0723     0.1880     0.2996     0.1880     8.4863     0.4490     0.8174
     0.0000     0.1537     0.0000     8.4863     0.0000     0.0000     0.0000
```

We will change the wrap by adjusting the relevant header section in the LASFile header:

```
>>> las.version
[HeaderItem(mnemonic="VERS", unit="", value="1.2", descr="CWLS log ASCII Standa"),
 HeaderItem(mnemonic="WRAP", unit="", value="YES", descr="Multiple lines per de")]

>>> las.version.WRAP = 'NO'
>>> las.version.WRAP
HeaderItem(mnemonic="WRAP", unit="", value="NO", descr="Multiple lines per dep")
>>> las.write('example-unwrapped.las')
WARNING:lasio.writer:[v1.2] line #58 has 396 chars (>256)
WARNING:lasio.writer:[v1.2] line #59 has 396 chars (>256)
WARNING:lasio.writer:[v1.2] line #60 has 396 chars (>256)
WARNING:lasio.writer:[v1.2] line #61 has 396 chars (>256)
WARNING:lasio.writer:[v1.2] line #62 has 396 chars (>256)
```

We get warnings because the LAS 1.2 standard doesn't allow writing lines longer than 256 characters. lasio provides the warning but still produces the long lines:

```
~Version ------------------------------------------------
VERS. 1.2 : CWLS LOG ASCII STANDARD - VERSION 1.2
WRAP.  NO : Multiple lines per depth step
~Well ---------------------------------------------------
STRT.M          910.0 :
STOP.M          909.5 :
STEP.M         -0.125 :
NULL.         -999.25 : Null value
COMP.         COMPANY : ANY OIL COMPANY INC.
WELL.            WELL : ANY ET AL XX-XX-XX-XX
FLD .           FIELD : WILDCAT
LOC .        LOCATION : XX-XX-XX-XXW3M
PROV.        PROVINCE : SASKATCHEWAN
SRVC. SERVICE COMPANY : ANY LOGGING COMPANY INC.
SON .   SERVICE ORDER : 142085
DATE.        LOG DATE : 13-DEC-86
UWI .  UNIQUE WELL ID :
~Curves -------------------------------------------------
DEPT.M     : Depth
DT  .US/M  : 1 Sonic Travel Time
RHOB.K/M   : 2 Density-Bulk Density
NPHI.V/V   : 3 Porosity -Neutron
RX0 .OHMM  : 4 Resistivity -Rxo
RESS.OHMM  : 5 Resistivity -Shallow
RESM.OHMM  : 6 Resistivity -Medium
RESD.OHMM  : 7 Resistivity -Deep
SP  .MV    : 8 Spon. Potential
GR  .GAPI  : 9 Gamma Ray
CALI.MM    : 10 Caliper
DRHO.K/M3  : 11 Delta-Rho
EATT.DBM   : 12 EPT Attenuation
TPL .NS/M  : 13 TP -EPT
PEF .      : 14 PhotoElectric Factor
FFI .V/V   : 15 Porosity -NML FFI
DCAL.MM    : 16 Caliper-Differential
RHGF.K/M3  : 17 Density-Formation
RHGA.K/M3  : 18 Density-Apparent
SPBL.MV    : 19 Baselined SP
GRC .GAPI  : 20 Gamma Ray BHC
```

(continues on next page)

(continued from previous page)

```
PHIA.V/V   : 21 Porosity -Apparent
PHID.V/V   : 22 Porosity -Density
PHIE.V/V   : 23 Porosity -Effective
PHIN.V/V   : 24 Porosity -Neut BHC
PHIC.V/V   : 25 Porosity -Total HCC
R0  .OHMM  : 26 Ro
RWA .OHMM  : 27 Rfa
SW  .      : 28 Sw -Effective
MSI .      : 29 Sh Idx -Min
BVW .      : 30 BVW
FGAS.      : 31 Flag -Gas Index
PIDX.      : 32 Prod Idx
FBH .      : 33 Flag -Bad Hole
FHCC.      : 34 Flag -HC Correction
LSWB.      : 35 Flag -Limit SWB
~Params ----------------------------------------------------
~Other -----------------------------------------------------
~ASCII -----------------------------------------------------
        910    -999.25     2692.7      0.314     19.409     19.409     13.171     12.
↪268     -1.501     96.531     204.72     30.582    -999.25    -999.25     3.2515
↪-999.25     4.7177       3025       3025     -1.501     93.138     0.1641      0.
↪0101     0.1641      0.314     0.1641      11.14     0.3304     0.9529          0
↪  0.1564          0      11.14          0          0          0
     909.88    -999.25     2712.6     0.2886     23.399     23.399     13.613     12.
↪474     -1.472      90.28     203.11     18.757    -999.25    -999.25     3.7058
↪-999.25     3.1093     3004.6     3004.6     -1.472     86.908     0.1456    -0.
↪0015     0.1456     0.2886     0.1456     14.143     0.2646          1          0
↪  0.1456          0     14.143          0          0          0
     909.75    -999.25     2692.8      0.273     22.591     22.591     13.682     12.
↪615    -1.4804     89.849     201.93     3.1551    -999.25    -999.25     4.3124
↪-999.25     1.9287     2976.4     2976.4    -1.4804     86.347     0.1435      0.
↪0101     0.1435      0.273     0.1435     14.567     0.2598          1          0
↪  0.1435          0     14.567          0          0          0
     909.62    -999.25     2644.4     0.2765     18.483     18.483     13.416      12.
↪69     -1.501       93.4     201.58    -6.5861    -999.25    -999.25     4.3822     -
↪999.25     1.5826     2955.4     2955.4     -1.501     89.714      0.159     0.0384
↪     0.159     0.2765      0.159      11.86      0.321     0.9667          0     0.
↪1538          0      11.86          0          0          0
      909.5    -999.25     2586.3     0.2996     13.919     13.919     12.919     12.
↪702    -1.4916     98.121     201.71    -4.5574    -999.25    -999.25     3.5967
↪-999.25     1.7126     2953.6     2953.6    -1.4916     94.267      0.188      0.
↪0723      0.188     0.2996      0.188     8.4863      0.449     0.8174          0
↪  0.1537          0     8.4863          0          0          0
```

If we decide to write the file in LAS 2.0 format, the warnings will go away:

```
>>> las.write('example-version-2.0.las', version=2.0)
```

```
~Version ---------------------------------------------------
VERS. 2.0 : CWLS log ASCII Standard -VERSION 2.0
WRAP.  NO : Multiple lines per depth step
~Well ------------------------------------------------------
STRT.M                  910.0 :
STOP.M                  909.5 :
STEP.M                 -0.125 :
NULL.                -999.25 : Null value
```

(continues on next page)

(continued from previous page)

```
COMP.     ANY OIL COMPANY INC. : COMPANY
WELL.    ANY ET AL XX-XX-XX-XX : WELL
FLD .                  WILDCAT : FIELD
LOC .           XX-XX-XX-XXW3M : LOCATION
PROV.             SASKATCHEWAN : PROVINCE
SRVC. ANY LOGGING COMPANY INC. : SERVICE COMPANY
SON .                   142085 : SERVICE ORDER
DATE.                13-DEC-86 : LOG DATE
UWI .                          : UNIQUE WELL ID
~Curves -----------------------------------------------------
DEPT.M     : Depth
DT  .US/M  : 1 Sonic Travel Time
RHOB.K/M   : 2 Density-Bulk Density
NPHI.V/V   : 3 Porosity -Neutron
RX0 .OHMM  : 4 Resistivity -Rxo
RESS.OHMM  : 5 Resistivity -Shallow
RESM.OHMM  : 6 Resistivity -Medium
RESD.OHMM  : 7 Resistivity -Deep
SP  .MV    : 8 Spon. Potential
GR  .GAPI  : 9 Gamma Ray
CALI.MM    : 10 Caliper
DRHO.K/M3  : 11 Delta-Rho
EATT.DBM   : 12 EPT Attenuation
TPL .NS/M  : 13 TP -EPT
PEF .      : 14 PhotoElectric Factor
FFI .V/V   : 15 Porosity -NML FFI
DCAL.MM    : 16 Caliper-Differential
RHGF.K/M3  : 17 Density-Formation
RHGA.K/M3  : 18 Density-Apparent
SPBL.MV    : 19 Baselined SP
GRC .GAPI  : 20 Gamma Ray BHC
PHIA.V/V   : 21 Porosity -Apparent
PHID.V/V   : 22 Porosity -Density
PHIE.V/V   : 23 Porosity -Effective
PHIN.V/V   : 24 Porosity -Neut BHC
PHIC.V/V   : 25 Porosity -Total HCC
R0  .OHMM  : 26 Ro
RWA .OHMM  : 27 Rfa
SW  .      : 28 Sw -Effective
MSI .      : 29 Sh Idx -Min
BVW .      : 30 BVW
FGAS.      : 31 Flag -Gas Index
PIDX.      : 32 Prod Idx
FBH .      : 33 Flag -Bad Hole
FHCC.      : 34 Flag -HC Correction
LSWB.      : 35 Flag -Limit SWB
~Params -----------------------------------------------------
~Other ------------------------------------------------------
~ASCII ------------------------------------------------------
        910    -999.25     2692.7      0.314     19.409     19.409     13.171     12.
↪268     -1.501     96.531     204.72     30.582    -999.25    -999.25     3.2515
↪-999.25     4.7177       3025       3025     -1.501     93.138     0.1641     0.
↪0101     0.1641      0.314     0.1641      11.14     0.3304     0.9529          0
↪  0.1564          0      11.14          0          0          0
     909.88    -999.25     2712.6     0.2886     23.399     23.399     13.613     12.
↪474     -1.472      90.28     203.11     18.757    -999.25    -999.25     3.7058
↪-999.25     3.1093     3004.6     3004.6     -1.472     86.908     0.1456    -0.
↪0015     0.1456     0.2886     0.1456     14.143     0.2646          1          0
↪  0.1456          0     14.143          0          0          0
```

(continues on next page)

(continued from previous page)

```
60      909.75     -999.25      2692.8       0.273      22.591      22.591      13.682      12.
↪615     -1.4804      89.849      201.93      3.1551     -999.25     -999.25      4.3124
↪-999.25      1.9287      2976.4      2976.4     -1.4804      86.347      0.1435      0.
↪0101      0.1435       0.273      0.1435      14.567      0.2598           1           0
↪   0.1435           0      14.567           0           0           0
61      909.62     -999.25      2644.4      0.2765      18.483      18.483      13.416      12.
↪69      -1.501        93.4      201.58     -6.5861     -999.25     -999.25      4.3822     -
↪999.25      1.5826      2955.4      2955.4      -1.501      89.714       0.159      0.0384
↪      0.159      0.2765       0.159       11.86       0.321      0.9667           0      0.
↪1538           0       11.86           0           0           0
62      909.5     -999.25      2586.3      0.2996      13.919      13.919      12.919      12.
↪702     -1.4916      98.121      201.71     -4.5574     -999.25     -999.25      3.5967
↪-999.25      1.7126      2953.6      2953.6     -1.4916      94.267       0.188      0.
↪0723       0.188      0.2996       0.188      8.4863       0.449      0.8174           0
↪   0.1537           0      8.4863           0           0           0
```

6.3 Formatting data section columns

The keyword parameters that control the column spacing in the data section are, the left-hand spacer, **lhs_spacer**, and the in-between column spacer, **spacer**. They are both set to one space by default.

Use the **len_numeric_field** parameter to configure the padding within the numeric data fields.

The following examples will use lasio/tests/examples/2.0/sample_2.0.las. It's data section looks like this:

```
~A  DEPTH      DT     RHOB         NPHI    SFLU     SFLA       ILM       ILD
1670.000   123.450 2550.000     0.450  123.450  123.450  110.200  105.600
1669.875   123.450 2550.000     0.450  123.450  123.450  110.200  105.600
1669.750   123.450 2550.000     0.450  123.450  123.450  110.200  105.600
```

6.3.1 Default data section formatting

If this file is read in and then written, the data section is formatted like this by default:

```
import lasio.examples
from lasio.reader import StringIO

las = lasio.examples.open("2.0/sample_2.0.las")
s = StringIO()

las.write(s)
s.seek(1665)
print(s.read())
```

```
~ASCII ------------------------------------------------
 1670.00000  123.45000 2550.00000    0.45000  123.45000  123.45000  110.20000  105.
↪60000
 1669.87500  123.45000 2550.00000    0.45000  123.45000  123.45000  110.20000  105.
↪60000
 1669.75000  123.45000 2550.00000    0.45000  123.45000  123.45000  110.20000  105.
↪60000
```

The default settings are:

- len_numeric_field defaults to 10 characters
  - 5 digits to the right of the decimal
  - 1 character for the decimal
  - 4 digits for the number to the left of the decimal
    - if there are less than 4 digits, the field is padded with blank spaces
    - if there are more than 4 digits, the field is expanded to include all the digits
- lhs_spacer defaults to 1 space. So the data is indented by one space.
- spacer defaults to 1 space. So data columns will have one space dividing them
  - if a number is padded with blanks there will be more spaces seen for that number's field

6.3.2 Examples: len_numeric_field

Turn off data column left-padding, set len_numeric_field to -1

This removes the padding of the numeric fields and leaves the lhs_spacer and spacer defaults of one space columns.

