lasio Documentation

Release latest

Kent Inverarity

Mar 18, 2018

Contents

This is a Python 2/3 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. In principle it is designed to read as many types of LAS files as possible, including ones containing common errors or non-compliant formatting.

Depending on your particular application you may also want to check out striplog for stratigraphic/lithological data, or (still in alpha dev) welly for dealing with data at the well level. lasio is primarily for reading & writing LAS files.

Note this is *not* a package for reading LiDAR data (also called "LAS files").

CHAPTER 1

Installation

lasio is written to be compatible with Python 2.6+, and 3.2+. The best way to install is using pip.

```
(test) C:\Users\kent>pip install lasio
```

This will download and install lasio's dependencies (numpy and ordereddict).

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

```
(test) C:\Users\kent\Code\lasio>pip install -r optional-packages.txt
```

lasio is now installed. See the following pages for examples of how to use the package.

To upgrade to the latest PyPI version, use:

```
(test2) C:\Users\kent\Code\testing\lasio>pip install --upgrade lasio
```

1.1 Development version

Installing via pip gets the latest release which has been published on PyPI.

The source code for lasio is kept at:

https://github.com/kinverarity1/lasio

Updates are made much more frequently to the master branch here. If you have Git installed, you can keep up to date with these changes:

```
(test2) C:\Users\kent\Code\testing>git clone https://github.com/kinverarity1/lasio

(test2) C:\Users\kent\Code\testing>cd lasio

```

```
(test2) C:\Users\kent\Code\testing\lasio>pip install -r requirements.txt

(test2) C:\Users\kent\Code\testing\lasio>python setup.py develop
```

To update your version with the latest changes on GitHub:

```
(test2) C:\Users\kent\Code\testing\lasio>git pull origin master
```

1.2 Testing

Every time `lasio` is updated, automated tests are run:

- Travis CI: Linux, Python versions 2.7, 3.3, 3.4, 3.5, and 3.6.
- BUILDING... Appveyor CI: Windows, Python versions 2.7, 3.4, 3.5, and 3.6.

`lasio` should also work on Python 2.6 and 3.2, but these are tested only occassionally.

To run tests yourself, first install the testing framework and all the optional packages:

```
(test2) C:\Users\kent\Code\testing\lasio>pip install pytest

(test2) C:\Users\kent\Code\testing\lasio>pip install -r optional-packages.txt
```

And then run tests:

```
(test2) C:\Users\kent\Code\testing\lasio>py.test
```

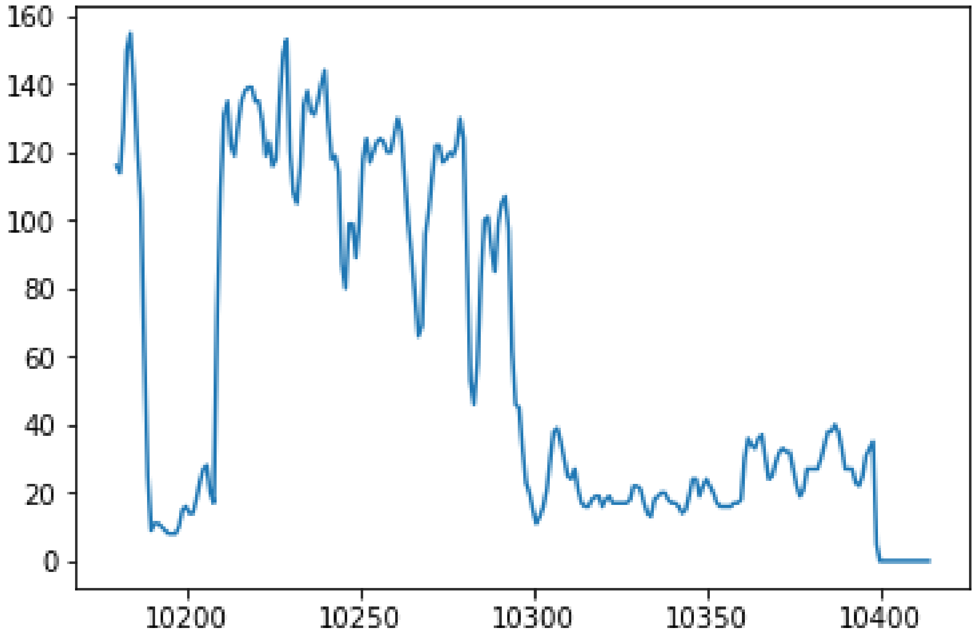
CHAPTER 2

Basic example

In the example below you can see how to:

- read a LAS file in
- look at the information in the header
- see basic curve information
- make a graph

```
In [29]: import lasio

In [30]: las = lasio.read(r"C:\Users\kent\Code\las\examples\2.0\49-005-30258.las")

In [31]: las.header
Out[31]:
{'Curves': [CurveItem(mnemonic=DEPT, unit=F, value=, descr=1 DEPTH, original_
↪mnemonic=DEPT, data.shape=(235,)),
  CurveItem(mnemonic=DT, unit=US/F, value=, descr=2 SONIC DELTA-T, original_
↪mnemonic=DT, data.shape=(235,)),
  CurveItem(mnemonic=RESD, unit=OHMM, value=, descr=3 DEEP RESISTIVITY, original_
↪mnemonic=RESD, data.shape=(235,)),
  CurveItem(mnemonic=SP, unit=MV, value=, descr=4 SP CURVE, original_mnemonic=SP,␣
↪data.shape=(235,)),
  CurveItem(mnemonic=GR, unit=GAPI, value=, descr=5 GAMMA RAY, original_mnemonic=GR,␣
↪data.shape=(235,))],
 'Other': '',
 'Parameter': [HeaderItem(mnemonic=BHT, unit=DEGF, value=194.0, descr=BOTTOM HOLE␣
↪TEMPERATURE, original_mnemonic=BHT),
  HeaderItem(mnemonic=RMF, unit=OHMM, value=0.441, descr=MUD FILTRATE RESISTIVITY,␣
↪original_mnemonic=RMF),
  HeaderItem(mnemonic=RMFT, unit=DEGF, value=68.0, descr=MEASURE TEMPERATURE OF RMF,␣
↪original_mnemonic=RMFT),
  HeaderItem(mnemonic=EKB, unit=F, value=4642.0, descr=ELEVATION KELLY BUSHING,␣
↪original_mnemonic=EKB),
  HeaderItem(mnemonic=SECT, unit=, value=36, descr=SECTION, original_mnemonic=SECT),
  HeaderItem(mnemonic=TOWN, unit=, value=47N, descr=TOWNSHIP, original_mnemonic=TOWN),
  HeaderItem(mnemonic=RANG, unit=, value=71W, descr=RANGE, original_mnemonic=RANG)],
 'Version': [HeaderItem(mnemonic=VERS, unit=, value=2.0, descr=CWLS log ASCII␣
↪Standard - Version 2.0, original_mnemonic=VERS),
  HeaderItem(mnemonic=WRAP, unit=, value=NO, descr=One Line per Depth Step, original_
↪mnemonic=WRAP),
  HeaderItem(mnemonic=CREA, unit=, value=02-08-2006, descr=LAS File Creation Date (MM-
↪DD-YYYY), original_mnemonic=CREA)],
 'Well': [HeaderItem(mnemonic=STRT, unit=F, value=10180.0, descr=START DEPTH,␣
↪original_mnemonic=STRT),
  HeaderItem(mnemonic=STOP, unit=F, value=10414.0, descr=STOP DEPTH, original_
↪mnemonic=STOP),
  HeaderItem(mnemonic=STEP, unit=F, value=1.0, descr=STEP, original_mnemonic=STEP),
  HeaderItem(mnemonic=NULL, unit=, value=-999.25, descr=NULL VALUE, original_
↪mnemonic=NULL),
  HeaderItem(mnemonic=COMP, unit=, value=Cramer Oil, descr=COMPANY, original_
↪mnemonic=COMP),
  HeaderItem(mnemonic=WELL, unit=, value=#36-16 State, descr=WELL, original_
↪mnemonic=WELL),
  HeaderItem(mnemonic=LOC, unit=, value=SE SE 36-47N-71W, descr=LOCATION, original_
↪mnemonic=LOC),
  HeaderItem(mnemonic=CNTY, unit=, value=Campbell, descr=COUNTY, original_
↪mnemonic=CNTY),
  HeaderItem(mnemonic=FLD, unit=, value=, descr=FIELD, original_mnemonic=FLD),
  HeaderItem(mnemonic=STAT, unit=, value=Wyoming, descr=STATE, original_
↪mnemonic=STAT),
  HeaderItem(mnemonic=CTRY, unit=, value=U.S.A., descr=COUNTRY, original_
↪mnemonic=CTRY),
  HeaderItem(mnemonic=DATE, unit=, value=11/91, descr=COMPLETION DATE (MM/YY),␣
↪original_mnemonic=DATE),
  HeaderItem(mnemonic=API, unit=, value=49-005-30258-0000, descr=API NUMBER, original_
↪mnemonic=API),
  HeaderItem(mnemonic=SRVC, unit=, value=, descr=SERVICE COMPANY, original_
↪mnemonic=SRVC)]}

In [33]: type(las.data)
Out[33]: numpy.ndarray

In [34]: las.data.shape
Out[34]: (235, 5)

In [35]: for curve in las.curves:
    ...:     print(curve.mnemonic)
    ...:     print(curve.unit)
    ...:     print(curve.data)
    ...:     print("\n")
    ...:
DEPT
F
[ 10180.  10181.  10182.  10183.  10184.  10185.  10186.  10187.  10188.
  10189.  10190.  10191.  10192.  10193.  10194.  10195.  10196.  10197.
  10198.  10199.  10200.  10201.  10202.  10203.  10204.  10205.  10206.
  10207.  10208.  10209.  10210.  10211.  10212.  10213.  10214.  10215.
  10216.  10217.  10218.  10219.  10220.  10221.  10222.  10223.  10224.
  10225.  10226.  10227.  10228.  10229.  10230.  10231.  10232.  10233.
  10234.  10235.  10236.  10237.  10238.  10239.  10240.  10241.  10242.
  10243.  10244.  10245.  10246.  10247.  10248.  10249.  10250.  10251.
  10252.  10253.  10254.  10255.  10256.  10257.  10258.  10259.  10260.
  10261.  10262.  10263.  10264.  10265.  10266.  10267.  10268.  10269.
  10270.  10271.  10272.  10273.  10274.  10275.  10276.  10277.  10278.
  10279.  10280.  10281.  10282.  10283.  10284.  10285.  10286.  10287.
  10288.  10289.  10290.  10291.  10292.  10293.  10294.  10295.  10296.
  10297.  10298.  10299.  10300.  10301.  10302.  10303.  10304.  10305.
  10306.  10307.  10308.  10309.  10310.  10311.  10312.  10313.  10314.
  10315.  10316.  10317.  10318.  10319.  10320.  10321.  10322.  10323.
  10324.  10325.  10326.  10327.  10328.  10329.  10330.  10331.  10332.
  10333.  10334.  10335.  10336.  10337.  10338.  10339.  10340.  10341.
  10342.  10343.  10344.  10345.  10346.  10347.  10348.  10349.  10350.
  10351.  10352.  10353.  10354.  10355.  10356.  10357.  10358.  10359.
  10360.  10361.  10362.  10363.  10364.  10365.  10366.  10367.  10368.
  10369.  10370.  10371.  10372.  10373.  10374.  10375.  10376.  10377.
  10378.  10379.  10380.  10381.  10382.  10383.  10384.  10385.  10386.
  10387.  10388.  10389.  10390.  10391.  10392.  10393.  10394.  10395.
  10396.  10397.  10398.  10399.  10400.  10401.  10402.  10403.  10404.
  10405.  10406.  10407.  10408.  10409.  10410.  10411.  10412.  10413.
  10414.]


DT
US/F
[ 59.9  59.9  60.5  63.5  64.5  64.6  61.5  59.2  55.9  52.1  49.1  47.8
  47.2  47.2  48.5  49.6  48.3  46.9  46.6  46.8  46.7  47.8  51.2  51.6
  51.1  51.4  52.3  52.3  51.5  51.2  53.3  57.6  60.6  60.8  59.5  59.7
  61.1  61.6  61.8  62.   62.2  62.2  62.2  60.9  60.8  61.5  61.4  61.9
  63.2  64.4  62.6  61.4  61.   61.1  62.8  65.4  66.3  66.2  68.3  69.8
  70.6  72.4  74.2  74.3  71.5  63.5  60.1  65.2  68.2  66.4  63.2  63.4
  65.3  65.1  64.1  63.9  63.9  63.9  63.9  63.5  62.7  63.1  63.6  61.1
  58.4  58.1  58.1  57.7  57.1  56.6  56.8  59.5  61.3  61.9  61.9  62.1
  62.5  62.5  62.5  62.4  62.   60.7  57.5  56.   56.   57.8  60.   60.3
  60.2  59.9  60.4  60.9  61.4  61.4  56.1  51.2  48.4  48.5  49.8  49.8
  50.   50.9  50.5  47.9  46.3  46.1  46.4  46.4  45.8  45.9  46.5  46.7
  47.3  51.9  55.7  61.2  66.5  68.9  69.6  69.6  69.1  68.   66.9  66.7
  66.6  66.   65.   64.4  64.   64.6  64.7  64.4  64.4  65.5  67.4  69.3
  70.9  72.4  73.3  73.7  73.8  73.4  73.4  74.4  75.4  75.2  72.6  71.6
  72.   74.3  74.6  74.7  72.3  71.9  75.5  77.6  78.3  75.8  73.8  71.6
  69.3  67.1  65.   64.   63.8  63.9  65.1  65.5  64.3  64.4  66.   66.
  64.6  64.9  65.   62.6  60.4  59.3  59.3  62.6  63.6  61.5  61.7  62.3
  61.9  62.3  63.2  63.5  63.5  62.7  60.   57.   54.   49.1  47.2  46.7
  47.1  47.6  48.8  49.8  50.8  51.1  50.2  49.   48.4  50.6  50.7  50.4
  49.9  49.7  49.6  51.5  52.5  53.2  54.1]


RESD
OHMM
[  2.20000000e+01   2.10000000e+01   1.97000000e+01   1.89000000e+01
   1.82000000e+01   1.80000000e+01   1.80000000e+01   2.10000000e+01
   2.90000000e+01   5.30000000e+01   3.90000000e+02   1.50100000e+03
   2.09300000e+03   1.67700000e+03   1.07700000e+03   7.65000000e+02
   5.64000000e+02   5.54000000e+02   4.87000000e+02   1.59000000e+02
   7.40000000e+01   5.70000000e+01   5.00000000e+01   4.80000000e+01
   4.80000000e+01   4.90000000e+01   5.60000000e+01   5.90000000e+01
   6.10000000e+01   5.20000000e+01   2.40000000e+01   1.75000000e+01
   1.54000000e+01   1.52000000e+01   1.52000000e+01   1.52000000e+01
   1.47000000e+01   1.29000000e+01   1.20000000e+01   1.10000000e+01
   1.06000000e+01   1.05000000e+01   1.05000000e+01   1.08000000e+01
   1.11000000e+01   1.12000000e+01   1.07000000e+01   9.90000000e+00
   9.30000000e+00   9.00000000e+00   9.40000000e+00   1.01000000e+01
   1.02000000e+01   1.00000000e+01   8.00000000e+00   7.10000000e+00
   6.50000000e+00   5.80000000e+00   5.00000000e+00   4.20000000e+00
   3.60000000e+00   3.30000000e+00   3.20000000e+00   3.30000000e+00
   4.00000000e+00   4.90000000e+00   5.40000000e+00   5.80000000e+00
   6.20000000e+00   6.60000000e+00   7.60000000e+00   8.90000000e+00
   1.01000000e+01   1.12000000e+01   1.24000000e+01   1.51000000e+01
   1.66000000e+01   1.75000000e+01   1.80000000e+01   1.80000000e+01
   1.80000000e+01   1.80000000e+01   1.90000000e+01   2.10000000e+01
   2.30000000e+01   2.70000000e+01   3.00000000e+01   3.30000000e+01
   3.50000000e+01   3.50000000e+01   3.00000000e+01   2.70000000e+01
   2.30000000e+01   1.99000000e+01   1.89000000e+01   1.85000000e+01
   1.94000000e+01   2.00000000e+01   2.00000000e+01   2.20000000e+01
   2.40000000e+01   2.60000000e+01   3.00000000e+01   3.30000000e+01
   3.40000000e+01   3.00000000e+01   2.80000000e+01   2.60000000e+01
   2.60000000e+01   2.90000000e+01   3.40000000e+01   3.50000000e+01
   3.90000000e+01   4.40000000e+01   6.60000000e+01   1.22000000e+02
   2.48000000e+02   1.72400000e+03   2.03600000e+03   2.03600000e+03
   2.05500000e+03   2.09300000e+03   2.11300000e+03   2.11300000e+03
   2.11300000e+03   2.09300000e+03   1.63100000e+03   7.51000000e+02
   2.50000000e+02   2.16000000e+02   1.99000000e+02   1.76000000e+02
   1.30000000e+02   9.50000000e+01   6.90000000e+01   4.70000000e+01
   3.10000000e+01   2.10000000e+01   1.75000000e+01   1.61000000e+01
   1.61000000e+01   1.61000000e+01   1.75000000e+01   1.80000000e+01
   1.83000000e+01   1.83000000e+01   1.83000000e+01   1.83000000e+01
   1.82000000e+01   1.74000000e+01   1.63000000e+01   1.54000000e+01
   1.40000000e+01   1.27000000e+01   1.10000000e+01   9.00000000e+00
   7.50000000e+00   6.70000000e+00   6.10000000e+00   5.70000000e+00
   5.60000000e+00   5.30000000e+00   5.00000000e+00   4.50000000e+00
   4.00000000e+00   3.50000000e+00   3.20000000e+00   2.80000000e+00
   2.50000000e+00   2.20000000e+00   1.94000000e+00   1.72000000e+00
   1.59000000e+00   1.50000000e+00   1.43000000e+00   1.37000000e+00
   1.34000000e+00   1.34000000e+00   1.38000000e+00   1.59000000e+00
   2.00000000e+00   2.90000000e+00   3.30000000e+00   3.80000000e+00
   4.50000000e+00   5.00000000e+00   5.30000000e+00   5.50000000e+00
   5.60000000e+00   5.60000000e+00   5.70000000e+00   5.70000000e+00
   5.70000000e+00   5.70000000e+00   5.80000000e+00   6.30000000e+00
   7.20000000e+00   8.10000000e+00   8.30000000e+00   8.30000000e+00
   8.10000000e+00   8.00000000e+00   8.80000000e+00   1.00000000e+01
   1.01000000e+01   9.20000000e+00   8.60000000e+00   8.50000000e+00
   9.40000000e+00   1.14000000e+01   1.48000000e+01   1.90000000e+01
   4.00000000e+01   8.90000000e+01   1.34000000e+02   2.20000000e+02
   1.93000000e+02   1.22000000e+02   9.60000000e+01   8.10000000e+01
   7.50000000e+01   7.50000000e+01   9.70000000e+01   1.67000000e+02
   3.15000000e+02   1.69300000e+03   1.87400000e+03   1.87400000e+03
   1.87400000e+03   5.91000000e+02   2.08000000e+02   1.34000000e+02
   1.16000000e+02   1.13000000e+02   1.56000000e+02]


SP
MV
[  45.6   49.    53.    55.6   58.4   62.5   64.7   66.9   69.3   71.3
   73.7   75.7   76.7   77.1   77.5   77.5   77.5   77.5   77.1   76.5
   75.9   74.7   73.7   71.1   67.3   63.7   60.6   57.8   53.2   48.2
   42.9   37.9   34.5   31.7   30.1   28.5   27.2   26.    24.4   23.4
   22.2   21.2   20.8   20.2   19.8   19.2   19.    19.    18.8   18.6
   18.4   18.2   18.2   18.2   18.2   18.2   18.8   19.4   20.4   21.6
   22.4   23.8   25.    26.4   28.    29.3   30.5   31.3   32.7   33.5
   34.3   34.9   35.7   36.1   36.1   36.3   36.3   36.3   36.3   36.3
   36.3   36.3   36.3   36.3   36.9   37.9   38.5   39.7   40.1   40.5
   40.5   40.7   40.9   40.9   41.1   41.1   41.1   41.3   41.7   42.1
   42.9   43.9   44.7   45.6   46.6   47.2   48.    48.6   50.2   51.2
   52.8   54.2   55.2   56.    57.2   58.    59.    59.6   60.8   62.3
   63.5   64.7   66.3   67.7   68.9   70.7   72.1   73.1   74.5   75.9
   77.3   78.8   80.8   83.4   85.4   87.6   89.8   91.4   93.4   94.2
   95.1   95.5   95.5   95.5   95.5   95.7   96.5   96.7   96.9   97.1
   97.1   96.7   95.9   95.1   94.8   94.8   94.8   94.8   95.5   96.3
   97.1   98.1   99.1   99.5   99.9  100.1  100.1  100.1   99.9   99.9
  100.1  101.9  102.1  103.9  104.9  105.7  106.1  106.3  106.5  106.5
  106.5  106.5  106.5  105.9  105.3  104.7  104.7  104.5  104.5  104.5
  104.5  104.7  105.9  106.9  109.1  109.7  108.7  107.9  107.3  106.9
  106.7  106.7  106.5  105.9  105.1  102.7   87.6   78.8   76.5   76.5
   76.5   76.5   76.5   76.5   75.5   74.1   72.1   70.5   68.9   67.5
   66.7   65.7   65.1   64.1   63.3   63.1   62.7   62.5   62.5   62.5
   62.5   62.5   62.9   64.3   65.9]


GR
GAPI
[ 116.  114.  127.  150.  155.  140.  121.  106.   62.   25.    9.   11.
   11.   10.    9.    8.    8.    8.   10.   15.   16.   14.   14.   18.
   23.   27.   28.   20.   17.   72.  109.  131.  135.  122.  119.  128.
  135.  138.  139.  139.  135.  135.  129.  119.  123.  116.  118.  135.
  149.  153.  120.  108.  105.  116.  134.  138.  132.  131.  135.  140.
  144.  129.  118.  119.  114.   87.   80.   99.   99.   89.  100.  118.
  124.  117.  120.  123.  124.  123.  120.  120.  125.  130.  126.  113.
  100.   90.   78.   66.   69.   96.  103.  113.  122.  122.  117.  118.
  120.  119.  122.  130.  124.   87.   53.   46.   58.   83.  100.  101.
   92.   85.   99.  105.  107.   97.   62.   46.   45.   33.   23.   20.
   15.   11.   13.   16.   21.   30.   38.   39.   35.   30.   25.   24.
   27.   21.   17.   16.   16.   18.   19.   19.   16.   18.   19.   17.
   17.   17.   17.   17.   18.   22.   22.   21.   17.   14.   13.   18.
   19.   20.   20.   18.   17.   17.   16.   14.   15.   19.   24.   24.
   19.   22.   24.   22.   20.   17.   16.   16.   16.   16.   17.   17.
   18.   30.   36.   34.   33.   36.   37.   30.   24.   25.   29.   32.
   33.   32.   32.   27.   22.   19.   21.   27.   27.   27.   27.   30.
   34.   38.   38.   40.   38.   33.   27.   27.   27.   23.   22.   25.
   31.   33.   35.    5.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.]



In [36]: import matplotlib.pyplot as plt

In [37]: %matplotlib inline

In [38]: plt.plot(las.index, las["GR"])
Out[38]: [<matplotlib.lines.Line2D at 0xb9dc1d0>]
```