```
remove_padding=-1
las.write(s, len_numeric_field=remove_padding)
s.seek(1665)
print(s.read())
```

```
~ASCII ------------------------------------------------
 1670.00000 123.45000 2550.00000 0.45000 123.45000 123.45000 110.20000 105.60000
 1669.87500 123.45000 2550.00000 0.45000 123.45000 123.45000 110.20000 105.60000
 1669.75000 123.45000 2550.00000 0.45000 123.45000 123.45000 110.20000 105.60000
```

Set column width to less than the default: > 0 & < 10

Note in this example that only column 4 is 8 characters wide the other columns are 9 or more characters and expand to fit all their characters.

```
col_width = 8
las.write(s, len_numeric_field=col_width)
s.seek(1665)
print(s.read())
```

```
~ASCII ------------------------------------------------
 1670.00000 123.45000 2550.00000  0.45000 123.45000 123.45000 110.20000 105.60000
 1669.87500 123.45000 2550.00000  0.45000 123.45000 123.45000 110.20000 105.60000
 1669.75000 123.45000 2550.00000  0.45000 123.45000 123.45000 110.20000 105.60000
```

Set column width to more than the default: > 10

In this example all the columns are padded with space to make them wider. The lhs_spacer, left hand spacer, is still one space wide. The additional space on the left hand side is from the padding of the first data column to the requested col_width of 12 characters.

```
col_width = 12
las.write(s, len_numeric_field=col_width)
s.seek(1665)
print(s.read())
```

```
~ASCII -----------------------------------------------------
   1670.00000    123.45000   2550.00000      0.45000    123.45000    123.45000    110.
↪20000    105.60000
   1669.87500    123.45000   2550.00000      0.45000    123.45000    123.45000    110.
↪20000    105.60000
   1669.75000    123.45000   2550.00000      0.45000    123.45000    123.45000    110.
↪20000    105.60000
```

6.3.3 Examples: lhs_spacer

Remove the left most space, set lhs_spacer to an empty string

The output here removes the default 1 space column from the left hand side. Otherwise, it is the same as the initial default example.

```
empty_space = ""
las.write(s, lhs_spacer=empty_space)
s.seek(1665)
print(s.read())
```

```
~ASCII -----------------------------------------------------
1670.00000  123.45000 2550.00000    0.45000  123.45000  123.45000  110.20000  105.
↪60000
1669.87500  123.45000 2550.00000    0.45000  123.45000  123.45000  110.20000  105.
↪60000
1669.75000  123.45000 2550.00000    0.45000  123.45000  123.45000  110.20000  105.
↪60000
```

Increase the left hand space, set lhs_spacer to a string with more spaces

In this example, there are 3 columns of space at the left hand side.

```
three_spaces = "   "
las.write(s, lhs_spacer=three_spaces)
s.seek(1665)
print(s.read())
```

```
~ASCII -----------------------------------------------------
   1670.00000  123.45000 2550.00000    0.45000  123.45000  123.45000  110.20000  105.
↪60000
   1669.87500  123.45000 2550.00000    0.45000  123.45000  123.45000  110.20000  105.
↪60000
   1669.75000  123.45000 2550.00000    0.45000  123.45000  123.45000  110.20000  105.
↪60000
```

6.3.4 Examples: spacer

Increase the space between columns, set spacer to as string with more spaces

In this example, there are 3 columns of space separating the data columns from each other. In addition some of the columns have more space due to space-padding of their digits to the right of the decimal.

Note that the left hand side only has the 1 space, because it is not in between the columns and is set by the default lhs_spacer setting of one space.

```
three_spaces = "   "
las.write(s, spacer=three_spaces)
s.seek(1665)
print(s.read())
```

```
~ASCII ------------------------------------------------
 1670.00000    123.45000   2550.00000      0.45000    123.45000    123.45000    110.
↪20000    105.60000
 1669.87500    123.45000   2550.00000      0.45000    123.45000    123.45000    110.
↪20000    105.60000
 1669.75000    123.45000   2550.00000      0.45000    123.45000    123.45000    110.
↪20000    105.60000
```

Use a different character as the spacer character

This example demonstrates using a comma as the column separator. This outputs a set of comma separated data values.

```
comma_spacer = ","
las.write(s, spacer=comma_spacer)
s.seek(1665)
print(s.read())
```

```
~ASCII ------------------------------------------------
 1670.00000, 123.45000,2550.00000,   0.45000, 123.45000, 123.45000, 110.20000, 105.
↪60000
 1669.87500, 123.45000,2550.00000,   0.45000, 123.45000, 123.45000, 110.20000, 105.
↪60000
 1669.75000, 123.45000,2550.00000,   0.45000, 123.45000, 123.45000, 110.20000, 105.
↪60000
```

6.3.5 Examples: a combined example

This example shows that these options can be combined to produce a variety of output formats. Here the data section is output as a tight comma separated data set.

```
empty_lhs_spacer = ""
comma_spacer = ","
no_padding = -1
las.write(s, lhs_spacer=empty_lhs_spacer, spacer=comma_spacer, len_numeric_field=no_
↪padding)
s.seek(1665)
print(s.read())
```

```
~ASCII ------------------------------------------------
1670.00000,123.45000,2550.00000,0.45000,123.45000,123.45000,110.20000,105.60000
```

(continues on next page)

(continued from previous page)

```
1669.87500,123.45000,2550.00000,0.45000,123.45000,123.45000,110.20000,105.60000
1669.75000,123.45000,2550.00000,0.45000,123.45000,123.45000,110.20000,105.60000
```

CHAPTER 7

Exporting to other formats

7.1 Comma-separated values (CSV)

lasio.LASFile objects can be converted to CSV files with a few options for how mnemonics and units are included (or not). It uses the *lasio.LASFile.to_csv()* method.

```
>>> import lasio.examples
>>> from sys import stdout
>>> las = lasio.examples.open('sample.las')
>>> las.to_csv(stdout)
DEPT,DT,RHOB,NPHI,SFLU,SFLA,ILM,ILD
M,US/M,K/M3,V/V,OHMM,OHMM,OHMM,OHMM
1670.0,123.45,2550.0,0.45,123.45,123.45,110.2,105.6
1669.875,123.45,2550.0,0.45,123.45,123.45,110.2,105.6
1669.75,123.45,2550.0,0.45,123.45,123.45,110.2,105.6
```

There are options for putting the units together with mnemonics:

```
>>> las.to_csv(stdout, units_loc='[]')
DEPT [M],DT [US/M],RHOB [K/M3],NPHI [V/V],SFLU [OHMM],SFLA [OHMM],ILM [OHMM],ILD␣
↪[OHMM]
1670.0,123.45,2550.0,0.45,123.45,123.45,110.2,105.6
1669.875,123.45,2550.0,0.45,123.45,123.45,110.2,105.6
1669.75,123.45,2550.0,0.45,123.45,123.45,110.2,105.6
```

Or leaving things out altogether:

```
>>> las.to_csv(stdout, mnemonics=False, units=False)
1670.0,123.45,2550.0,0.45,123.45,123.45,110.2,105.6
1669.875,123.45,2550.0,0.45,123.45,123.45,110.2,105.6
1669.75,123.45,2550.0,0.45,123.45,123.45,110.2,105.6
```

7.2 Excel spreadsheet (XLSX)

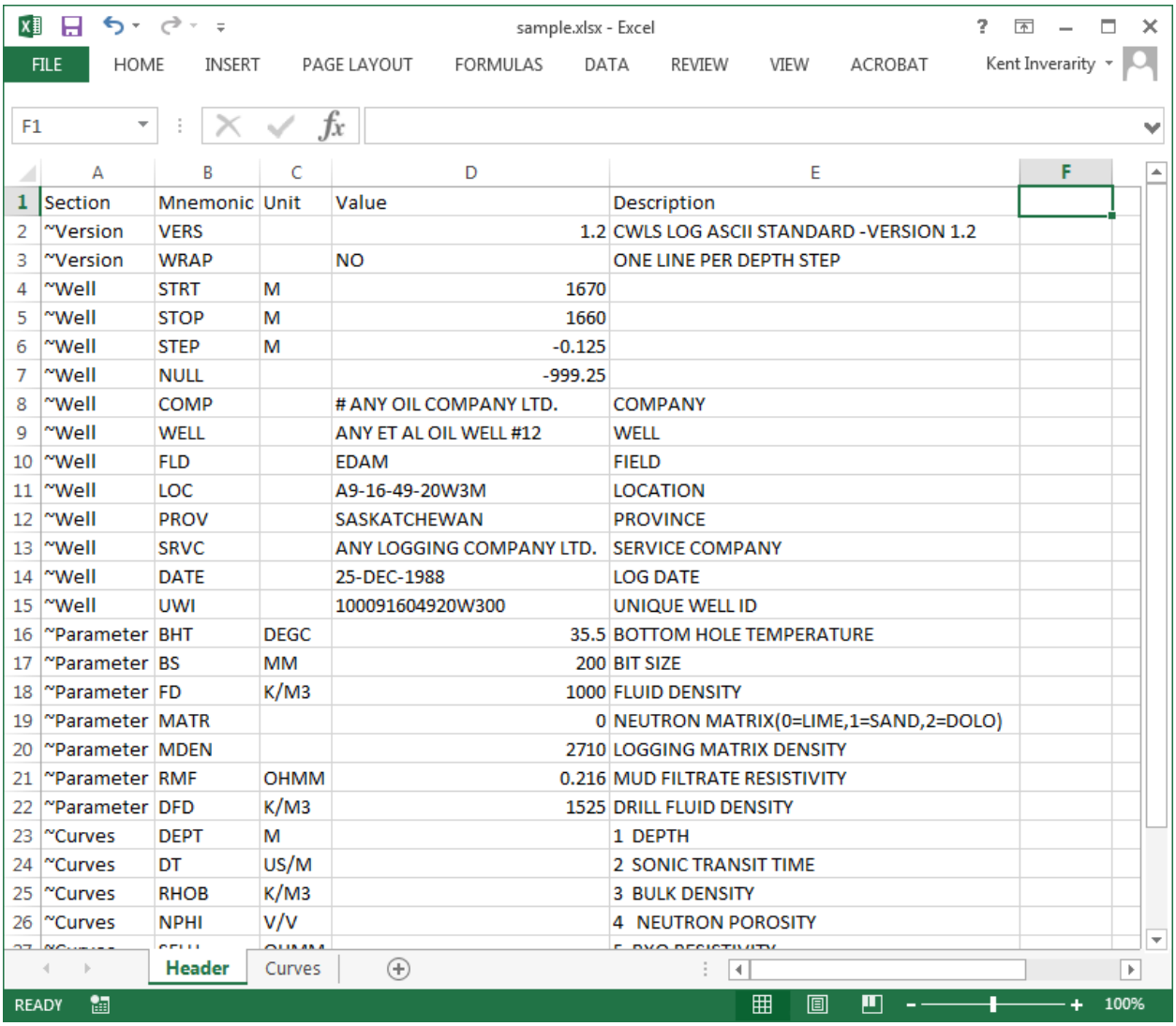
You can easily convert LAS files into Excel, retaining the header information. If we are working in Python, you export like this:

```
>>> las.to_excel('sample.xlsx')
```

You will need to have openpyxl installed (`$ pip install openpyxl`).

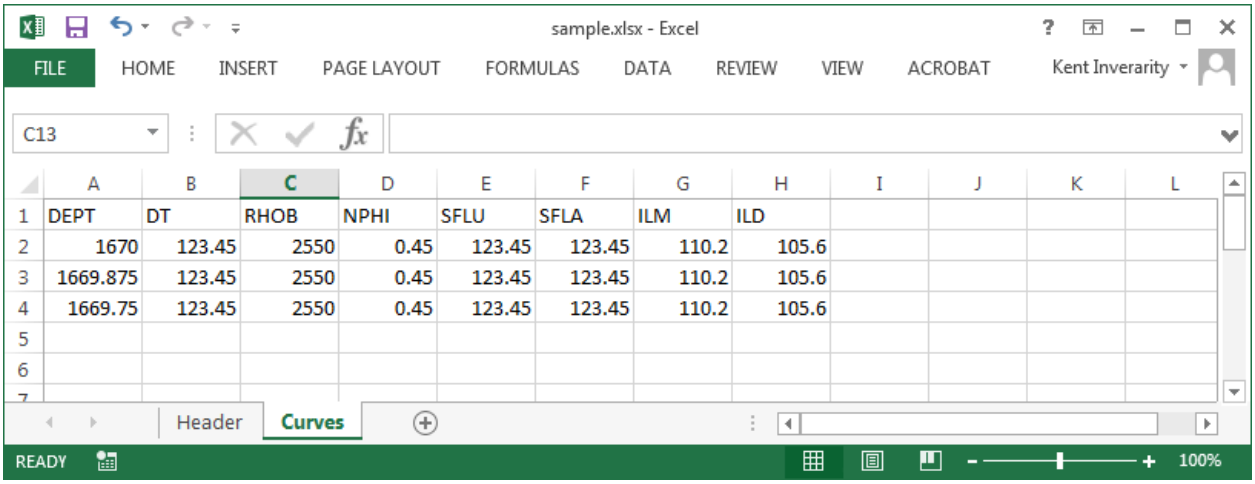
7.2.1 Format of exported Excel file

The exported spreadsheet has two sheets named "Header" and "Curves". The "Header" sheet has five columns named "Section", "Mnemonic", "Unit", "Value", and "Description", containing the information from all the sections in the header.



	A	B	C	D	E	F
1	Section	Mnemonic	Unit	Value	Description	
2	~Version	VERS		1.2	CWLS LOG ASCII STANDARD -VERSION 1.2	
3	~Version	WRAP		NO	ONE LINE PER DEPTH STEP	
4	~Well	STRT	M	1670		
5	~Well	STOP	M	1660		
6	~Well	STEP	M	-0.125		
7	~Well	NULL		-999.25		
8	~Well	COMP		# ANY OIL COMPANY LTD.	COMPANY	
9	~Well	WELL		ANY ET AL OIL WELL #12	WELL	
10	~Well	FLD		EDAM	FIELD	
11	~Well	LOC		A9-16-49-20W3M	LOCATION	
12	~Well	PROV		SASKATCHEWAN	PROVINCE	
13	~Well	SRVC		ANY LOGGING COMPANY LTD.	SERVICE COMPANY	
14	~Well	DATE		25-DEC-1988	LOG DATE	
15	~Well	UWI		100091604920W300	UNIQUE WELL ID	
16	~Parameter	BHT	DEGC	35.5	BOTTOM HOLE TEMPERATURE	
17	~Parameter	BS	MM	200	BIT SIZE	
18	~Parameter	FD	K/M3	1000	FLUID DENSITY	
19	~Parameter	MATR		0	NEUTRON MATRIX(0=LIME,1=SAND,2=DOLO)	
20	~Parameter	MDEN		2710	LOGGING MATRIX DENSITY	
21	~Parameter	RMF	OHMM	0.216	MUD FILTRATE RESISTIVITY	
22	~Parameter	DFD	K/M3	1525	DRILL FLUID DENSITY	
23	~Curves	DEPT	M		1 DEPTH	
24	~Curves	DT	US/M		2 SONIC TRANSIT TIME	
25	~Curves	RHOB	K/M3		3 BULK DENSITY	
26	~Curves	NPHI	V/V		4 NEUTRON POROSITY	

The "Curves" sheet contains the data as a table, with the curve mnemonics as a header row.



	A	B	C	D	E	F	G	H
1	DEPT	DT	RHOB	NPHI	SFLU	SFLA	ILM	ILD
2	1670	123.45	2550	0.45	123.45	123.45	110.2	105.6
3	1669.875	123.45	2550	0.45	123.45	123.45	110.2	105.6
4	1669.75	123.45	2550	0.45	123.45	123.45	110.2	105.6
5								
6								

7.2.2 Script interfaces

Single file

```
$ las2excel --help
usage: Convert LAS file to XLSX [-h] LAS_filename XLSX_filename

positional arguments:
  LAS_filename
  XLSX_filename

optional arguments:
  -h, --help     show this help message and exit

$ las2excel sample.las sample.xlsx
```

Multiple files (`las2excelbulk`)

The better script to use is `las2excelbulk`:

```
$ las2excelbulk --help
usage: Convert LAS files to XLSX [-h] [-g GLOB] [-r] [-i] path

positional arguments:
  path

optional arguments:
  -h, --help            show this help message and exit
  -g GLOB, --glob GLOB  Match LAS files with this pattern (default: *.las)
  -r, --recursive       Recurse through subfolders. (default: False)
  -i, --ignore-header-errors
                        Ignore header section errors. (default: False)
```

Here is the command to create Excel versions of all the LAS files contained within the folder `test_folder`, and any sub-folders:

Notice that some LAS files raised exceptions (in this case, `ValueError`) and were not converted. In some cases these will relate to errors in the header sections:

```
$ las2excelbulk.exe -r .
Converting .\4424\PN31769.LAS -> .\4424\pn31769.xlsx
Converting .\4424\PN31769L.LAS -> .\4424\pn31769l.xlsx
Converting .\4424\PN31769R.LAS -> .\4424\pn31769r.xlsx
Converting .\4428\pn31769.las -> .\4428\pn31769.xlsx
Failed to convert file. Error message:
Traceback (most recent call last):
  File "c:\program files (x86)\misc\kentcode\lasio\lasio\reader.py", line 366, in␣
↪parse_header_section
    values = read_line(line)
  File "c:\program files (x86)\misc\kentcode\lasio\lasio\reader.py", line 522, in␣
↪read_line
    return read_header_line(*args, **kwargs)
  File "c:\program files (x86)\misc\kentcode\lasio\lasio\reader.py", line 548, in␣
↪read_header_line
    mdict = m.groupdict()
AttributeError: 'NoneType' object has no attribute 'groupdict'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "c:\program files (x86)\misc\kentcode\lasio\lasio\excel.py", line 133, in main_
↪bulk
    l = las.LASFile(lasfn, ignore_header_errors=args.ignore_header_errors)
  File "c:\program files (x86)\misc\kentcode\lasio\lasio\las.py", line 77, in __init__
    self.read(file_ref, **read_kwargs)
  File "c:\program files (x86)\misc\kentcode\lasio\lasio\las.py", line 156, in read
    ignore_header_errors=ignore_header_errors)
  File "c:\program files (x86)\misc\kentcode\lasio\lasio\las.py", line 110, in add_
↪section
    **sect_kws)
  File "c:\program files (x86)\misc\kentcode\lasio\lasio\reader.py", line 375, in␣
↪parse_header_section
    raise exceptions.LASHeaderError(message)
lasio.exceptions.LASHeaderError: Line #21 - failed in ~Well Information section on␣
↪line:
PN        PERMIT NUMBER: 31769AttributeError: 'NoneType' object has no attribute
↪'groupdict'

Converting .\4526\PENRICE.LAS -> .\4526\penrice.xlsx
```

But in this case I'm happy to lose that single corrupted line in the header in the conversion. In order to force lasio to ignore the error and continue to convert the file, use the `--ignore-header-errors` flag (`-i` for short):

```
$ las2excelbulk.exe -r -i .
Converting .\4424\PN31769.LAS -> .\4424\pn31769.xlsx
Converting .\4424\PN31769L.LAS -> .\4424\pn31769l.xlsx
Converting .\4424\PN31769R.LAS -> .\4424\pn31769r.xlsx
Converting .\4428\pn31769.las -> .\4428\pn31769.xlsx
Line #21 - failed in ~Well Information section on line:
PN        PERMIT NUMBER: 31769AttributeError: 'NoneType' object has no attribute
↪'groupdict'
Converting .\4526\PENRICE.LAS -> .\4526\penrice.xlsx
```

lasio still reports the problem, but ignores it and continues the conversion of the file.

CHAPTER 8

Building a LAS file from scratch

When you create a *lasio.LASFile* from scratch, it comes with some default metadata:

```
>>> import lasio
>>> las = lasio.LASFile()
>>> las.header
{'Version': [HeaderItem(mnemonic="VERS", unit="", value="2.0", descr="CWLS log ASCII␣
↪Standa"),
 HeaderItem(mnemonic="WRAP", unit="", value="NO", descr="One line per depth ste"),
 HeaderItem(mnemonic="DLM", unit="", value="SPACE", descr="Column Data Section ")],
'Well': [HeaderItem(mnemonic="STRT", unit="m", value="nan", descr="START DEPTH"),
 HeaderItem(mnemonic="STOP", unit="m", value="nan", descr="STOP DEPTH"),
 HeaderItem(mnemonic="STEP", unit="m", value="nan", descr="STEP"),
 HeaderItem(mnemonic="NULL", unit="", value="-9999.25", descr="NULL VALUE"),
 HeaderItem(mnemonic="COMP", unit="", value="", descr="COMPANY"),
 HeaderItem(mnemonic="WELL", unit="", value="", descr="WELL"),
 HeaderItem(mnemonic="FLD", unit="", value="", descr="FIELD"),
 HeaderItem(mnemonic="LOC", unit="", value="", descr="LOCATION"),
 HeaderItem(mnemonic="PROV", unit="", value="", descr="PROVINCE"),
 HeaderItem(mnemonic="CNTY", unit="", value="", descr="COUNTY"),
 HeaderItem(mnemonic="STAT", unit="", value="", descr="STATE"),
 HeaderItem(mnemonic="CTRY", unit="", value="", descr="COUNTRY"),
 HeaderItem(mnemonic="SRVC", unit="", value="", descr="SERVICE COMPANY"),
 HeaderItem(mnemonic="DATE", unit="", value="", descr="DATE"),
 HeaderItem(mnemonic="UWI", unit="", value="", descr="UNIQUE WELL ID"),
 HeaderItem(mnemonic="API", unit="", value="", descr="API NUMBER")],
'Curves': [],
'Parameter': [],
'Other': ''}
```

In our case, let's set the correct date:

```
>>> from datetime import datetime
>>> las.well.DATE = datetime.today().strftime('%Y-%m-%d %H:%M:%S')
```

And add some new header fields:

```
>>> las.params['ENG'] = lasio.HeaderItem('ENG', value='Kent Inverarity')
>>> las.params['LMF'] = lasio.HeaderItem('LMF', value='GL')
>>> las.other = 'Example of how to create a LAS file from scratch using lasio'
```

We will invent some data for a curve:

```
>>> import numpy as np
>>> depths = np.arange(10, 50, 0.5)
>>> synth = np.log10(depths)*5+np.random.random(len(depths))
>>> synth[:8] = np.nan
```

. . . add these to the LASFile object:

```
>>> las.append_curve('DEPT', depths, unit='m')
>>> las.append_curve('SYNTH', synth, descr='fake data')
```

And write the result to files:

```
>>> las.write('scratch_v1.2.las', version=1.2)
>>> las.write('scratch_v2.las', version=2)
```

Here is the resulting scratch_v1.2.las:

```
~Version ------------------------------------------------
VERS.   1.2 : CWLS LOG ASCII STANDARD - VERSION 1.2
WRAP.    NO : One line per depth step
DLM . SPACE : Column Data Section Delimiter
~Well ---------------------------------------------------
STRT.m       10.00000 : START DEPTH
STOP.m       49.50000 : STOP DEPTH
STEP.m        0.50000 : STEP
NULL.        -9999.25 : NULL VALUE
COMP.         COMPANY :
WELL.            WELL :
FLD .           FIELD :
LOC .        LOCATION :
PROV.        PROVINCE :
CNTY.          COUNTY :
STAT.           STATE :
CTRY.         COUNTRY :
SRVC. SERVICE COMPANY :
DATE.            DATE : 2023-01-26 14:58:21
UWI .  UNIQUE WELL ID :
API .      API NUMBER :
~Curve Information --------------------------------------
DEPT .m  :
SYNTH.   : fake data
~Params -------------------------------------------------
ENG. Kent Inverarity :
LMF.              GL :
~Other --------------------------------------------------
Example of how to create a LAS file from scratch using lasio
~ASCII --------------------------------------------------
   10.00000   -9999.25
   10.50000   -9999.25
   11.00000   -9999.25
   11.50000   -9999.25
```

(continues on next page)

(continued from previous page)

```
   12.00000   -9999.25
   12.50000   -9999.25
   13.00000   -9999.25
   13.50000   -9999.25
   14.00000    6.32656
   14.50000    6.32279
   15.00000    6.24716
   15.50000    6.07168
   16.00000    6.40693
   16.50000    6.74994
   17.00000    6.16163
   17.50000    7.08836
   18.00000    6.31721
   18.50000    7.19034
   19.00000    6.72278
   19.50000    7.01719
   20.00000    7.49475
   20.50000    6.92995
   21.00000    7.44739
   21.50000    7.55360
   22.00000    6.94753
   22.50000    7.64236
   23.00000    7.74817
   23.50000    7.23852
   24.00000    7.88034
   24.50000    7.07664
   25.00000    7.19182
   25.50000    7.62403
   26.00000    7.80678
   26.50000    7.93082
   27.00000    8.08903
   27.50000    7.81581
   28.00000    8.08901
   28.50000    7.60532
   29.00000    7.86530
   29.50000    7.72080
   30.00000    7.74472
   30.50000    7.68292
   31.00000    8.00722
   31.50000    8.12406
   32.00000    7.60265
   32.50000    7.73699
   33.00000    7.72325
   33.50000    8.02248
   34.00000    8.04029
   34.50000    8.65056
   35.00000    8.30488
   35.50000    8.59884
   36.00000    7.83725
   36.50000    8.72173
   37.00000    7.95948
   37.50000    8.12969
   38.00000    8.75692
   38.50000    8.73753
   39.00000    8.22793
   39.50000    8.86533
   40.00000    8.56819
```

(continues on next page)

(continued from previous page)

```
   40.50000    9.00213
   41.00000    8.51844
   41.50000    8.81121
   42.00000    8.51106
   42.50000    8.28359
   43.00000    8.65719
   43.50000    8.33235
   44.00000    8.52983
   44.50000    9.04601
   45.00000    8.53333
   45.50000    9.20433
   46.00000    8.60132
   46.50000    8.94629
   47.00000    8.60415
   47.50000    8.56460
   48.00000    9.35277
   48.50000    8.65887
   49.00000    9.33907
   49.50000    9.30430
```

and scratch_v2.las:

```
~Version ------------------------------------------------
VERS.   2.0 : CWLS log ASCII Standard -VERSION 2.0
WRAP.    NO : One line per depth step
DLM . SPACE : Column Data Section Delimiter
~Well ---------------------------------------------------
STRT.m           10.00000 : START DEPTH
STOP.m           49.50000 : STOP DEPTH
STEP.m            0.50000 : STEP
NULL.            -9999.25 : NULL VALUE
COMP.                     : COMPANY
WELL.                     : WELL
FLD .                     : FIELD
LOC .                     : LOCATION
PROV.                     : PROVINCE
CNTY.                     : COUNTY
STAT.                     : STATE
CTRY.                     : COUNTRY
SRVC.                     : SERVICE COMPANY
DATE. 2023-01-26 14:58:21 : DATE
UWI .                     : UNIQUE WELL ID
API .                     : API NUMBER
~Curve Information --------------------------------------
DEPT .m  :
SYNTH.   : fake data
~Params -------------------------------------------------
ENG. Kent Inverarity :
LMF.             GL :
~Other --------------------------------------------------
Example of how to create a LAS file from scratch using lasio
~ASCII --------------------------------------------------
   10.00000   -9999.25
   10.50000   -9999.25
   11.00000   -9999.25
   11.50000   -9999.25
   12.00000   -9999.25
```

(continues on next page)

(continued from previous page)

```
   12.50000   -9999.25
   13.00000   -9999.25
   13.50000   -9999.25
   14.00000    6.32656
   14.50000    6.32279
   15.00000    6.24716
   15.50000    6.07168
   16.00000    6.40693
   16.50000    6.74994
   17.00000    6.16163
   17.50000    7.08836
   18.00000    6.31721
   18.50000    7.19034
   19.00000    6.72278
   19.50000    7.01719
   20.00000    7.49475
   20.50000    6.92995
   21.00000    7.44739
   21.50000    7.55360
   22.00000    6.94753
   22.50000    7.64236
   23.00000    7.74817
   23.50000    7.23852
   24.00000    7.88034
   24.50000    7.07664
   25.00000    7.19182
   25.50000    7.62403
   26.00000    7.80678
   26.50000    7.93082
   27.00000    8.08903
   27.50000    7.81581
   28.00000    8.08901
   28.50000    7.60532
   29.00000    7.86530
   29.50000    7.72080
   30.00000    7.74472
   30.50000    7.68292
   31.00000    8.00722
   31.50000    8.12406
   32.00000    7.60265
   32.50000    7.73699
   33.00000    7.72325
   33.50000    8.02248
   34.00000    8.04029
   34.50000    8.65056
   35.00000    8.30488
   35.50000    8.59884
   36.00000    7.83725
   36.50000    8.72173
   37.00000    7.95948
   37.50000    8.12969
   38.00000    8.75692
   38.50000    8.73753
   39.00000    8.22793
   39.50000    8.86533
   40.00000    8.56819
   40.50000    9.00213
```

(continues on next page)

(continued from previous page)

```
   41.00000    8.51844
   41.50000    8.81121
   42.00000    8.51106
   42.50000    8.28359
   43.00000    8.65719
   43.50000    8.33235
   44.00000    8.52983
   44.50000    9.04601
   45.00000    8.53333
   45.50000    9.20433
   46.00000    8.60132
   46.50000    8.94629
   47.00000    8.60415
   47.50000    8.56460
   48.00000    9.35277
   48.50000    8.65887
   49.00000    9.33907
   49.50000    9.30430
```

CHAPTER 9

Character encodings

There are four options:

1. Specify the encoding (internally lasio uses the open function from codecs which is part of the standard library):

```
>>> las = lasio.read('example.las', encoding='windows-1252')
```

2. Do nothing. By default *lasio.read()* uses the keyword argument `autodetect_encoding=True`. This will try to open the file with a few different encodings, like 'ascii', 'windows-1252', and 'latin-1'. The first one to raise no `UnicodeDecodeError` exceptions will be used.

This may still result in an error, or incorrectly decoded characters.

3. Install optional package chardet to automatically detect the character encoding. If chardet is installed then lasio will use it by default:

```
>>> import logging
>>> logging.basicConfig()
>>> logging.getLogger().setLevel(logging.DEBUG)
>>> las = lasio.read('encodings_utf8.las')
DEBUG:lasio.reader:get_encoding Using chardet
DEBUG:lasio.reader:chardet method detected encoding of utf-8 at confidence 0.99
INFO:lasio.reader:Opening encodings_utf8.las as utf-8 and treating errors with
↪"replace"
...
```

This may still result in an error, or incorrectly decoded characters.

If you are certain that you have no "extended characters" (or that you don't care), you can easily speed up lasio's performance by using:

```
>>> try:
...     las = lasio.read('example.las', autodetect_encoding=False)
... except UnicodeDecodeError:
...     continue
```

CHAPTER 10

Docstrings for the lasio package

10.1 Reading LAS files

lasio.**read**(*file_ref*, ***kwargs*)

Read a LAS file.

Note that only versions 1.2 and 2.0 of the LAS file specification are fully supported. There is partial support for reading LAS 3.0 files.

Parameters **file_ref** (file-like object or `str`) – either a filename, an open file object, or a string containing the contents of a file.

Keyword Arguments

- **ignore_header_errors** (*bool*) – ignore LASHeaderErrors (False by default)
- **ignore_comments** (*sequence/str*) – ignore lines beginning with these characters e.g. `("#", '"')` in header sections.
- **ignore_data_comments** (*str*) – ignore lines beginning with this character in data sections only.
- **mnemonic_case** (*str*) – 'preserve': keep the case of HeaderItem mnemonics 'upper': convert all HeaderItem mnemonics to uppercase 'lower': convert all HeaderItem mnemonics to lowercase
- **ignore_data** (*bool*) – if True, do not read in any of the actual data, just the header metadata. False by default.
- **engine** (*str*) – "normal": parse data section with normal Python reader (quite slow); "numpy": parse data section with *numpy.genfromtxt* (fast). By default the engine is "numpy".
- **use_normal_engine_for_wrapped** (*bool*) – if header metadata indicates that the file is wrapped, always use the 'normal' engine. Default is True. The only reason you should use False is if speed is a very high priority and you had files with metadata that incorrectly indicates they are wrapped.

- **read_policy** (*str or list*) – Apply regular expression substitutions for common errors in fixed-width formatted data sections. If you do not want any such substitutions to applied, pass `read_policy=()`.
- **null_policy** (*str or list*) – see https://lasio.readthedocs.io/en/latest/data-section.html#handling-invalid-data-indicators-automatically
- **accept_regexp_sub_recommendations** (*bool*) – Accept recommendations to auto- matically remove read substitutions (applied by the default read_policy) which look for numeric run-on errors involving hyphens. This avoids incorrect parsing of dates such as '2018-05-22' as three separate columns containing '2018', '-5' and '-22'. The read substitutions are applied only if the inspection code of the data section finds a hyphen in every line. The only circumstance where this should be manually set to False is where you have very problematic fixed-column-width data sections involving negative values.
- **index_unit** (*str*) – Optionally force-set the index curve's unit to "m" or "ft"
- **dtypes** (*"auto", dict or list*) – specify the data types for each curve in the ~ASCII data section. If "auto", each curve will be converted to floats if possible and remain as str if not. If a dict you can specify only the curve mnemonics you want to convert as a key. If a list, please specify data types for each curve in order. Note that the conversion currently only occurs via numpy.ndarray.astype() and therefore only a few simple casts will work e.g. *int*, *float*, *str*.
- **encoding** (*str*) – character encoding to open file_ref with, using `io.open()` (this is handled by *lasio.reader.open_with_codecs()*)
- **encoding_errors** (*str*) – 'strict', 'replace' (default), 'ignore' - how to handle errors with encodings (see this section of the standard library's `codecs` module for more information) (this is handled by *lasio.reader.open_with_codecs()*)
- **autodetect_encoding** (*str or bool*) – default True to use chardet to detect encoding. Note if set to False several common encodings will be tried but chardet won't be used. (this is handled by *lasio.reader.open_with_codecs()*)
- **autodetect_encoding_chars** (*int/None*) – number of chars to read from LAS file for auto-detection of encoding. (this is handled by *lasio.reader.open_with_codecs()*)

Returns a *lasio.LASFile* object representing the file

The documented arguments above are combined from these methods:

- *lasio.reader.open_with_codecs()* - manage issues relate to character encodings
- *lasio.LASFile.read()* - control how NULL values and errors are handled during parsing

class lasio.**LASFile**(*file_ref=None*, ***read_kwargs*)

LAS file object.

Keyword Arguments

- **file_ref** (file-like object or `str`) – either a filename, an open file object, or a string containing the contents of a file.
- **ignore_header_errors** (*bool*) – ignore LASHeaderErrors (False by default)
- **ignore_comments** (*sequence/str*) – ignore lines beginning with these characters e.g. `("#", '"')` in header sections.
- **ignore_data_comments** (*str*) – ignore lines beginning with this character in data sections only.

- **mnemonic_case** (*str*) – 'preserve': keep the case of HeaderItem mnemonics 'upper': convert all HeaderItem mnemonics to uppercase 'lower': convert all HeaderItem mnemonics to lowercase
- **ignore_data** (*bool*) – if True, do not read in any of the actual data, just the header metadata. False by default.
- **engine** (*str*) – "normal": parse data section with normal Python reader (quite slow); "numpy": parse data section with *numpy.genfromtxt* (fast). By default the engine is "numpy".
- **use_normal_engine_for_wrapped** (*bool*) – if header metadata indicates that the file is wrapped, always use the 'normal' engine. Default is True. The only reason you should use False is if speed is a very high priority and you had files with metadata that incorrectly indicates they are wrapped.
- **read_policy** (*str or list*) – Apply regular expression substitutions for common errors in fixed-width formatted data sections. If you do not want any such substitutions to applied, pass `read_policy=()`.
- **null_policy** (*str or list*) – see https://lasio.readthedocs.io/en/latest/data-section.html#handling-invalid-data-indicators-automatically
- **accept_regexp_sub_recommendations** (*bool*) – Accept recommendations to auto- matically remove read substitutions (applied by the default read_policy) which look for numeric run-on errors involving hyphens. This avoids incorrect parsing of dates such as '2018-05-22' as three separate columns containing '2018', '-5' and '-22'. The read substitutions are applied only if the inspection code of the data section finds a hyphen in every line. The only circumstance where this should be manually set to False is where you have very problematic fixed-column-width data sections involving negative values.
- **index_unit** (*str*) – Optionally force-set the index curve's unit to "m" or "ft"
- **dtypes** (*"auto", dict or list*) – specify the data types for each curve in the ~ASCII data section. If "auto", each curve will be converted to floats if possible and remain as str if not. If a dict you can specify only the curve mnemonics you want to convert as a key. If a list, please specify data types for each curve in order. Note that the conversion currently only occurs via numpy.ndarray.astype() and therefore only a few simple casts will work e.g. *int*, *float*, *str*.
- **encoding** (*str*) – character encoding to open file_ref with, using `io.open()` (this is handled by *lasio.reader.open_with_codecs()*)
- **encoding_errors** (*str*) – 'strict', 'replace' (default), 'ignore' - how to handle errors with encodings (see this section of the standard library's `codecs` module for more information) (this is handled by *lasio.reader.open_with_codecs()*)
- **autodetect_encoding** (*str or bool*) – default True to use chardet to detect encoding. Note if set to False several common encodings will be tried but chardet won't be used. (this is handled by *lasio.reader.open_with_codecs()*)
- **autodetect_encoding_chars** (*int/None*) – number of chars to read from LAS file for auto-detection of encoding. (this is handled by *lasio.reader.open_with_codecs()*)

The documented arguments above are combined from these methods:

- *lasio.reader.open_with_codecs()* - manage issues relate to character encodings
- *lasio.LASFile.read()* - control how NULL values and errors are handled during parsing

encoding
the character encoding used when reading the file in from disk

Type str or None

LASFile.**read**(*file_ref*, *ignore_header_errors=False*, *ignore_comments=('#',)*, *ignore_data_comments='#'*, *mnemonic_case='upper'*, *ignore_data=False*, *engine='numpy'*, *use_normal_engine_for_wrapped=True*, *read_policy='default'*, *null_policy='strict'*, *accept_regexp_sub_recommendations=True*, *index_unit=None*, *dtypes='auto'*, ***kwargs*)

Read a LAS file.

Parameters **file_ref** (file-like object or `str`) – either a filename, an open file object, or a string containing the contents of a file.

Keyword Arguments

- **ignore_header_errors** (*bool*) – ignore LASHeaderErrors (False by default)
- **ignore_comments** (*sequence/str*) – ignore lines beginning with these characters e.g. `("#", '"')` in header sections.
- **ignore_data_comments** (*str*) – ignore lines beginning with this character in data sections only.
- **mnemonic_case** (*str*) – 'preserve': keep the case of HeaderItem mnemonics 'upper': convert all HeaderItem mnemonics to uppercase 'lower': convert all HeaderItem mnemonics to lowercase
- **ignore_data** (*bool*) – if True, do not read in any of the actual data, just the header metadata. False by default.
- **engine** (*str*) – "normal": parse data section with normal Python reader (quite slow); "numpy": parse data section with *numpy.genfromtxt* (fast). By default the engine is "numpy".
- **use_normal_engine_for_wrapped** (*bool*) – if header metadata indicates that the file is wrapped, always use the 'normal' engine. Default is True. The only reason you should use False is if speed is a very high priority and you had files with metadata that incorrectly indicates they are wrapped.
- **read_policy** (*str or list*) – Apply regular expression substitutions for common errors in fixed-width formatted data sections. If you do not want any such substitutions to applied, pass `read_policy=()`.
- **null_policy** (*str or list*) – see https://lasio.readthedocs.io/en/latest/data-section.html#handling-invalid-data-indicators-automatically
- **accept_regexp_sub_recommendations** (*bool*) – Accept recommendations to auto- matically remove read substitutions (applied by the default read_policy) which look for numeric run-on errors involving hyphens. This avoids incorrect parsing of dates such as '2018-05-22' as three separate columns containing '2018', '-5' and '-22'. The read substitutions are applied only if the inspection code of the data section finds a hyphen in every line. The only circumstance where this should be manually set to False is where you have very problematic fixed-column-width data sections involving negative values.
- **index_unit** (*str*) – Optionally force-set the index curve's unit to "m" or "ft"
- **dtypes** (*"auto", dict or list*) – specify the data types for each curve in the ~ASCII data section. If "auto", each curve will be converted to floats if possible and remain as str if not. If a dict you can specify only the curve mnemonics you want to convert as a key. If a list, please specify data types for each curve in order. Note that the conversion

currently only occurs via numpy.ndarray.astype() and therefore only a few simple casts will work e.g. *int*, *float*, *str*.

- **encoding** (*str*) – character encoding to open file_ref with, using `io.open()` (this is handled by *lasio.reader.open_with_codecs()*)
- **encoding_errors** (*str*) – 'strict', 'replace' (default), 'ignore' - how to handle errors with encodings (see this section of the standard library's `codecs` module for more information) (this is handled by *lasio.reader.open_with_codecs()*)
- **autodetect_encoding** (*str or bool*) – default True to use chardet to detect encoding. Note if set to False several common encodings will be tried but chardet won't be used. (this is handled by *lasio.reader.open_with_codecs()*)
- **autodetect_encoding_chars** (*int/None*) – number of chars to read from LAS file for auto-detection of encoding. (this is handled by *lasio.reader.open_with_codecs()*)

lasio.**open_file**(*file_ref*, ***encoding_kwargs*)

Open a file if necessary.

If `autodetect_encoding=True` then `chardet` needs to be installed, or else an `ImportError` will be raised.

Parameters **file_ref** (*file-like object, str*) – either a filename, an open file object, or a string containing the contents of a file.

See *lasio.reader.open_with_codecs()* for keyword arguments that can be used here.

Returns tuple of an open file-like object, and the encoding that was used to decode it (if it were read from disk).

lasio.reader.**open_with_codecs**(*filename*, *encoding=None*, *encoding_errors='replace'*, *autodetect_encoding=True*, *autodetect_encoding_chars=4000*)

Read Unicode data from file.

Parameters **filename** (*str*) – path to file

Keyword Arguments

- **encoding** (*str*) – character encoding to open file_ref with, using `io.open()`.
- **encoding_errors** (*str*) – 'strict', 'replace' (default), 'ignore' - how to handle errors with encodings (see this section of the standard library's `codecs` module for more information)
- **autodetect_encoding** (*str or bool*) – default True to use chardet to detect encoding. Note if set to False several common encodings will be tried but chardet won't be used.
- **autodetect_encoding_chars** (*int/None*) – number of chars to read from LAS file for auto-detection of encoding.

Returns a unicode or string object

This function is called by `lasio.reader.open_file()`.

lasio.reader.**get_encoding**(*auto*, *raw*)

Automatically detect character encoding.

Parameters

- **auto** (*str*) – auto-detection of character encoding - can be one of 'chardet', False, or True (the latter will pick the fastest available option)

- **raw** (*bytes*) – array of bytes to detect from

Returns A string specifying the character encoding.

LASFile.**match_raw_section**(*pattern*, *re_func='match'*, *flags=<RegexFlag.IGNORECASE: 2>*)
Find raw section with a regular expression.

Parameters **pattern** (*str*) – regular expression (you need to include the tilde)

Keyword Arguments

- **re_func** (*str*) – either "match" or "search", see python `re` module.
- **flags** (*int*) – flags for `re.compile()`

Returns dict

Intended for internal use only.

lasio.reader.**read_data_section_iterative_normal_engine**(*file_obj*, *line_nos*, *regexp_subs*, *value_null_subs*, *ignore_data_comments*, *n_columns*, *dtypes*, *line_splitter*)
Read data section into memory.

Parameters

- **file_obj** – file-like object open for reading at the beginning of the section
- **line_nos** (*tuple*) – the first and last line no of the section to read
- **regexp_subs** (*list*) – each item should be a tuple of the pattern and substitution string for a call to re.sub() on each line of the data section. See defaults.py READ_SUBS and NULL_SUBS for examples.
- **value_null_subs** (*list*) – list of numerical values to be replaced by numpy.nan values.
- **ignore_data_comments** (*str*) – lines beginning with this character will be ignored
- **n_columns** (*int*) – expected number of columns
- **dtypes** (*list, "auto", False*) – list of expected data types for each column, (each data type can be specified as e.g. *int*, *float*, *str*, *datetime*). If you specify 'auto', then this function will attempt to convert each column to a float and if that fails, the column will be returned as a string. If you specify False, no conversion of data types will be attempt at all.
- **line_splitter** (*function*) – This function is dynamically configured to split data lines on the configured delimiter

Returns: generator which yields the data as a 1D ndarray for each column at a time.

lasio.reader.**read_data_section_iterative_numpy_engine**(*file_obj*, *line_nos*)
Read data section into memory.

Parameters

- **file_obj** – file-like object open for reading at the beginning of the section
- **line_nos** (*tuple*) – the first and last line no of the section to read

Returns A numpy ndarray.

lasio.reader.**get_substitutions**(*read_policy*, *null_policy*)
Parse read and null policy definitions into a list of regexp and value substitutions.

Parameters

- **read_policy** (*str, list, or substitution*) – either (1) a string defined in defaults.READ_POLICIES; (2) a list of substitutions as defined by the keys of defaults.READ_SUBS; or (3) a list of actual substitutions similar to the values of defaults.READ_SUBS. You can mix (2) and (3) together if you want.
- **null_policy** (*str, list, or sub*) – as for read_policy but for defaults.NULL_POLICIES and defaults.NULL_SUBS

Returns regexp_subs, value_null_subs, version_NULL - two lists and a bool. The first list is pairs of regexp patterns and substrs, and the second list is just a list of floats or integers. The bool is whether or not 'NULL' was located as a substitution.

The default READ_POLICIES are

- comma-decimal-mark : in numbers replace a comma divider with a decimal
- run-on(-) : separate 2 numbers that run together on the negative sign
- run-on(.) : replace numbers with 2 or more decimals or a NaN and a decimal with 2 NaNs

class lasio.reader.**SectionParser**(*title*, *version=1.2*)

Parse lines from header sections.

Parameters **title** (*str*) – title line of section. Used to understand different order formatting across the special sections ~C, ~P, ~W, and ~V, depending on version 1.2 or 2.0.

Keyword Arguments **version** (*float*) – version to parse according to. Default is 1.2.

lasio.reader.**read_header_line**(*line*, *pattern=None*, *section_name=None*)

Read a line from a LAS header section.

The line is parsed with a regular expression – see LAS file specs for more details, but it should basically be in the format:

```
name.unit       value : descr
```

Parameters

- **line** (*str*) – line from a LAS header section
- **section_name** (*str*) – Name of the section the 'line' is from. The default
- **is None.** (*value*) –

Returns A dictionary with keys 'name', 'unit', 'value', and 'descr', each containing a string as value.

class lasio.**HeaderItem**(*mnemonic=''*, *unit=''*, *value=''*, *descr=''*, *data=None*)

Dictionary/namedtuple-style object for a LAS header line.

Parameters

- **mnemonic** (*str*) – the mnemonic
- **unit** (*str*) – the unit (no whitespace!)
- **value** (*str*) – value
- **descr** (*str*) – description

These arguments are available for use as either items or attributes of the object.

HeaderItem.**set_session_mnemonic_only**(*value*)
Set the mnemonic for session use.

See source comments for `lasio.HeaderItem.__init__` for a more in-depth explanation.

class lasio.**CurveItem**(*mnemonic=''*, *unit=''*, *value=''*, *descr=''*, *data=None*)
Dictionary/namedtuple-style object for a LAS curve.

See `lasio.HeaderItem`` for the (keyword) arguments.

Keyword Arguments **data** (*array-like, 1-D*) – the curve's data.

class lasio.**SectionItems**(**args*, ***kwargs*)
Variant of a `list` which is used to represent a LAS section.

10.2 Reading data

LASFile.**__getitem__**(*key*)
Provide access to curve data.

Parameters **key** (*str, int*) – either a curve mnemonic or the column index.

Returns 1D `numpy.ndarray` (the data for the curve)

LASFile.**__setitem__**(*key*, *value*)
Append a curve.

Parameters

- **key** (*str*) – the curve mnemonic
- **value** (*1D data or CurveItem*) – either the curve data, or a CurveItem

See *lasio.LASFile.append_curve_item()* or *lasio.LASFile.append_curve()* for more details.

LASFile.**get_curve**(*mnemonic*)
Return CurveItem object.

Parameters **mnemonic** (*str*) – the name of the curve

Returns *lasio.CurveItem* (not just the data array)

LASFile.**keys**()
Return curve mnemonics.

LASFile.**values**()
Return data for each curve.

LASFile.**items**()
Return mnemonics and data for all curves.

LASFile.**df**()
Return data as a `pandas.DataFrame` structure.

The first Curve of the LASFile object is used as the pandas DataFrame's index.

LASFile.**version**
Header information from the Version (~V) section.

Returns *lasio.SectionItems* object.

LASFile.**well**
Header information from the Well (~W) section.

Returns *lasio.SectionItems* object.

LASFile.**curves**
Curve information and data from the Curves (~C) and data section..

Returns *lasio.SectionItems* object.

LASFile.**curvesdict**
Curve information and data from the Curves (~C) and data section..

Returns dict

LASFile.**params**
Header information from the Parameter (~P) section.

Returns *lasio.SectionItems* object.

LASFile.**other**
Header information from the Other (~O) section.

Returns str

LASFile.**index**
Return data from the first column of the LAS file data (depth/time).

LASFile.**depth_m**
Return the index as metres.

LASFile.**depth_ft**
Return the index as feet.

LASFile.**data**

LASFile.**stack_curves**(*mnemonic*, *sort_curves=True*)
Stack multi-channel curve data to a numpy 2D ndarray.

Provide a stub name (prefix shared by all curves that will be stacked) or a list of curve mnemonic strings.

Keyword Arguments

- **mnemonic** (*str or list*) – Supply the first several characters of the channel set to be stacked. Alternatively, supply a list of the curve names (mnemonics strings) to be stacked.
- **sort_curves** (*bool*) – Natural sort curves based on mnemonic prior to stacking.

Returns 2-D numpy array

10.3 Reading and modifying header data

class lasio.**SectionItems**(**args*, ***kwargs*)
Variant of a `list` which is used to represent a LAS section.

keys()
Return mnemonics of all the HeaderItems in the section.

values()
Return HeaderItems in the section.

items()
Return pairs of (mnemonic, HeaderItem) from the section.

get(*mnemonic*, *default=''*, *add=False*)
Get an item, with a default value for the situation when it is missing.

Parameters

- **mnemonic** (*str*) – mnemonic of item to retrieve
- **default** (*str, HeaderItem, or CurveItem*) – default to provide if *mnemonic* is missing from the section. If a string is provided, it will be used as the `value` attribute of a new HeaderItem or the `descr` attribute of a new CurveItem.
- **add** (*bool*) – if True, the returned HeaderItem/CurveItem will also be appended to the SectionItems. By default this is not done.

Returns item from the section, if it is in there, or a new item, if it is not. If a CurveItem is returned, the `data` attribute will contain `numpy.nan` values.

Return type *lasio.HeaderItem*/*lasio.CurveItem*

set_item(*key*, *newitem*)
Replace an item by comparison of session mnemonics.

Parameters

- **key** (*str*) – the item mnemonic (or HeaderItem with mnemonic) you want to replace.
- **newitem** (HeaderItem) – the new item

If **key** is not present, it appends **newitem**.

set_item_value(*key*, *value*)
Set the `value` attribute of an item.

Parameters

- **key** (*str*) – the mnemonic of the item (or HeaderItem with the mnemonic) you want to edit
- **value** (*str, int, float*) – the new value.

append(*newitem*)
Append a new HeaderItem to the object.

insert(*i*, *newitem*)
Insert a new HeaderItem to the object.

assign_duplicate_suffixes(*test_mnemonic=None*)
Check and re-assign suffixes for duplicate mnemonics.

Parameters **test_mnemonic** (*str, optional*) – check for duplicates of this mnemonic. If it is None, check all mnemonics.

dictview()
View of mnemonics and values as a dict.

Returns dict - keys are the mnemonics and the values are the `value` attributes.

10.4 Modifying data

LASFile.**set_data**(*array_like*, *names=None*, *truncate=False*)
Set the data for the LAS; actually sets data on individual curves.

Parameters **array_like** (array_like or `pandas.DataFrame`) – 2-D data array

Keyword Arguments

- **names** (*list, optional*) – used to replace the names of the existing *lasio.CurveItem* objects.
- **truncate** (*bool*) – remove any columns which are not included in the Curves (~C) section.

Note: you can pass a `pandas.DataFrame` to this method. If you do this, the index of the DataFrame will be used as the first curve in the LAS file (i.e. it will not be discarded).

`LASFile.`**`set_data_from_df`**(*df*, ***kwargs*)
Set the LAS file data from a `pandas.DataFrame`.

Parameters **df** (*pandas.DataFrame*) – curve mnemonics are the column names. The depth column for the curves must be the index of the DataFrame.

Keyword arguments are passed to *lasio.LASFile.set_data()*.

`LASFile.`**`append_curve`**(*mnemonic*, *data*, *unit=''*, *descr=''*, *value=''*)
Add a curve.

Parameters

- **mnemonic** (*str*) – the curve mnemonic
- **data** (*1D ndarray*) – the curve data

Keyword Arguments

- **unit** (*str*) – curve unit
- **descr** (*str*) – curve description
- **value** (*int/float/str*) – value e.g. API code.

`LASFile.`**`insert_curve`**(*ix*, *mnemonic*, *data*, *unit=''*, *descr=''*, *value=''*)
Insert a curve.

Parameters

- **ix** (*int*) – position to insert curve at i.e. 0 for start.
- **mnemonic** (*str*) – the curve mnemonic
- **data** (*1D ndarray*) – the curve data

Keyword Arguments

- **unit** (*str*) – curve unit
- **descr** (*str*) – curve description
- **value** (*int/float/str*) – value e.g. API code.

`LASFile.`**`delete_curve`**(*mnemonic=None*, *ix=None*)
Delete a curve.

Keyword Arguments

- **ix** (*int*) – index of curve in LASFile.curves.
- **mnemonic** (*str*) – mnemonic of curve.

The index takes precedence over the mnemonic.

`LASFile.`**`append_curve_item`**(*curve_item*)
Add a CurveItem.

Parameters **curve_item** (`lasio.CurveItem`) –

LASFile.**insert_curve_item**(*ix*, *curve_item*)
Insert a CurveItem.

Parameters

- **ix** (*int*) – position to insert CurveItem i.e. 0 for start
- **curve_item** (lasio.CurveItem) –

LASFile.**update_start_stop_step**(*STRT=None*, *STOP=None*, *STEP=None*, *fmt='%.5f'*)
Configure or change STRT, STOP, and STEP values on the LASFile object.

Keyword Arguments

- **STOP, STEP** (*STRT,*) – value to set on the relevant header item in the ~Well section - can be any data type.
- **fmt** (*str*) – Python format string for formatting the STRT/STOP/STEP value in the situation where any of those keyword arguments are None

If STRT/STOP/STEP are not passed to this method, they will be automatically calculated from the index curve.

LASFile.**update_units_from_index_curve**()
Align STRT/STOP/STEP header item units with the index curve's units.

10.5 Writing data out

LASFile.**write**(*file_ref*, ***kwargs*)
Write LAS file to disk.

Parameters **file_ref** (open file-like object or str) – either a file-like object open for writing, or a filename.

All **kwargs are passed to *lasio.writer.write()* – please check the docstring of that function for more keyword arguments you can use here!

Examples

```
>>> import lasio
>>> las = lasio.read("tests/examples/sample.las")
>>> with open('test_output.las', mode='w') as f:
...     las.write(f, version=2.0)    # <-- this method
```

lasio.writer.**write**(*las*, *file_object*, *version=None*, *wrap=None*, *STRT=None*, *STOP=None*, *STEP=None*, *fmt='%.5f'*, *column_fmt=None*, *len_numeric_field=None*, *lhs_spacer=' '*, *spacer=' '*, *data_width=79*, *header_width=60*, *data_section_header='~ASCII'*, *mnemonics_header=False*)
Write a LAS files.

Parameters

- **las** (*lasio.LASFile*) –
- **file_object** (*file-like object open for writing*) – output
- **version** (*float or None*) – version of written file, either 1.2 or 2. If this is None, las.version.VERS.value will be used.
- **wrap** (*bool or None*) – whether to wrap the output data section. If this is None, las.version.WRAP.value will be used.

- **STRT** (*float or None*) – value to use as STRT (note the data will not be clipped). If this is None, the data value in the first column, first row will be used.
- **STOP** (*float or None*) – value to use as STOP (note the data will not be clipped). If this is None, the data value in the first column, last row will be used.
- **STEP** (*float or None*) – value to use as STEP (note the data will not be resampled and/or interpolated). If this is None, the STEP will be estimated from the first two rows of the first column.
- **fmt** (*str*) – Python string formatting operator for numeric data to be used.
- **column_fmt** (*dict or None*) – use this to set a different format string for specific columns from the data ndarray. E.g. to use `'%.3f'` for the depth column and `'%.2f'` for all the other columns, you would use `fmt='%.2f', column_fmt={0: '%.3f'}`.
- **len_numeric_field** (*int*) – width of each numeric field column (must be greater than than all the formatted numeric values in the file). If it is None, the maximum necessary value will be used automatically (i.e. all columns will have the same width). If it is -1, then the columns will not have consistent widths. You can combine -1 with the *fmt* keyword argument to control column widths closely.
- **data_width** (*79*) – width of data field in characters
- **lhs_spacer** (*str*) – string which goes on left hand side of data section - by default it is " ".
- **spacer** (*str*) – string which goes between each column of the data section
- **data_section_header** (*str*) – default "~ASCII"
- **mnemonics_header** (*bool*) – include mnemonic curve names in the data_section_header at the top of data section

Creating an output file is not the only side-effect of this function. It will also modify the STRT, STOP and STEP HeaderItems so that they correctly reflect the ~Data section's units and the actual first, last, and interval values.

However, passing a version to this write() function only changes the version of the object written to. Example: las.write(myfile, version=2). Lasio's internal-las-object version will remain separate and defined by las.version.VERS.value

You should avoid calling this function directly - instead use the *lasio.LASFile.write()* method.

lasio.writer.**get_formatter_function**(*order*, *left_width=None*, *middle_width=None*)

Create function to format a LAS header item for output.

Parameters **order** – format of item, either 'descr:value' or 'value:descr'

Keyword Arguments

- **left_width** (*int*) – number of characters to the left hand side of the first period
- **middle_width** (*int*) – total number of characters minus 1 between the first period from the left and the first colon from the left.

Returns A function which takes a header item (e.g. *lasio.HeaderItem*) as its single argument and which in turn returns a string which is the correctly formatted LAS header line.

lasio.writer.**get_section_order_function**(*section, version, order_definitions={1.2: {'Curves': ['value:descr'], 'Parameter': ['value:descr'], 'Version': ['value:descr'], 'Well': ['descr:value', ('value:descr', ['STRT', 'STOP', 'STEP', 'NULL', 'strt', 'stop', 'step', 'null'])]}, 2.0: {'Curves': ['value:descr'], 'Parameter': ['value:descr'], 'Version': ['value:descr'], 'Well': ['value:descr']}, 2.1: {'Curves': ['value:descr'], 'Parameter': ['value:descr'], 'Version': ['value:descr'], 'Well': ['value:descr']}, 3.0: {'Curves': ['value:descr'], 'Parameter': ['value:descr'], 'Version': ['value:descr'], 'Well': ['value:descr']}}*)

Get a function that returns the order per the mnemonic and section.

Parameters

- **section** (*str*) – either 'well', 'params', 'curves', 'version'
- **version** (*float*) – either 1.2 and 2.0

Keyword Arguments **order_definitions** (*dict*) – see source of defaults.py for more information

Returns A function which takes a mnemonic (str) as its only argument, and in turn returns the order 'value:descr' or 'descr:value'.

lasio.writer.**get_section_widths**(*section_name*, *items*, *version*, *order_func*)

Find minimum section widths fitting the content in *items*.

Parameters

- **section_name** (*str*) – either 'version', 'well', 'curves', or 'params'
- **items** (SectionItems) – section items
- **version** (*float*) – either 1.2 or 2.0
- **order_func** (*func*) – see *lasio.writer.get_section_order_function()*

LASFile.**to_csv**(*file_ref*, *mnemonics=True*, *units=True*, *units_loc='line'*, ***kwargs*)

Export to a CSV file.

Parameters **file_ref** (open file-like object or str) – either a file-like object open for writing, or a filename.

Keyword Arguments

- **mnemonics** (*list, True, False*) – write mnemonics as a header line at the start. If list, use the supplied items as mnemonics. If True, use the curve mnemonics.
- **units** (*list, True, False*) – as for mnemonics.
- **units_loc** (*str or None*) – either 'line', '[]' or '()'. 'line' will put units on the line following the mnemonics (good for WellCAD). '[]' and '()' will put the units in either brackets or parentheses following the mnemonics, on the single header line (better for Excel)
- ****kwargs*** – passed to `csv.writer`. Note that if `lineterminator` is **not** specified here, then it will be sent to `csv.writer` as `lineterminator='\\n'`.

LASFile.**to_excel**(*filename*)

Export LAS file to a Microsoft Excel workbook.

Parameters **filename** (*str*) – a name for the file to be created and written to.

LASFile.**to_json**()

lasio.convert_version.**convert_version**()

lasio.convert_version.**get_convert_version_parser**()

10.6 Defaults

10.7 Custom exceptions

class lasio.exceptions.**LASDataError**
Error during reading of numerical data from LAS file.

class lasio.exceptions.**LASHeaderError**
Error during reading of header data from LAS file.

class lasio.exceptions.**LASUnknownUnitError**
Error of unknown unit in LAS file.

10.8 Test data

lasio.examples.**open**(*filename*, ***kwargs*)
Open an example LAS file from lasio's test suite.

Parameters **filename** (*str*) – forward-slash separated filename of a LAS file from lasio's test suite, starting from the "tests/examples" subfolder e.g. "1001178549.las" or "2.0/sample_2.0.las"

Other keyword arguments are passed to *lasio.LASFile*. If lasio has been installed locally from source, then the local version of the example file will be opened. If lasio has not been installed from source then the LAS file will be downloaded from GitHub.

Returns: LASFile object

lasio.examples.**open_github_example**(*filename*, *url_prefix='https://raw.githubusercontent.com/kinverarity1/lasio/master/t*
***kwargs*)
Open an example LAS file from lasio's test suite on GitHub

Parameters **filename** (*str*) – forward-slash separated filename of a LAS file from lasio's test suite, starting from the "tests/examples" subfolder e.g. "1001178549.las" or "2.0/sample_2.0.las"

Other keyword arguments are passed to *lasio.LASFile*.

Returns: LASFile object

lasio.examples.**open_local_example**(*filename*, ***kwargs*)
Open an example LAS file from lasio's test suite.

Parameters **filename** (*str*) – forward-slash separated filename of a LAS file from lasio's test suite, starting from the "tests/examples" subfolder e.g. "1001178549.las" or "2.0/sample_2.0.las"

Other keyword arguments are passed to *lasio.LASFile*. If lasio has not been installed from source then an exception will be raised.

Returns: LASFile object

lasio.examples.**get_local_examples_path**()
Return the path to the examples from lasio's test suite, if it is installed locally.

Returns: path as str.

10.9 Logging

lasio.**add_logging_level**(*levelName*, *levelNum*, *methodName=None*)
Add a new logging level to current logger.

Comprehensively adds a new logging level to the *logging* module and the currently configured logging class.

levelName becomes an attribute of the *logging* module with the value *levelNum*. *methodName* becomes a convenience method for both *logging* itself and the class returned by *logging.getLoggerClass()* (usually just *logging.Logger*). If *methodName* is not specified, *levelName.lower()* is used.

To avoid accidental clobberings of existing attributes, this method will raise an *AttributeError* if the level name is already an attribute of the *logging* module or if the method name is already present

Example