160
140
120
100
80
60
40
20
0
10200
10250
10300
10350
10400

CHAPTER 3

Integration with pandas.DataFrame

The `lasio.LASFile.df()` method converts the LAS data to a `pandas.DataFrame`.

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

```
In [168]: las = lasio.read('tests/examples/6038187_v1.2.las')

In [169]: df = las.df()
```

There are some summary methods handy for data exploration:

```
In [170]: df.head(10)
Out[170]:
        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

In [171]: df.tail(40)
Out[171]:
           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
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

In [172]: df.describe()
Out[172]:
              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
```

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

```
In [44]: import numpy as np

In [173]: df['GAMN'][df['GAMN'] == -2324.28] = np.nan

In [174]: df.describe()['GAMN']
Out[174]:
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:

```
In [175]: 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:

```
In [176]: las.set_data(df)

In [177]: las.curves
Out[177]:
[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,))]

In [178]: las.df().describe()
Out[178]:
              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 fairly well for LAS 1.2 and 2.0 files, and lasio does not require LAS files to be strictly compliant with either standard.

```
In [179]: import lasio

In [180]: las = lasio.read('tests/examples/6038187_v1.2_short.las')
```

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

```
In [206]: type(las.sections)
Out[206]: dict

In [207]: las.sections.keys()
Out[207]: 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.las_items.SectionItems*	`LASFile.version` and `LASFile.sections['Version']`
`~w` or `~W`	*lasio.las_items.SectionItems*	`LASFile.well` and `LASFile.sections['Well']`
`~c` or `~C`	*lasio.las_items.SectionItems*	`LASFile.curves` and `LASFile.sections['Curves']`
`~p` or `~P`	*lasio.las_items.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:

```
In [208]: las.sections['Version']
Out[208]:
[HeaderItem(mnemonic=VERS, unit=, value=2.0, descr=CWLS LOG ASCII STANDARD - VERSION␣
↪2.0, original_mnemonic=VERS),
 HeaderItem(mnemonic=WRAP, unit=, value=NO, descr=ONE LINE PER DEPTH STEP, original_
↪mnemonic=WRAP)]

In [182]: las.version
Out[182]:
[HeaderItem(mnemonic=VERS, unit=, value=2.0, descr=CWLS LOG ASCII STANDARD - VERSION␣
↪2.0, original_mnemonic=VERS),
 HeaderItem(mnemonic=WRAP, unit=, value=NO, descr=ONE LINE PER DEPTH STEP, original_
↪mnemonic=WRAP)]
```

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

```
In [209]: las.version.VERS
Out[209]: HeaderItem(mnemonic=VERS, unit=, value=2.0, descr=CWLS LOG ASCII STANDARD -␣
↪VERSION 2.0, original_mnemonic=VERS)

In [210]: las.version['VERS']
Out[210]: HeaderItem(mnemonic=VERS, unit=, value=2.0, descr=CWLS LOG ASCII STANDARD -␣
↪VERSION 2.0, original_mnemonic=VERS)

In [211]: las.version[0]
Out[211]: HeaderItem(mnemonic=VERS, unit=, value=2.0, descr=CWLS LOG ASCII STANDARD -␣
↪VERSION 2.0, original_mnemonic=VERS)

In [212]: id(Out[209]), id(Out[210]), id(Out[211])
Out[212]: (250964032, 250964032, 250964032)
```

As you can see, either attribute-style or item-style access is fine - with one exception, see below.

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

```
In [188]: las.well
Out[188]:
```

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

The CTRY item is blank. We will set it:

```
In [190]: las.well.CTRY = 'Australia'

In [191]: las.well.CTRY
Out[191]: HeaderItem(mnemonic=CTRY, unit=, value=Australia, descr=, original_
↪mnemonic=CTRY)
```

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

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

```
In [192]: las.curves
Out[192]:
[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,))]

In [193]: las.curves.PR.unit = 'ohmm'

In [194]: las.curves.PR
Out[194]: 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, original_mnemonic=BS),
 HeaderItem(mnemonic=JOBN, unit=, value=, descr=JOBN, original_mnemonic=JOBN),
 HeaderItem(mnemonic=WPMT, unit=, value=, descr=WPMT, original_mnemonic=WPMT),
 HeaderItem(mnemonic=AGL, unit=, value=, descr=AGL, original_mnemonic=AGL),
 HeaderItem(mnemonic=PURP, unit=, value=Cased hole stratigraphy, descr=PURP, original_
↪mnemonic=PURP),
 HeaderItem(mnemonic=X, unit=, value=560160, descr=X, original_mnemonic=X),
 HeaderItem(mnemonic=CSGL, unit=, value=0 m - 135 m, descr=CSGL, original_
↪mnemonic=CSGL),
 HeaderItem(mnemonic=UNIT, unit=, value=, descr=UNIT, original_mnemonic=UNIT),
 HeaderItem(mnemonic=Y, unit=, value=6686430, descr=Y, original_mnemonic=Y),
 HeaderItem(mnemonic=TDL, unit=, value=135.2 m, descr=TDL, original_mnemonic=TDL),
 HeaderItem(mnemonic=PROD, unit=, value=, descr=PROD, original_mnemonic=PROD),
 HeaderItem(mnemonic=MUD, unit=, value=Water, descr=MUD, original_mnemonic=MUD),
 HeaderItem(mnemonic=CSGS, unit=, value=100 mm, descr=CSGS, original_mnemonic=CSGS),
 HeaderItem(mnemonic=ENG, unit=, value=, descr=ENG, original_mnemonic=ENG),
 HeaderItem(mnemonic=STEP, unit=, value=5 cm, descr=STEP, original_mnemonic=STEP),
 HeaderItem(mnemonic=FluidLevel, unit=, value=54 m, descr=FluidLevel, original_
↪mnemonic=FluidLevel),
 HeaderItem(mnemonic=CSGT, unit=, value=PVC, descr=CSGT, original_mnemonic=CSGT),
 HeaderItem(mnemonic=WIT, unit=, value=, descr=WIT, original_mnemonic=WIT),
 HeaderItem(mnemonic=EREF, unit=, value=, descr=EREF, original_mnemonic=EREF),
 HeaderItem(mnemonic=PROJ, unit=, value=, descr=PROJ, original_mnemonic=PROJ),
 HeaderItem(mnemonic=ZONE, unit=, value=53J, descr=ZONE, original_mnemonic=ZONE),
 HeaderItem(mnemonic=DREF, unit=, value=GL, descr=DREF, original_mnemonic=DREF),
 HeaderItem(mnemonic=TDD, unit=, value=136 m, descr=TDD, original_mnemonic=TDD)]
```