```
>>> add_logging_level('TRACE', logging.DEBUG - 5)
>>> logging.getLogger(__name__).setLevel("TRACE")
>>> logging.getLogger(__name__).trace('that worked')
>>> logging.trace('so did this')
>>> logging.TRACE
5
```

CHAPTER 11

Contributing to lasio

lasio is an open source project released under the MIT License. It has grown over the years through the wonderful work of many contributors on GitHub, and through many discussions.

Your help is very welcome! No contribution is too small. You can help with the documentation, adding example notebooks, posting ideas or feature requests to GitHub, or by working on the code - or anything else.

11.1 Places you can help

- Please don't hesitate to open a GitHub issue for any problems you are having with lasio, or any ideas for improvements. There are templates to guide you in how to file a bug report, or a request for a new feature or improvement. If you are not sure whether your issue fits under these categories, please go ahead and raise one anyway!
- Please feel free to contribute suggested changes. The easiest method is to fork lasio on GitHub and submit a pull request against the "main" branch. Don't worry about getting all the details right, either way it's still the most convenient way for me or other maintainers to see your changes in context.
- Example LAS files: if you have an interesting/difficult/silly/standards-challenged LAS file (any version) you would be willing to share with me, please do so! Email it to me at kinverarity@hotmail.com. The more examples we can incorporate into lasio's regression testing, the better. If you have concerns about privacy, I'd suggest obfuscating with find-and-replace on various alpha (or numeric) characters before sending it on, and/or deleting any sensitive header lines.

11.2 How to make contributions

Contributions are always welcome to the code, documentation, or example notebooks. If you are making a contribution, please make sure you are working off the latest GitHub main branch. You will want to make your contributions in a branch taken from *main*, and then when you want to share your changes, you can publish them by "pushing" your branch to your GitHub fork of the lasio repository, and opening a PR (pull request) here.

First, create a fork of the lasio repository using the GitHub website. Then clone your fork locally to your computer:

```
$ git clone https://github.com/your-username/lasio
$ cd lasio
```

Your fork will be called the "origin" repository - you'll need to know this for when you push/pull changes to and from your computer.

11.2.1 Adding kinverarity1/lasio as "upstream"

Now also add the kinverarity1/lasio repository as the "upstream" repository. This is so that when other people make changes to kinverarity1/lasio, you can "pull" those changes into your local copy:

```
$ git remote add upstream https://github.com/kinverarity1/lasio
```

To update the *main* branch of the local copy you have of your fork from the "upstream" repository:

```
$ git checkout main
$ git pull upstream main
```

And to update the GitHub fork from your local copy:

```
$ git checkout main
$ git push origin main
```

11.2.2 Making sure you have necessary development dependencies

There are some additional packages you needing for running unit/regression tests (*pytest*) and formatting Python code (*black*). You can install these easily by using:

```
$ pip install --editable ".[test]"
```

11.2.3 Making changes to the code

First, start by making sure your local copy is using the latest copy of code from "upstream" main (see above). Then create a branch - you can call it whatever is meaningful to you. We switch to *main* so that your changes are relative to the latest copy of the code in *main*:

```
$ git checkout main
$ git checkout -b your-branch-name
Switched to a new branch 'your-branch-name'

(your-branch-name) $
```

Then you can make your changes. To test them, make sure you have an "editable" installation of lasio in your Python environment. Shift to the root folder of the repository and run:

```
$ pip install -e .
```

Then to run all the tests:

```
$ pytest
```

Before publishing your changes please make the code is formatted using black:

```
$ black .
```

Then you can push your changes to your fork:

```
$ git push origin your-branch-name
```

And follow the instructions on your fork's GitHub page to open a pull request (PR) for lasio!

11.2.4 Making changes to the documentation

Just as valuable as changes to the code, are changes or improvements to the Sphinx documentation! If you would like to help in this regard, you will need Sphinx and IPython installed:

```
$ pip install sphinx IPython sphinx_rtd_theme
```

Then create a new branch as above. The documentation is written in RestructuredText, and can be found in the *docs/source* subfolder of the lasio repository. If you have any changes, you can build a local copy of the HTML repository to test how it looks. First change into the docs folder:

```
$ cd docs
```

Then run this to generate a local copy of the HTML docs in the *build/html* folder:

```
$ make clean
$ make html
```

Once you are happy, please publish your branch and open a PR in the same way as above.

11.3 Testing

Every time lasio's main branch is updated, automated tests are run using GitHub Actions on Python 3.7, 3.8, 3.9 and 3.10 on Ubuntu and Windows. lasio may work on Python 3.3, 3.4, 3.5, 3.6 but these are not regularly tested.

To run tests yourself:

```
$ pip install "lasio[test]"
$ pytest
```

11.3.1 Comparative benchmarking of performance when reading LAS files

The test file `tests/test_speed.py` reads in a large LAS file and is used to generate the data used in the following benchmark comparisons.

To compare two branches, run and store the benchmark from the first branch e.g. main and generate the benchmark from the second branch e.g. dev-branch. Then run the comparison command.

This same basic technique can be used for testing subsquent changes on a branch.

Make benchmark report for first branch:

```
$ mkdir ../lasio-benchmarks
$ git checkout main
$ pytest tests/test_speed.py --benchmark-autosave --benchmark-storage ../lasio-
↪benchmarks
```

Make benchmark report for second branch.

```
$ git checkout dev-branch
$ pytest tests/test_speed.py --benchmark-autosave --benchmark-storage ../lasio-
↪benchmarks
```

List the available benchmark reports. Their names start with an incremented number: 0001, 0002, etc, followed by their git commit.

```
$ pytest-benchmark  --storage file://../lasio-benchmarks list
<path>/0001_d39237c38dcbd4255ac61708287c5f012f8f56da_20220630_185722.json
<path>/0002_ede364ae5cb8aaa2f821fdda017196121e92ffe6_20220630_194028.json
...
```

Compare two benchmark reports. If the terminal is set to display color then the output will color data green for better performance and red for worse performance.

```
$ pytest-benchmark  --storage file://../lasio-benchmarks compare 0001 0002

-------------------------------------------------------------------------------------------
↪------- benchmark: 2 tests ---------------------------------------------------------------
↪-----------------------------------
Name (time in ms)                                  Min                 Max
↪  Mean            StdDev              Median                IQR            Outliers ⌟ ⌟
↪   OPS            Rounds  Iterations
-------------------------------------------------------------------------------------------
↪-------------------------------------------------------------------------------------------
↪-----------------------------------
test_read_v12_sample_big (0001_d39237c)     149.3796 (1.0)      157.5133 (1.00)
↪150.8693 (1.00)     2.9515 (1.03)     149.5928 (1.0)      0.7392 (2.43)          1;
↪1  6.6283 (1.00)          7           1
test_read_v12_sample_big (0002_ede364a)     149.6045 (1.00)     157.3494 (1.0)
↪150.8314 (1.0)      2.8771 (1.0)      149.7972 (1.00)     0.3038 (1.0)           1;
↪1  6.6299 (1.0)           7           1
-------------------------------------------------------------------------------------------
↪-------------------------------------------------------------------------------------------
↪-----------------------------------
```

11.4 Publishing a new release

1. Ensure you are on main: `$ git checkout main`
2. Ensure you are using the latest copy of main: `$ git pull origin main`
3. Check for any local changes to main: `$ git status` - test locally and push if necessary.
4. Check that GitHub Actions Python CI for main is passing.
5. Find changes since last version release: see list of commits.
6. Summarise these changes in `changelog.rst`.
7. Run the Jupyter Noteook at docs/Add links to GitHub for all issue and PR refs in changelog.ipynb to add hyperlinks for all issue and PR references.
8. Edit the citation file: `CITATION.cff`
9. Commit with a message e.g. `Release v0.31`

10. Tag with the same message e.g. `git tag v0.31`
11. Push to github - first the commit: `git push origin main --tags`
12. Create a universal wheel: `python setup.py bdist_wheel --universal`
13. This will put a new wheel file in `dist/`
14. Also create a source distribution: `python setup.py sdist`
15. This will put a source distribution archive in `dist/`
16. Upload all the new distribution release files (wheel and archive) to PyPI: `twine upload -u USERNAME -p PASSWORD dist/file`
17. Create a new GitHub release via https://github.com/kinverarity1/lasio/releases/new - select the tag
18. Copy the CHANGELOG text in - convert to RST to Markdown quickly by replacing '# with # and removing '_
19. Copy the wheel and source distribution archive files into the release page.
20. Publish the release.

That's it.

11.5 Email

Please feel free to email me at kinverarity@hotmail.com with any suggestions, criticisms, questions, example files.

11.6 Code of Conduct

11.6.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, nationality, personal appearance, race, religion, or sexual identity and orientation.

11.6.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment

- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

11.6.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

11.6.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

11.6.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at kinverarity@hotmail.com. The project team will review and investigate all complaints, and will respond in a way that it deems appropriate to the circumstances. The project team is obliged to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

11.6.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant version 1.4.

CHAPTER 12

List of changes

12.1 Unreleased changes (Available on GitHub)

12.2 Version 0.31 (18 May 2023)

- Many improvements to code style and formatting, and the documentation
- #555 - Fix problem when writing with changed data (different number of depths)
- #554 / #556 - Enable DLM (delimiter) TAB
- #552 - Remove or replace cchardet with chardet
- #530 - Detect hyphens in data section and adjust regexp_subs as needed
- Fix #322 - provide a way to consistently retrieve header items which may or may not be present in the header:

  If you try ordinary item-style access, as is normal in Python, a KeyError exception will be raised if it is missing:

```
>>> permit = las.well['PRMT']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "c:\devapps\kinverarity\projects\lasio\lasio\las_items.py", line 313, in __
↪getitem__
    raise KeyError("%s not in %s" % (key, self.keys()))
KeyError: "PRMT not in ['STRT', 'STOP', 'STEP', 'NULL', 'COMP', 'WELL', 'FLD',
↪'LOC', 'PROV', 'SRVC', 'DATE', 'UWI']"
```

  A better pattern is to use the *lasio.SectionItems.get()* method, which allows you to specify a default value in the case of it missing:

```
>>> permit = las.well.get('PRMT', 'unknown')
>>> permit
HeaderItem(mnemonic="PRMT", unit="", value="unknown", descr="")
```

You can use the `add=True` keyword argument if you would like this header item to be added, as well as returned:

```
>>> permit = las.well.get('PRMT', 'unknown', add=True)
>>> las.well
[HeaderItem(mnemonic="STRT", unit="M", value="0.05", descr="FIRST INDEX VALUE"),
HeaderItem(mnemonic="STOP", unit="M", value="136.6", descr="LAST INDEX VALUE"),
HeaderItem(mnemonic="STEP", unit="M", value="0.05", descr="STEP"),
HeaderItem(mnemonic="NULL", unit="", value="-99999", descr="NULL VALUE"),
HeaderItem(mnemonic="COMP", unit="", value="", descr="COMP"),
HeaderItem(mnemonic="WELL", unit="", value="Scorpio E1", descr="WELL"),
HeaderItem(mnemonic="FLD", unit="", value="", descr=""),
HeaderItem(mnemonic="LOC", unit="", value="Mt Eba", descr="LOC"),
HeaderItem(mnemonic="SRVC", unit="", value="", descr=""),
HeaderItem(mnemonic="CTRY", unit="", value="", descr=""),
HeaderItem(mnemonic="STAT", unit="", value="SA", descr="STAT"),
HeaderItem(mnemonic="CNTY", unit="", value="", descr=""),
HeaderItem(mnemonic="DATE", unit="", value="15/03/2015", descr="DATE"),
HeaderItem(mnemonic="UWI", unit="", value="6038-187", descr="WUNT"),
HeaderItem(mnemonic="PRMT", unit="", value="unknown", descr="")]
```

12.3 Version 0.30 (12 May 2022)

- Fixes for #446 (Implement a numpy-based reader for the data section; **‘#452‘_**) and #444 (Provide a way to benchmark speed performance of LASFile.read; **‘#449‘_**). This leads to a more than two-fold improvement in speed (e.g. from 1091 msec to 341 msec for a large sample file).
- Fixes for #439 (Data section containing non-numeric chars is parsed entirely as str) and #227 (NULL value replacing valid value in DEPT) by allowing different data types for different columns (PRs **‘#459‘_**, **‘#460‘_**, **‘#461‘_**)
- Partially fix #375 (allow writing mnemonics in ~ASCII line; PRs **‘#466‘_**, **‘#471‘_**, **‘#465‘_**)
- Partial fix for #265 (Parse comma-delimited ~ASCII sections; **‘#485‘_**)
- Fix #83 (LAS file with more curves defined in ~C section than available in ~A section; **‘#480‘_**)
- Improve visibility and access to the Code of Conduct (**‘#469‘_**)
- Fix #453 (rounding issues when writing LAS file) (**‘#455‘_**)
- Fix #268 (remove side-effects from writer) (**‘#447‘_**, **‘#451‘_**)
- Documentation updates (**‘#475‘_**, #477, **‘#481‘_**, **‘#501‘_**, **‘#498‘_**, **‘#500‘_**)
- Fix #473 (Header lines without a period are discarded unnecessarily)
- Fix #472 (Detect and raise IOError exception for LiDAR files; **‘#482‘_**)
- Fix #483 (perfomance decrease in DEBUG logging level; **‘#484‘_**)
- Fix #478 (:1 :2 should only be appended to mnemonics when using LASFile.read) with **‘#479‘_**: Add update_curve and replace_curve_item methods to LASFile
- Fix #490 (LAS 2.1 reading error; **‘#491‘_**)
- Fix #486 (Adding a citation file, and maybe a DOI; **‘#487‘_**)
- Fix #392 (ignore_comments not mentioned in documentation; **‘#495‘_**)
- Fix #332: Describe default read_policies in docstring, with **‘#489‘_**

- Fix #502 (NumPy type conversion alias deprecation; **‘#503‘**_)
- minor performance improvements (**‘#470‘**_,)

12.4 Version 0.29 (14 April 2021)

- Fix #404 (lasio changes STEP with imprecise floating-point number behaviour; **‘#432‘**_)
- Add option `len_numeric_field=-1`, `lhs_spacer=" "`, and `spacer=" "` to writer.py:write (see #412; PR **‘#418‘**_)
- Fix #271 (Read quoted strings in data section properly; **‘#423‘**_)
- Fix #427 (Change null_policy to handle small non-zero values; **‘#429‘**_)
- Fix #417 (Fix parsing for empty ~Other section; **‘#430‘**_)
- Fix #402 (fixes issue when unit starts with dot; **‘#403‘**_)
- Fix #395 (Update doc examples to reflect new HeaderItem repr; **‘#410‘**_)
- Fix #426 (Update urllib.request to be the preferred urllib; **‘#428‘**_)
- Add check for pushed tag version to version tests (**‘#396‘**_)
- Update GitHub Action Python CI testing (**‘#399‘**_, **‘#400‘**_)
- Improve `las_items.py:HeaderItem.__repr__` truncation logic (**‘#397‘**_)
- Remove codecs import (unused) and fix typo (**‘#406‘**_)
- Exclude LAS files from GitHubs Language Stats (**‘#411‘**_)
- Re-add try-except check around call to reader.read_data_section_iterative() (**‘#401‘**_)
- Remove reader.py:read_file_contents - unused code (see **‘#401‘**_; **‘#393‘**_)
- Add test for timestring with colon in ~Well section (see #419 - PR **‘#420‘**_)
- Fix SyntaxWarning in writer.py (**‘#425‘**_)
- Add bugfix and feature request issue templates to GitHub repository
- Apply `black` code style to all Python files (**‘#438‘**_, **‘#398‘**_)
- Update demo notebook for using logging levels with current behaviour
- Update contributing guide (**‘#437‘**_, **‘#441‘**_)

12.5 Version 0.28 (12 September 2020)

- Major re-write of reader code working towards LAS 3.0 support (**‘#327‘**_; **‘#347‘**_, **‘#345‘**_, **‘#353‘**_, **‘#355‘**_, **‘#358‘**_, **‘#367‘**_, **‘#368‘**_, **‘#369‘**_)
- Fix #377 (writing “None” as the value instead of “”; #377)
- Fix #373 (enable GitHub Actions CI testing on MacOS, Windows, Ubuntu; **‘#374‘**_, **‘#387‘**_)
- Fix #363 (parse composite units such as “1000 lbf” correctly; **‘#390‘**_)
- Fix #319 (allow skipping comment lines in data sections; **‘#391‘**_)
- Avoid unnecessary exceptions on reading LAS 3.0 data sections (**‘#385‘**_)

- Fix broken ReadTheDocs build

12.6 Version 0.27 (4 September 2020)

- Fix #380 (install failed without git installed; **‘#382‘_**)

12.7 Version 0.26 (31 August 2020)

- This is the final version which works on Python 2.7 (#364)
- Fix #333 (header lines not parsed when colon is in description; **‘#335‘_**)
- Fix #359 (sections not found when leading whitespace in line; **‘#360‘_**, **‘#361‘_**)
- Fix #350 (bug with NULL; **‘#352‘_**)
- Fix #339 (0.1IN not recognised as index unit; **‘#340‘_**, **‘#349‘_**)
- Fix #31 (add command-line script to convert between LAS versions; **‘#329‘_**)
- Fix #75 (add Cyrillic variant for metres; **‘#330‘_**)
- Fix **‘#326‘_** (Support header-only LAS files–don’t lose the last header section before a missing ~A section)
- Improve documentation regarding deleting items and curves (#315, **‘#325‘_**)
- Add deprecation markers (**‘#331‘_**)
- Align json.dumps and LASFile.to_json() (**‘#328‘_**)
- Fixes and updates to setup.py relating to the adoption of setuptools_scm (#312, **‘#317‘_**, **‘#318‘_**)
- Clean up and background changes related to future LAS 3.0 support: **‘#334‘_**, **‘#337‘_**, **‘#338‘_**, **‘#341‘_**, **‘#342‘_**, **‘#346‘_**, **‘#348‘_**, **‘#372‘_**

12.8 Version 0.25.1 (1 May 2020)

- Shift to setuptools_scm (**‘#311‘_**)
- Fix #321 (EOF character causes error on read)
- Fix #182 (remove side-effect LASFile.write causing LASFile.version.VERS to change)
- Fix #310 (remove LASFile.metadata which was not working)

12.9 Version 0.25 (28 March 2020)

- Add stack_curves() method to allow joining a set of curves into a 2D array (issue #284, PR **‘#293‘_**)
- Add lasio.examples module (**‘#296‘_**)
- Fix #278 (leading zeroes were being stripped from API/UWI numbers)
- Fix **‘#286‘_** (error on trying to write a file with one row of data)
- Fix **‘#258‘_** (do not catch Ctrl+C when reading file)

- Fix **'#292'**_ (improve error checking for when trying to write non-2D data)
- Fix #277 (allow pathlib objects to lasio.read)
- Fix #264 (allow periods in mnemonics to be retained in specific cases)
- Fix #201 (adjust descr parsing in ~P section to allow times in the descr, see PR **'#298'**_)
- Fix **'#302'**_ (change in str(datetime) handling)
- Fixes to JSON output (**'#300'**_, #303)
- Fix #304 (add column_fmt argument to LASFile.write method)

12.10 Version 0.24

- Fix #256 (parse units in brackets and add index_unit kwarg)

12.11 Version 0.23

- Fix #259 (error when encoding missing from URL response headers)
- Fix #262 (broken build due to cchardet dependency)

12.12 Version 0.22

- Fix **'#252'**_ (removing case sensitivity in index_unit checks)
- Fix **'#249'**_ (fix bug producing df without converting to floats)
- Attempt to fix Lasso classification on GitHub

12.13 Version 0.21

- Fix #236 and #237 (can now read ASCII in ~Data section)
- Fix **'#239'**_ (Petrel can't read lasio output)

12.14 Version 0.20

- Fix #233 (pickling error lost Curve.data during multiprocessing)
- Fix #226 (do not issue warning on empty ~Parameter section)
- Revised default behaviour to using null_policy='strict' (ref. #227)
- Fix **'#221'**_ (depths > 10000 were being rounded by default)
- Fix **'#225'**_ (file handle leaked if exception during parsing)

12.15 Version 0.19

- Fix #223 (critical version/installation bug)

12.16 Version 0.18

- Fix version numbering setup
- Fix #92 (can ignore blah blah lines in ~C section)
- Fix #209 (can now add curves with LASFile['mnemonic'] = [1, 2, 3])
- Fix #213 (LASFile.data is now a lazily generated property, with setter)
- Fix #218 (LASFile.append_curve was not adding data=[...] properly)
- Fix #216 (LASFile now raises KeyError for missing mnemonics)
- Fix #214 (first duplicate mnemonic when added was missing the :1)

12.17 Version 0.17

- Add Appveyor continuous integration testing
- Add example notebook for how to use python logging module
- Fix #160 (add methods to LASFile for inserting curves)
- Fix #155 (implement del keyword for header items)
- Fix #142 (implement slicing for SectionItems)
- Fix #135 (UWI numbers losing their leading zeros)
- Fix #153 (fix SectionItems pprint repr in Python 3)
- Fix #81 (accept header items with missing colon)
- Fix #71 (add Docker build for lasio to DockerHub)
- Fix #210 (allow upper/lowercase standardization of mnemonics on read)
- Document recent additions (nearly up to date) (in Sphinx docs)

12.18 Version 0.16

- Add read_policy and null_policy keywords - see documentation for details
- Fix bugs around files with missing ~V ~W ~P or ~C sections (#84 #85 #78)
- Fix #17 involving files with commas as a decimal mark
- Improve LASHeaderError traceback message
- Fix bug involving files with ~A but no data lines following
- Fix bug with blank line at start of file
- Fix bug involving missing or duplicate STRT, STOP and STEP mnemonics

12.19 Version 0.15.1

- Major performance improvements with both memory and speed
- Major improvement to read parser, now using iteration
- Add `LASFile.to_excel()` and `LASFile.to_csv()` export methods
- Improve `las2excelbulk.py` script
- Published new and updated Sphinx documentation
- Improved character encoding handling when `chardet` not installed
- `autodetect_encoding=True` by default
- Allow reading of multiple non-standard header sections (#167, **'#168'**_)
- Add flexibility in reading corrupted headers (`ignore_header_errors=True`)
- Add ability to avoid reading in data (`ignore_data=True`)
- Remove excessive debugging messages
- Fix bug #164 where `FEET` was not recognised as `FT`
- Fix major globals() bug #141 affecting LASFile.add_curve
- Add command-line version script `$ lasio` to show version number.

Version 0.14 and 0.15 skipped due to broken PyPI upload.

12.20 Version 0.13

- Other minor bug fixes inc inability to rename mnemonics in written LAS file.

12.21 Version 0.11.2

- Fix bug with not correctly figuring out units for LASFile.write()
- Add `LASFile.add_curve(CurveItem)` method which automatically goes to the old method at `LASFile.add_curve_raw(mnemonic=, data=, ...)` if necessary, so it should be transparent to users

12.22 Version 0.11

- Reorganise code into modules
- various

12.23 Version 0.10

- Internal change to SectionItems for future LAS 3.0 support
- Added JSON encoder

- Added examples for using pandas DataFrame (.df attribute)
- LAS > Excel script refined (las2excel.py)

12.24 Version 0.9.1 (2015-11-11)

- pandas.DataFrame now as .df attribute, bugfix

12.25 Version 0.8 (2015-08-20)

- numerous bug fixes, API documentation added

12.26 Version 0.7 (2015-08-08)

- all tests passing on Python 2.6 through 3.4

12.27 Version 0.6 (2015-08-05)

- bugfixes and renamed from `las_reader` to `lasio`

12.28 Version 0.5 (2015-08-01)

- Improvements to writing LAS files

12.29 Version 0.4 (2015-07-26)

- Improved handling of character encodings, other internal improvements

12.30 Version 0.3 (2015-07-23)

- Added Python 3 support, now reads LAS 1.2 and 2.0

12.31 Version 0.2 (2015-07-08)

- Tidied code and published on PyPI

CHAPTER 13

Indices and tables

- genindex
- search

Python Module Index

l

Index

Symbols

A

C

D

E

G

H

I

K

L

M

O

P

R

S

T

U

V

W