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

```
In [197]: las.params.DREF.mnemonic = 'LMF'

In [198]: las.params
Out[198]:
[HeaderItem(mnemonic=BS, unit=, value=216 mm, descr=BS, original_mnemonic=BS),
 HeaderItem(mnemonic=JOBN, unit=, value=, descr=JOBN, original_mnemonic=JOBN),
 HeaderItem(mnemonic=WPMT, unit=, value=, descr=WPMT, original_mnemonic=WPMT),
 HeaderItem(mnemonic=AGL, unit=, value=, descr=AGL, original_mnemonic=AGL),
 HeaderItem(mnemonic=PURP, unit=, value=Cased hole stratigraphy, descr=PURP, original_
↪mnemonic=PURP),
 HeaderItem(mnemonic=X, unit=, value=560160, descr=X, original_mnemonic=X),
 HeaderItem(mnemonic=CSGL, unit=, value=0 m - 135 m, descr=CSGL, original_
↪mnemonic=CSGL),
 HeaderItem(mnemonic=UNIT, unit=, value=, descr=UNIT, original_mnemonic=UNIT),
 HeaderItem(mnemonic=Y, unit=, value=6686430, descr=Y, original_mnemonic=Y),
 HeaderItem(mnemonic=TDL, unit=, value=135.2 m, descr=TDL, original_mnemonic=TDL),
 HeaderItem(mnemonic=PROD, unit=, value=, descr=PROD, original_mnemonic=PROD),
 HeaderItem(mnemonic=MUD, unit=, value=Water, descr=MUD, original_mnemonic=MUD),
 HeaderItem(mnemonic=CSGS, unit=, value=100 mm, descr=CSGS, original_mnemonic=CSGS),
 HeaderItem(mnemonic=ENG, unit=, value=, descr=ENG, original_mnemonic=ENG),
 HeaderItem(mnemonic=STEP, unit=, value=5 cm, descr=STEP, original_mnemonic=STEP),
 HeaderItem(mnemonic=FluidLevel, unit=, value=54 m, descr=FluidLevel, original_
↪mnemonic=FluidLevel),
 HeaderItem(mnemonic=CSGT, unit=, value=PVC, descr=CSGT, original_mnemonic=CSGT),
 HeaderItem(mnemonic=WIT, unit=, value=, descr=WIT, original_mnemonic=WIT),
 HeaderItem(mnemonic=EREF, unit=, value=, descr=EREF, original_mnemonic=EREF),
 HeaderItem(mnemonic=PROJ, unit=, value=, descr=PROJ, original_mnemonic=PROJ),
 HeaderItem(mnemonic=ZONE, unit=, value=53J, descr=ZONE, original_mnemonic=ZONE),
 HeaderItem(mnemonic=LMF, unit=, value=GL, descr=DREF, original_mnemonic=LMF),
 HeaderItem(mnemonic=TDD, unit=, value=136 m, descr=TDD, original_mnemonic=TDD)]
```

And now we need to add a new mnemonic. Adding via an attribute **will not work**. You need to use the item-style access.

```
In [201]: las.params['DRILL'] = lasio.HeaderItem(mnemonic='DRILL', value='John Smith',
↪ descr='Driller on site')

In [202]: las.params
Out[202]:
[HeaderItem(mnemonic=BS, unit=, value=216 mm, descr=BS, original_mnemonic=BS),
 HeaderItem(mnemonic=JOBN, unit=, value=, descr=JOBN, original_mnemonic=JOBN),
 HeaderItem(mnemonic=WPMT, unit=, value=, descr=WPMT, original_mnemonic=WPMT),
 HeaderItem(mnemonic=AGL, unit=, value=, descr=AGL, original_mnemonic=AGL),
 HeaderItem(mnemonic=PURP, unit=, value=Cased hole stratigraphy, descr=PURP, original_
↪mnemonic=PURP),
 HeaderItem(mnemonic=X, unit=, value=560160, descr=X, original_mnemonic=X),
 HeaderItem(mnemonic=CSGL, unit=, value=0 m - 135 m, descr=CSGL, original_
↪mnemonic=CSGL),
 HeaderItem(mnemonic=UNIT, unit=, value=, descr=UNIT, original_mnemonic=UNIT),
 HeaderItem(mnemonic=Y, unit=, value=6686430, descr=Y, original_mnemonic=Y),
 HeaderItem(mnemonic=TDL, unit=, value=135.2 m, descr=TDL, original_mnemonic=TDL),
 HeaderItem(mnemonic=PROD, unit=, value=, descr=PROD, original_mnemonic=PROD),
 HeaderItem(mnemonic=MUD, unit=, value=Water, descr=MUD, original_mnemonic=MUD),
 HeaderItem(mnemonic=CSGS, unit=, value=100 mm, descr=CSGS, original_mnemonic=CSGS),
 HeaderItem(mnemonic=ENG, unit=, value=, descr=ENG, original_mnemonic=ENG),
 HeaderItem(mnemonic=STEP, unit=, value=5 cm, descr=STEP, original_mnemonic=STEP),
 HeaderItem(mnemonic=FluidLevel, unit=, value=54 m, descr=FluidLevel, original_
↪mnemonic=FluidLevel),
 HeaderItem(mnemonic=CSGT, unit=, value=PVC, descr=CSGT, original_mnemonic=CSGT),
 HeaderItem(mnemonic=WIT, unit=, value=, descr=WIT, original_mnemonic=WIT),
 HeaderItem(mnemonic=EREF, unit=, value=, descr=EREF, original_mnemonic=EREF),
 HeaderItem(mnemonic=PROJ, unit=, value=, descr=PROJ, original_mnemonic=PROJ),
 HeaderItem(mnemonic=ZONE, unit=, value=53J, descr=ZONE, original_mnemonic=ZONE),
 HeaderItem(mnemonic=LMF, unit=, value=GL, descr=DREF, original_mnemonic=LMF),
 HeaderItem(mnemonic=TDD, unit=, value=136 m, descr=TDD, original_mnemonic=TDD),
 HeaderItem(mnemonic=DRILL, unit=, value=John Smith, descr=Driller on site, original_
↪mnemonic=DRILL)]
```

Bingo.

4.2 Handling errors

lasio will do its best to read every line from the header section. If it can make sense of it, it will parse it into a mnemonic, unit, value, and description.

However often there are problems in LAS files. For example, a header section might contain something like:

```
COUNTY: RUSSELL
```

(missing period, should be `COUNTY.  :  RUSSELL`). Or:

```
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
```

Obviously the line with " causes an error.

All these (and any other kind of error in the header section) can be turned from LASHeaderError exceptions into `logger.warning()` calls instead by using `lasio.read(..., ignore_header_errors=True)`.

Here is an example. First we try reading a file without this argument:

```
In [2]: las = lasio.read('tests/examples/dodgy_param_sect.las', ignore_header_
↪errors=False)
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\Code\lasio\lasio\reader.py in parse_header_section(sectdict, version, ignore_header_
↪errors, mnemonic_case)
    458         try:
--> 459             values = read_line(line)
    460         except:

~\Code\lasio\lasio\reader.py in read_line(*args, **kwargs)
    625     '''
--> 626     return read_header_line(*args, **kwargs)
    627

~\Code\lasio\lasio\reader.py in read_header_line(line, pattern)
    656     m = re.match(pattern, line)
--> 657     mdict = m.groupdict()
    658     for key, value in mdict.items():

AttributeError: 'NoneType' object has no attribute 'groupdict'

During handling of the above exception, another exception occurred:

LASHeaderError                            Traceback (most recent call last)
<ipython-input-2-3c0606fe7dc1> in <module>()
----> 1 las = lasio.read('tests/examples/dodgy_param_sect.las', ignore_header_
↪errors=False)

~\Code\lasio\lasio\__init__.py in read(file_ref, **kwargs)
    41
    42     '''
---> 43     return LASFile(file_ref, **kwargs)

~\Code\lasio\lasio\las.py in __init__(self, file_ref, **read_kwargs)
    76
    77         if not (file_ref is None):
---> 78             self.read(file_ref, **read_kwargs)
    79
    80     def read(self, file_ref,

~\Code\lasio\lasio\las.py in read(self, file_ref, ignore_data, read_policy, null_
↪policy, ignore_header_errors, mnemonic_case, **kwargs)
```

```
    185         add_section("~P", "Parameter", version=version,
    186                     ignore_header_errors=ignore_header_errors,
--> 187                     mnemonic_case=mnemonic_case)
    188         s = self.match_raw_section("~O")
    189 

~\Code\lasio\lasio\las.py in add_section(pattern, name, **sect_kws)
    122             if raw_section:
    123                 self.sections[name] = reader.parse_header_section(raw_section,
--> 124                                                                   **sect_kws)
    125                 drop.append(raw_section["title"])
    126             else:

~\Code\lasio\lasio\reader.py in parse_header_section(sectdict, version, ignore_header_
↪errors, mnemonic_case)
    465                 logger.warning(message)
    466             else:
--> 467                 raise exceptions.LASHeaderError(message)
    468         else:
    469             if mnemonic_case == 'upper':

LASHeaderError: line 31 (section ~PARAMETER INFORMATION): "DEPTH     DT        RHOB     ␣
↪ NPHI     SFLU     SFLA      ILM      ILD"
```

Now if we use ignore_header_errors=True:

```
In [3]: las = lasio.read('tests/examples/dodgy_param_sect.las', ignore_header_
↪errors=True)
line 31 (section ~PARAMETER INFORMATION): "DEPTH     DT        RHOB     NPHI     SFLU ␣
↪   SFLA      ILM      ILD"

In [4]: las.params
[]

In [5]: las.curves
Out[5]:
[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,))]
```

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

4.3 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:

```
In [2]: las = lasio.read('tests/examples/mnemonic_duplicate2.las')

In [3]: las.curves
Out[3]:
[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,))]

In [4]: las.curves['RES:2']
Out[4]: 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:

```
In [6]: import sys

In [7]: las.write(sys.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
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
```

4.3.1 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.

```
In [8]: las = lasio.read('tests/examples/mnemonic_case.las')

In [9]: las.curves
Out[9]:
[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,))]
```

```
In [10]: las = lasio.read('tests/examples/mnemonic_case.las', mnemonic_case='preserve
↪')

In [11]: las.curves
Out[11]:
[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,)),
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 errors

`lasio` has a flexible way of handling "errors" in the ~ASCII data section to accommodate how strict or flexible you want to be.

5.1.1 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
```

5.1.2 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
    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:

```
In [9]: las = lasio.read('tests/examples/null_policy_runon.las')

In [12]: las.data
Out[12]:
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]])
```

5.1.3 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 '),
           (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
```

Ordinarily it would raise an exception:

```
In [13]: las = lasio.read('tests/examples/null_policy_ERR.las')
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\Code\lasio\lasio\reader.py in read_file_contents(file_obj, regexp_subs, value_null_
↪subs, ignore_data)
    271                 try:
--> 272                     data = read_data_section_iterative(file_obj, regexp_subs,␣
↪value_null_subs)
    273                 except:

~\Code\lasio\lasio\reader.py in read_data_section_iterative(file_obj, regexp_subs,␣
↪value_null_subs)
    348
--> 349     array = np.fromiter(items(file_obj), np.float64, -1)
    350     for value in value_null_subs:

ValueError: could not convert string to float: 'ERR'
```

```
During handling of the above exception, another exception occurred:

LASDataError                              Traceback (most recent call last)
<ipython-input-13-0cb27623119d> in <module>()
----> 1 las = lasio.read('tests/examples/null_policy_ERR.las')

~\Code\lasio\lasio\__init__.py in read(file_ref, **kwargs)
     41
     42     '''
---> 43     return LASFile(file_ref, **kwargs)

~\Code\lasio\lasio\las.py in __init__(self, file_ref, **read_kwargs)
     76
     77         if not (file_ref is None):
---> 78             self.read(file_ref, **read_kwargs)
     79
     80     def read(self, file_ref,

~\Code\lasio\lasio\las.py in read(self, file_ref, ignore_data, read_policy, null_
↪policy, ignore_header_errors, **kwargs)
    106
    107         self.raw_sections = reader.read_file_contents(
--> 108             file_obj, regexp_subs, value_null_subs, ignore_data=ignore_data, )
    109
    110         if hasattr(file_obj, "close"):

~\Code\lasio\lasio\reader.py in read_file_contents(file_obj, regexp_subs, value_null_
↪subs, ignore_data)
    274                     raise exceptions.LASDataError(
    275                         traceback.format_exc()[:-1] +
--> 276                         ' in data section beginning line {}'.format(i + 1))
    277                 sections[line] = {
    278                     "section_type": "data",

LASDataError: Traceback (most recent call last):
  File "C:\Users\kent\Code\lasio\lasio\reader.py", line 272, in read_file_contents
    data = read_data_section_iterative(file_obj, regexp_subs, value_null_subs)
  File "C:\Users\kent\Code\lasio\lasio\reader.py", line 349, in read_data_section_
↪iterative
    array = np.fromiter(items(file_obj), np.float64, -1)
ValueError: could not convert string to float: 'ERR' in data section beginning line 43
```

But if we specify the regular expression to use with `re.sub()`, we can easily load it:

```
In [14]: las = lasio.read('tests/examples/null_policy_ERR.las', null_policy=[('ERR',
↪' NaN '), ])

In [16]: las.data
Out[16]:
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, nan, 0.45, 123.45, -999.25, 110.2, 105.6]])

In [17]:
```

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
 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:

```
In [31]: las = lasio.read("tests/examples/2.0/sample_2.0.las")

In [33]: 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
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
 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:

```
In [26]: las.version
Out[26]:
```

```
[HeaderItem(mnemonic=VERS, unit=, value=1.2, descr=CWLS log ASCII Standard -VERSION 1.
↪20, original_mnemonic=VERS),
 HeaderItem(mnemonic=WRAP, unit=, value=YES, descr=Multiple lines per depth step,␣
↪original_mnemonic=WRAP)]

In [27]: las.version.WRAP = 'NO'

In [28]: las.version.WRAP
Out[28]: HeaderItem(mnemonic=WRAP, unit=, value=NO, descr=Multiple lines per depth␣
↪step, original_mnemonic=WRAP)

In [29]: 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
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:

```
In [23]: las.write('example-version-2.0.las', version=2.0)

In [24]:
```

```
~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
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
```

```
    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
```

CHAPTER 7

Exporting to other formats

The following examples all use `sample.las`:

```
~VERSION INFORMATION
 VERS.                  1.2:   CWLS LOG ASCII STANDARD -VERSION 1.2
 WRAP.                  NO:   ONE LINE PER DEPTH STEP
~WELL INFORMATION BLOCK
#MNEM.UNIT       DATA TYPE    INFORMATION
#---------    -------------   ------------------------------
 STRT.M        1670.000000:
 STOP.M        1660.000000:
 STEP.M            -0.1250:
 NULL.           -999.2500:
 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
~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
 SFLA.OHMM                   :  6  SHALLOW RESISTIVITY
 ILM .OHMM                   :  7  MEDIUM RESISTIVITY
 ILD .OHMM                   :  8  DEEP RESISTIVITY
~PARAMETER INFORMATION
#MNEM.UNIT        VALUE       DESCRIPTION
#---------    -------------   ------------------------------
 BHT .DEGC          35.5000:   BOTTOM HOLE TEMPERATURE
 BS   .MM           200.0000:   BIT SIZE
 FD   .K/M3        1000.0000:   FLUID DENSITY
 MATR.                0.0000:   NEUTRON MATRIX(0=LIME,1=SAND,2=DOLO)
 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 meters causing the data
       between 625 meters and 615 meters 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
```

7.1 Comma-separated values (CSV)

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.las.LASFile.to_csv()* method.

```
In [3]: import lasio

In [4]: las = lasio.read('tests/examples/sample.las')

In [6]: las.to_csv('sample.csv')
```

```
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:

```
In [7]: las.to_csv('sample.csv', 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:

```
In [11]: las.to_csv('sample.csv', 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.

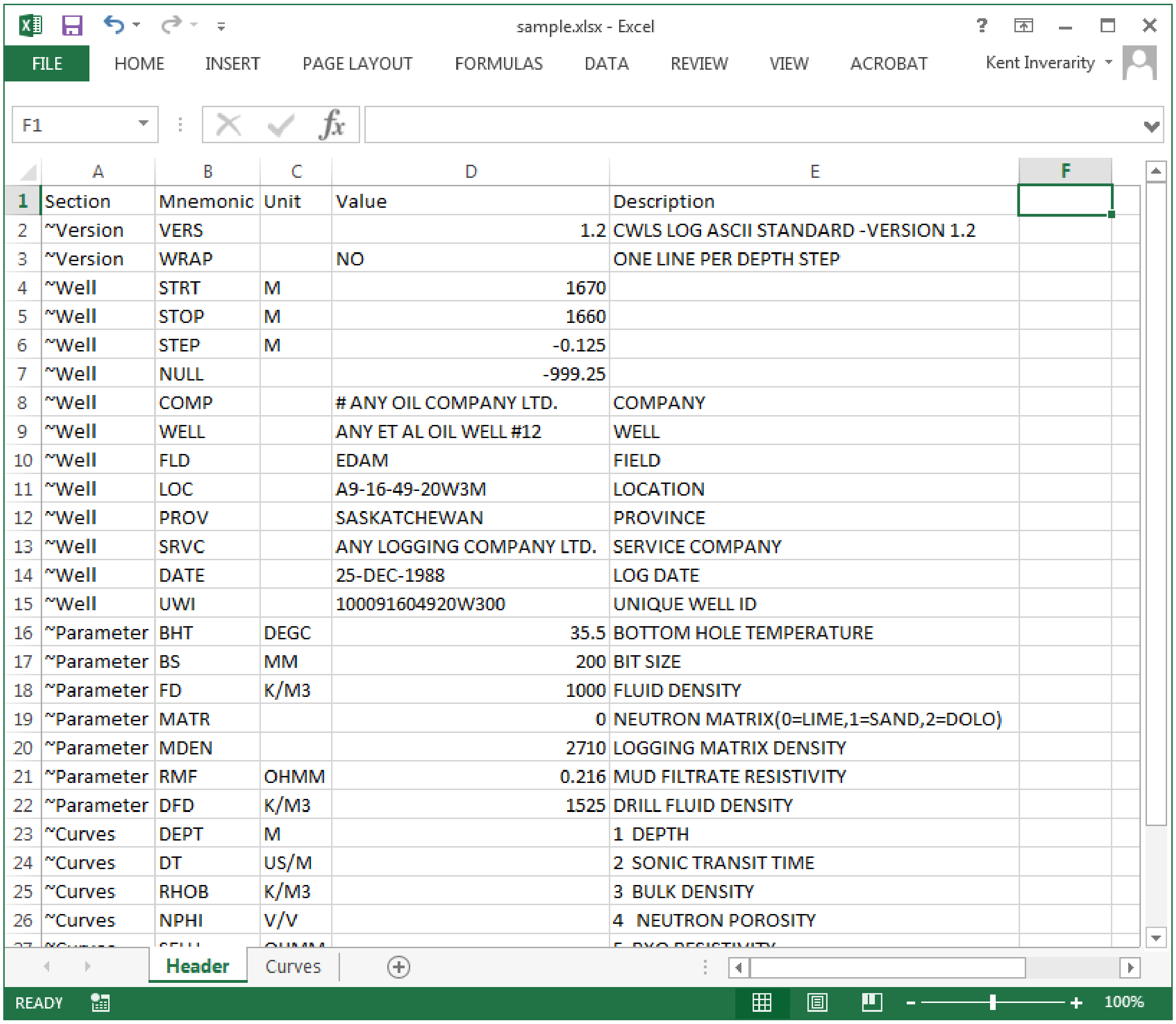
If we are working in Python, you export like this:

```
In [58]: las = lasio.read('tests/examples/sample.las')

In [59]: las.to_excel('sample.xlsx')
```

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

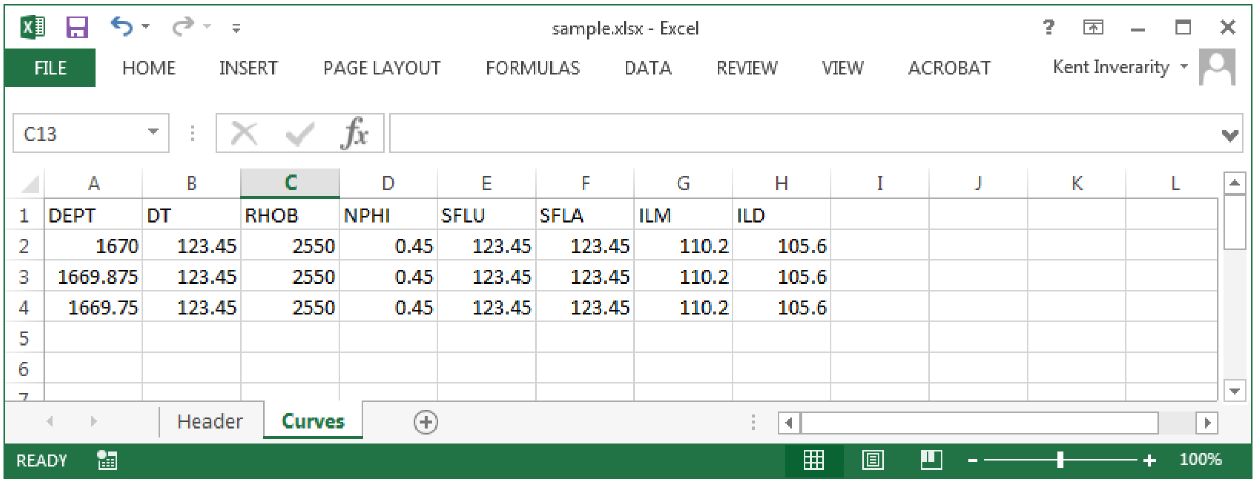
7.2.1 Format of exported Excel file

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	
27	~Curves	SFLU	OHMM		5 RXO RESISTIVITY	

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

7.2.2.1 Single file

```
(py36) C:\Program Files (x86)\Misc\kentcode\lasio>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

(py36) C:\Program Files (x86)\Misc\kentcode\lasio>las2excel tests\examples\sample.las c:\users\kinverarity\Desktop\sample.xlsx
```

7.2.2.2 Multiple files (`las2excelbulk`)

The better script to use is `las2excelbulk`:

```
(py36) C:\Windows\System32>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:

```
(py36) C:\Users\kinverarity\Documents\scratch2017\November>las2excelbulk --recursive␣
↪test_folder
Converting test_folder\-2793 & -2746\5086\PN41497.LAS -> test_folder\-2793 & -
↪2746\5086\pn41497.xlsx
Converting test_folder\-2793 & -2746\5149\PN41497.LAS -> test_folder\-2793 & -
↪2746\5149\pn41497.xlsx
Converting test_folder\-2794\6356\66302794.las -> test_folder\-2794\6356\66302794.xlsx
Converting test_folder\-2794\6808\66302794.las -> test_folder\-2794\6808\66302794.xlsx
Converting test_folder\-2794\7608\2794HYD.LAS -> test_folder\-2794\7608\2794hyd.xlsx
Converting test_folder\-2794\7608\66302794.LAS -> test_folder\-2794\7608\66302794.xlsx
Failed to convert file. Error message:
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)
  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 204, in read
    data = np.reshape(arr, (-1, n_arr_cols))
  File "C:\Program Files (x86)\Miniconda3\envs\py36\lib\site-
↪packages\numpy\core\fromnumeric.py", line 232, in reshape
    return _wrapfunc(a, 'reshape', newshape, order=order)
  File "C:\Program Files (x86)\Miniconda3\envs\py36\lib\site-
↪packages\numpy\core\fromnumeric.py", line 57, in _wrapfunc
    return getattr(obj, method)(*args, **kwds)
ValueError: cannot reshape array of size 25708 into shape (11)

Converting test_folder\-2794\7627\clr105.las -> test_folder\-2794\7627\clr105.xlsx
Converting test_folder\-2839 &c\4830\PN36385.LAS -> test_folder\-2839 &c\4830\pn36385.
↪xlsx
Converting test_folder\-2874\6375\66302874.las -> test_folder\-2874\6375\66302874.xlsx
Converting test_folder\-2874\7607\2874HYD.LAS -> test_folder\-2874\7607\2874hyd.xlsx
Converting test_folder\-2874\7607\66302874.LAS -> test_folder\-2874\7607\66302874.xlsx
Failed to convert file. Error message:
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)
  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 204, in read
    data = np.reshape(arr, (-1, n_arr_cols))
  File "C:\Program Files (x86)\Miniconda3\envs\py36\lib\site-
↪packages\numpy\core\fromnumeric.py", line 232, in reshape
    return _wrapfunc(a, 'reshape', newshape, order=order)
  File "C:\Program Files (x86)\Miniconda3\envs\py36\lib\site-
↪packages\numpy\core\fromnumeric.py", line 57, in _wrapfunc
    return getattr(obj, method)(*args, **kwds)
ValueError: cannot reshape array of size 31666 into shape (16)

Converting test_folder\-2874\7626\clr121.las -> test_folder\-2874\7626\clr121.xlsx
Converting test_folder\-2875\5220\1cm\PN44456.LAS -> test_folder\-
↪2875\5220\1cm\pn44456.xlsx
Converting test_folder\-2875\5220\5cm\PN44456.LAS -> test_folder\-
↪2875\5220\5cm\pn44456.xlsx
Converting test_folder\-2875\5220\980402\PN44456.LAS -> test_folder\-
↪2875\5220\980402\pn44456.xlsx
Converting test_folder\-2875\5220\980403_0\PN44456.LAS -> test_folder\-
↪2875\5220\980403_0\pn44456.xlsx
```

```
Converting test_folder\-2875\5220\980403_1\PN44456.LAS -> test_folder\-
↪2875\5220\980403_1\pn44456.xlsx
Converting test_folder\-2875\5220\cal1cm\PN44456.LAS -> test_folder\-
↪2875\5220\cal1cm\pn44456.xlsx
Converting test_folder\-2875\5220\cal5cm\PN44456.LAS -> test_folder\-
↪2875\5220\cal5cm\pn44456.xlsx
Converting test_folder\-2875\5220\tm2\PN44456.LAS -> test_folder\-
↪2875\5220\tm2\pn44456.xlsx
Converting test_folder\-2875\6813\2875HYD.LAS -> test_folder\-2875\6813\2875hyd.xlsx
Header section Parameter regexp=~P was not found.
Converting test_folder\-2875\6813\66302875.LAS -> test_folder\-2875\6813\66302875.xlsx
Converting test_folder\-2876\5219\PN44457.LAS -> test_folder\-2876\5219\pn44457.xlsx
Converting test_folder\-2876\5219\PN44457H.LAS -> test_folder\-2876\5219\pn44457h.xlsx
Converting test_folder\-2876\5219\PN44457I.LAS -> test_folder\-2876\5219\pn44457i.xlsx
Converting test_folder\-2876\7609\2876H.LAS -> test_folder\-2876\7609\2876h.xlsx
Converting test_folder\-2876\7609\66302876.LAS -> test_folder\-2876\7609\66302876.xlsx
Failed to convert file. Error message:
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)
  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 204, in read
    data = np.reshape(arr, (-1, n_arr_cols))
  File "C:\Program Files (x86)\Miniconda3\envs\py36\lib\site-
↪packages\numpy\core\fromnumeric.py", line 232, in reshape
    return _wrapfunc(a, 'reshape', newshape, order=order)
  File "C:\Program Files (x86)\Miniconda3\envs\py36\lib\site-
↪packages\numpy\core\fromnumeric.py", line 57, in _wrapfunc
    return getattr(obj, method)(*args, **kwds)
ValueError: cannot reshape array of size 19791 into shape (11)

Converting test_folder\-2876\7629\clr120.las -> test_folder\-2876\7629\clr120.xlsx
Converting test_folder\-2877\7597\CLR118.LAS -> test_folder\-2877\7597\clr118.xlsx
Converting test_folder\-2877\7628\clr118.las -> test_folder\-2877\7628\clr118.xlsx
Converting test_folder\-3066\6372\66303066.las -> test_folder\-3066\6372\66303066.xlsx
Converting test_folder\-3066\6810\3066HYD.LAS -> test_folder\-3066\6810\3066hyd.xlsx
Converting test_folder\-3066\6810\66303066.LAS -> test_folder\-3066\6810\66303066.xlsx
Converting test_folder\-3067\6373\66303067.las -> test_folder\-3067\6373\66303067.xlsx
Converting test_folder\-3067\6811\3067HYD.LAS -> test_folder\-3067\6811\3067hyd.xlsx
Converting test_folder\-3067\6811\66303067.LAS -> test_folder\-3067\6811\66303067.xlsx
Header section Parameter regexp=~P was not found.
Converting test_folder\-3068\6374\66303068.las -> test_folder\-3068\6374\66303068.xlsx
Converting test_folder\-3068\6812\3068HYD.LAS -> test_folder\-3068\6812\3068hyd.xlsx
Converting test_folder\-3068\6812\66303068.LAS -> test_folder\-3068\6812\66303068.xlsx
```

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:

```
(py36) Q:\>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):

```
(py36) Q:\>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 LASFile from scratch, it comes with some default metadata:

```
In [5]: import lasio

In [6]: las = lasio.LASFile()

In [7]: las.header
Out[7]:
{'Curves': [],
 'Other': '',
 'Parameter': [],
 'Version': [HeaderItem(mnemonic=VERS, unit=, value=2.0, descr=CWLS log ASCII␣
↪Standard -VERSION 2.0, original_mnemonic=VERS),
  HeaderItem(mnemonic=WRAP, unit=, value=NO, descr=One line per depth step, original_
↪mnemonic=WRAP),
  HeaderItem(mnemonic=DLM, unit=, value=SPACE, descr=Column Data Section Delimiter,␣
↪original_mnemonic=DLM)],
 'Well': [HeaderItem(mnemonic=STRT, unit=m, value=nan, descr=START DEPTH, original_
↪mnemonic=STRT),
  HeaderItem(mnemonic=STOP, unit=m, value=nan, descr=STOP DEPTH, original_
↪mnemonic=STOP),
  HeaderItem(mnemonic=STEP, unit=m, value=nan, descr=STEP, original_mnemonic=STEP),
  HeaderItem(mnemonic=NULL, unit=, value=-9999.25, descr=NULL VALUE, original_
↪mnemonic=NULL),
  HeaderItem(mnemonic=COMP, unit=, value=, descr=COMPANY, original_mnemonic=COMP),
  HeaderItem(mnemonic=WELL, unit=, value=, descr=WELL, original_mnemonic=WELL),
  HeaderItem(mnemonic=FLD, unit=, value=, descr=FIELD, original_mnemonic=FLD),
  HeaderItem(mnemonic=LOC, unit=, value=, descr=LOCATION, original_mnemonic=LOC),
  HeaderItem(mnemonic=PROV, unit=, value=, descr=PROVINCE, original_mnemonic=PROV),
  HeaderItem(mnemonic=CNTY, unit=, value=, descr=COUNTY, original_mnemonic=CNTY),
  HeaderItem(mnemonic=STAT, unit=, value=, descr=STATE, original_mnemonic=STAT),
  HeaderItem(mnemonic=CTRY, unit=, value=, descr=COUNTRY, original_mnemonic=CTRY),
  HeaderItem(mnemonic=SRVC, unit=, value=, descr=SERVICE COMPANY, original_
↪mnemonic=SRVC),
  HeaderItem(mnemonic=DATE, unit=, value=, descr=DATE, original_mnemonic=DATE),
  HeaderItem(mnemonic=UWI, unit=, value=, descr=UNIQUE WELL ID, original_
↪mnemonic=UWI),
  HeaderItem(mnemonic=API, unit=, value=, descr=API NUMBER, original_mnemonic=API)]}
```

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

```
In [8]: from datetime import datetime

In [9]: las.well.DATE = str(datetime.today())
```

And add some new header fields:

```
In [10]: las.params['ENG'] = lasio.HeaderItem('ENG', value='Kent Inverarity')

In [11]: las.params['LMF'] = lasio.HeaderItem('LMF', value='GL')

In [12]: las.other = 'Example of how to create a LAS file from scratch using lasio'
```

We will invent some data for a curve:

```
In [1]: import numpy as np

In [2]: depths = np.arange(10, 50, 0.5)

In [3]: synth = np.log10(depths)*5+np.random.random(len(depths))

In [4]: synth[:8] = np.nan
```

. . . add these to the LASFile object:

```
In [13]: las.add_curve('DEPT', depths, unit='m')

In [14]: las.add_curve('SYNTH', synth, descr='fake data')
```

And write the result to files:

```
In [16]: las.write('scratch_v1.2.las', version=1.2)

In [15]: 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           10.0 : START DEPTH
STOP.m           49.5 : STOP DEPTH
STEP.m            0.5 : 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 : 2017-11-04 15:33:20.963287
UWI .  UNIQUE WELL ID :
API .      API NUMBER :
~Curves ----------------------------------------------------
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   -9999.25
       10.5   -9999.25
         11   -9999.25
       11.5   -9999.25
         12   -9999.25
       12.5   -9999.25
         13   -9999.25
       13.5   -9999.25
         14      5.799
       14.5     6.3938
         15     6.4122
       15.5     6.4605
         16     6.9518
       16.5      6.567
         17     6.3816
       17.5     6.2872
         18     6.4336
       18.5     7.0252
         19     6.7988
       19.5     6.7172
         20     6.6929
       20.5     7.0971
         21      7.145
       21.5     6.7192
         22     7.6034
       22.5     7.3078
         23     7.2213
       23.5      7.668
         24      7.853
       24.5     7.4073
         25     7.4238
       25.5     7.9173
         26     7.1282
       26.5     7.4131
         27     7.8014
       27.5      7.348
         28        7.9
       28.5     7.6294
         29     8.1244
       29.5     7.9835
         30     7.4759
       30.5     8.3766
         31     7.4717
       31.5     7.6432
```

```
         32     8.2327
       32.5     7.6541
         33     8.4481
       33.5     7.8811
         34     8.2332
       34.5     8.4302
         35     7.7218
       35.5       8.71
         36     8.3965
       36.5     8.4355
         37     8.6836
       37.5     8.2236
         38     8.4997
       38.5     8.6656
         39     8.8295
       39.5     8.1707
         40     8.9034
       40.5      8.681
         41     8.1698
       41.5     8.3001
         42     9.0266
       42.5     8.4398
         43     8.7562
       43.5     8.2673
         44     8.4682
       44.5     8.5801
         45     8.9065
       45.5     8.8392
         46      8.661
       46.5     9.2355
         47     9.0468
       47.5     8.8249
         48     9.0298
       48.5     8.6864
         49     8.5745
       49.5     8.6143
```

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                      10.0 : START DEPTH
STOP.m                      49.5 : STOP DEPTH
STEP.m                       0.5 : 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. 2017-11-04 15:33:20.963287 : DATE
```

```
UWI .                            : UNIQUE WELL ID
API .                            : API NUMBER
~Curves ----------------------------------------------------
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   -9999.25
       10.5   -9999.25
         11   -9999.25
       11.5   -9999.25
         12   -9999.25
       12.5   -9999.25
         13   -9999.25
       13.5   -9999.25
         14      5.799
       14.5     6.3938
         15     6.4122
       15.5     6.4605
         16     6.9518
       16.5      6.567
         17     6.3816
       17.5     6.2872
         18     6.4336
       18.5     7.0252
         19     6.7988
       19.5     6.7172
         20     6.6929
       20.5     7.0971
         21      7.145
       21.5     6.7192
         22     7.6034
       22.5     7.3078
         23     7.2213
       23.5      7.668
         24      7.853
       24.5     7.4073
         25     7.4238
       25.5     7.9173
         26     7.1282
       26.5     7.4131
         27     7.8014
       27.5      7.348
         28        7.9
       28.5     7.6294
         29     8.1244
       29.5     7.9835
         30     7.4759
       30.5     8.3766
         31     7.4717
       31.5     7.6432
         32     8.2327
       32.5     7.6541
         33     8.4481
       33.5     7.8811
         34     8.2332
       34.5     8.4302
         35     7.7218
       35.5       8.71
         36     8.3965
       36.5     8.4355
         37     8.6836
       37.5     8.2236
         38     8.4997
       38.5     8.6656
         39     8.8295
       39.5     8.1707
         40     8.9034
       40.5      8.681
         41     8.1698
       41.5     8.3001
         42     9.0266
       42.5     8.4398
         43     8.7562
       43.5     8.2673
         44     8.4682
       44.5     8.5801
         45     8.9065
       45.5     8.8392
         46      8.661
       46.5     9.2355
         47     9.0468
       47.5     8.8249
         48     9.0298
       48.5     8.6864
         49     8.5745
       49.5     8.6143
```

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 a package like cChardet (faster) or chardet (slower) to automatically detect the character encoding. If these packages are installed then lasio will use them by default:

```
>>> import logging
>>> logging.basicConfig()
>>> logging.getLogger().setLevel(logging.DEBUG)
>>> las = lasio.read('encodings_utf8.las')
DEBUG:lasio.reader:get_encoding Using cchardet
DEBUG:lasio.reader:cchardet method detected encoding of UTF-8 at confidence 0.
↪9900000095367432
INFO:lasio.reader:Opening encodings_utf8.las as UTF-8 and treating errors with
↪"replace"
DEBUG:lasio.las:n_curves=8 ncols=8
DEBUG:lasio.las:set_data data.shape = (3, 8)
DEBUG:lasio.las:set_data self.data.shape = (3, 8)
```

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 Module contents

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

Read a LAS file.

Note that only versions 1.2 and 2.0 of the LAS file specification are currently supported.

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

Returns A LASFile object representing the file – see above

There are a number of optional keyword arguments that can be passed to this function that control how the LAS file is opened and parsed. Any of the keyword arguments from the below functions can be used here:

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

10.2 Submodules

10.3 lasio.las module

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

Bases: `object`

LAS file object.

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

See these routines for additional keyword arguments you can use when reading in a LAS file:

- *`lasio.reader.open_with_codecs()`* - manage issues relate to character encodings

- *lasio.las.LASFile.read()* - control how NULL values and errors are handled during parsing

encoding

str or None – the character encoding used when reading the file in from disk

read (*file_ref*, *ignore_data=False*, *read_policy='default'*, *null_policy='common'*, *ignore_header_errors=False*, *mnemonic_case='upper'*, ***kwargs*)

Read a LAS file.

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

Keyword Arguments

- **null_subs** (*bool*) – if True, replace invalid values with np.nan
- **ignore_data** (*bool*) – if True, do not read in any of the actual data, just the header metadata. False by default.
- **ignore_header_errors** (*bool*) – ignore LASHeaderErrors (False by default)
- **mnemonic_case** (*str*) – 'preserve': keep the case of HeaderItem mnemonics 'upper': convert all HeaderItem mnemonics to uppercase 'lower': convert all HeaderItem mnemonics to lowercase

See *lasio.reader.open_with_codecs()* for additional keyword arguments which help to manage issues relate to character encodings.

write (*file_ref*, ***kwargs*)

Write LAS file to disk.

Parameters **file_ref** (*open file-like object or str*) – a file-like object opening 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!

10.3.0.0.1 Examples

```
>>> with open('test_output.las', mode='w') as f:
...     lasfile_obj.write(f, version=2.0)   # <-- this method
```

to_excel (*filename*)

Export LAS file to a Microsoft Excel workbook.

This function will raise an `ImportError` if `openpyxl` is not installed.

Parameters **filename** (*str*) –

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*) – a file-like object opening 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'`.

match_raw_section(*pattern*, *re_func='match'*, *flags=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.

get_curve(*mnemonic*)

Return CurveItem object.

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

Returns `lasio.las_items.CurveItem` (not just the data array)

keys()

Return curve mnemonics.

values()

Return data for each curve.

items()

Return mnemonics and data for all curves.

iterkeys()

itervalues()

iteritems()

version

Header information from the Version (~V) section.

Returns `lasio.las_items.SectionItems` object.

well

Header information from the Well (~W) section.

Returns `lasio.las_items.SectionItems` object.

curves

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

Returns `lasio.las_items.SectionItems` object.

curvesdict

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

Returns dict

params

Header information from the Parameter (~P) section.

Returns `lasio.las_items.SectionItems` object.

other

Header information from the Other (~O) section.

Returns str

metadata

All header information joined together.

Returns `lasio.las_items.SectionItems` object.

header

All header information

Returns dict

df()

Return data as a `pandas.DataFrame` structure.

data

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.las_items.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.

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.

Keyword arguments are passed to `lasio.las.LASFile.set_data()`.

index

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

depth_m

Return the index as metres.

depth_ft

Return the index as feet.

add_curve_raw(*mnemonic*, *data*, *unit=”*, *descr=”*, *value=”*)

Deprecated. Use append_curve_item() or insert_curve_item() instead.

append_curve_item(*curve_item*)

Add a CurveItem.

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

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*) –

add_curve (**args*, ***kwargs*)
Deprecated. Use append_curve() or insert_curve() instead.

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.

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.

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.

json
Return object contents as a JSON string.

class lasio.las.**Las** (*file_ref=None*, ***read_kwargs*)
Bases: *lasio.las.LASFile*

LAS file object.

Retained for backwards compatibility.

class lasio.las.**JSONEncoder** (*skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)
Bases: json.encoder.JSONEncoder

default (*obj*)

10.4 lasio.las_items module

class lasio.las_items.**HeaderItem**(*mnemonic=”*, *unit=”*, *value=”*, *descr=”*)
Bases: collections.OrderedDict

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.

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

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

__getitem__(*key*)
Provide item dictionary-like access.

class lasio.las_items.**CurveItem**(*mnemonic=”*, *unit=”*, *value=”*, *descr=”*, *data=None*)
Bases: *lasio.las_items.HeaderItem*

Dictionary/namedtuple-style object for a LAS curve.

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

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

API_code
Equivalent to the `value` attribute.

class lasio.las_items.**SectionItems**(**args*, ***kwargs*)
Bases: list

Variant of a `list` which is used to represent a LAS section.

__contains__(*testitem*)
Check whether a header item or mnemonic is in the section.

Parameters **testitem** (HeaderItem, CurveItem, *str*) – either an item or a mnemonic

Returns bool

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.

__getslice__(*i0*, *i1*)
For Python 2.7 compatibility.

__getitem__(*key*)

Item-style access by either mnemonic or index.

Parameters **key** (`str, int, slice`) – either a mnemonic or the index to the list.

Returns item from the list (either HeaderItem or CurveItem)

__delitem__(*key*)

Delete item by either mnemonic or index.

Parameters **key** (`str, int`) – either a mnemonic or the index to the list.

__setitem__(*key*, *newitem*)

Either replace the item or its value.

Parameters

- **key** (`int, str`) – either the mnemonic or the index.
- **newitem** (`HeaderItem or str/float/int`) – the thing to be set.

If `newitem` is a `lasio.las_items.HeaderItem` then the existing item will be replaced. Otherwise the existing item's `value` attribute will be replaced.

i.e. this allows us to do

```
>>> section.OPERATOR
HeaderItem(mnemonic='OPERATOR', value='John')
>>> section.OPERATOR = 'Kent'
>>> section.OPERATOR
HeaderItem(mnemonic='OPERATOR', value='Kent')
```

See `lasio.las_items.SectionItems.set_item()` and `lasio.las_items.SectionItems.set_item_value()`.

__getattr__(*key*)

Provide attribute access via __contains__ e.g.

```
>>> section['VERS']
HeaderItem(mnemonic='VERS', ...)
>>> 'VERS' in section
True
>>> section.VERS
HeaderItem(mnemonic='VERS', ...)
```

__setattr__(*key*, *value*)

Allow access to `lasio.las_items.SectionItems.__setitem__()` via attribute access.

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.

__weakref__
list of weak references to the object (if defined)

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.

10.5 lasio.reader module

lasio.reader.**open_file** (*file_ref*, ***encoding_kwargs*)
Open a file if necessary.

If `autodetect_encoding=True` then either `cchardet` or `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 `codecs.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/cchardet 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.**adhoc_test_encoding**(*filename*)

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

Automatically detect character encoding.

Parameters

- **auto** (*str*) – auto-detection of character encoding - can be either 'chardet', 'cchardet', 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.

lasio.reader.**read_file_contents**(*file_obj*, *regexp_subs*, *value_null_subs*, *ignore_data=False*)

Read file contents into memory.

Parameters **file_obj** (*open file-like object*) –

Keyword Arguments

- **null_subs** (*bool*) – True will substitute numpy.nan for invalid values
- **ignore_data** (*bool*) – if True, do not read in the numerical data in the ~ASCII section

Returns OrderedDict

I think of the returned dictionary as a "raw section". The keys are the first line of the LAS section, including the tilde. Each value is a dict with either:

```
{"section_type": "header",
 "title": str,                  # title of section (including the ~)
 "lines": [str, ],              # a list of the lines from the lAS file
 "line_nos": [int, ]            # line nos from the original file
 }
```

or:

```
{"section_type": "data",
 "title": str,                 # title of section (including the ~)
 "start_line": int,            # location of data section (the title line)
 "ncols": int,                 # no. of columns on first line of data,
 "array": ndarray              # 1-D numpy.ndarray,
 }
```

lasio.reader.**read_data_section_iterative**(*file_obj*, *regexp_subs*, *value_null_subs*)

Read data section into memory.

Parameters

- **file_obj** (*open file-like object*) – should be positioned in line-by-line reading mode, with the last line read being the title of the ~ASCII data section.
- **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.

Returns A 1-D 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.

`lasio.reader.`**`parse_header_section`**(*sectdict*, *version*, *ignore_header_errors=False*, *mnemonic_case='preserve'*)

Parse a header section dict into a SectionItems containing HeaderItems.

Parameters

- **`sectdict`** (*`dict`*) – object returned from *`lasio.reader.read_file_contents()`*
- **`version`** (*`float`*) – either 1.2 or 2.0

Keyword Arguments

- **`ignore_header_errors`** (*`bool`*) – if True, issue HeaderItem parse errors as `logging.warning()` calls instead of a *`lasio.exceptions.LASHeaderError`* exception.
- **`mnemonic_case`** (*`str`*) – 'preserve': keep the case of HeaderItem mnemonics 'upper': convert all HeaderItem mnemonics to uppercase 'lower': convert all HeaderItem mnemonics to lowercase

Returns *`lasio.las_items.SectionItems`*

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

Bases: `object`

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.

`num`(*x*, *default=None*)

Attempt to parse a number.

Parameters

- **`x`** (*`str`*, *`int`*, *`float`*) – potential number
- **`default`** (*`int`*, *`float`*, *`None`*) – fall-back option

Returns int, float, or **default** - from most to least preferred types.

`metadata`(***keys*)

Return HeaderItem correctly formatted according to the order prescribed for LAS v 1.2 or 2.0 for the ~W section.

Keyword arguments should be the key:value pairs returned by `lasio.reader.read_header_line()`.

`curves` (***keys*)
Return CurveItem.

Keyword arguments should be the key:value pairs returned by `lasio.reader.read_header_line()`.

`params` (***keys*)
Return HeaderItem for ~P section (the same between 1.2 and 2.0 specs)

Keyword arguments should be the key:value pairs returned by `lasio.reader.read_header_line()`.

`lasio.reader.`**`read_line`**(**args*, ***kwargs*)
Retained for backwards-compatibility.

See `lasio.reader.read_header_line()`.

`lasio.reader.`**`read_header_line`**(*line*, *pattern=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

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

10.6 lasio.writer module

`lasio.writer.`**`write`**(*las*, *file_object*, *version=None*, *wrap=None*, *STRT=None*, *STOP=None*, *STEP=None*, *fmt='%10.5g'*)
Write a LAS files.

Parameters

- **`las`** (`lasio.las.LASFile`) –
- **`file_object`** (`file-like object open for writing`) – output

Keyword Arguments

- **`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.

You should avoid calling this function directly - instead use the *lasio.las.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.las_items.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: OrderedDict([('Version', ['value:descr']), ('Well', ['descr:value', ('value:descr', ['STRT', 'STOP', 'STEP', 'NULL'])]), ('Curves', ['value:descr']), ('Parameter', ['value:descr'])]), 2.0: OrderedDict([('Version', ['value:descr']), ('Well', ['value:descr']), ('Curves', ['value:descr']), ('Parameter', ['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()*

10.7 lasio.excel module

10.8 lasio.defaults module

`lasio.defaults.`**`get_default_items`**`()`

10.9 lasio.exceptions module

`exception` `lasio.exceptions.`**`LASDataError`**

Bases: `Exception`

Error during reading of numerical data from LAS file.

`exception` `lasio.exceptions.`**`LASHeaderError`**

Bases: `Exception`

Error during reading of header data from LAS file.

`exception` `lasio.exceptions.`**`LASUnknownUnitError`**

Bases: `Exception`

Error of unknown unit in LAS file.

- genindex
- search

Python Module Index

Index