TotalDepth Documentation

Release 0.4.0

Paul Ross

Sep 14, 2021

CONTENTS

TotalDepth is an Open Source, cross platform, software collection that processes petrophysical data from the oil field such as wireline logs.

If you are new here then have a look at some *TotalDepth Example Outputs*. For more detail see *Introduction to TotalDepth*.

The TotalDepth project is currently at **Beta**, development version 0.4.0rc0, release version 0.4.0. For the licence see here *Licence*.

CHAPTER

ONE

CONTENTS

1.1 Introduction to TotalDepth

TotalDepth is an Open Source, cross platform, software collection that can process petrophysical data from the oil field such as wireline logs, seismic data and so on.

Conventional, proprietary, software for petrophysical data tends to be expensive to licence, restrictive, slow to develop for and tied to expensive hardware. TotalDepth changes all of that.

TotalDepth is open and cross-platform, and produces results straight to the bowser. TotalDepth supports such technologies such as HTML5, AJAX, Software as a Service (SaaS) and Cloud Computing.

TotalDepth is currently at **Alpha**, development version 0.4.0rc0, release version 0.4.0. For the licence see *Licence*.

TotalDepth is written in Python with performance critical code written in C or C++.

1.1.1 Petrophysical File Formats Supported by TotalDepth

Format	Support	Since	Notes.
LIS	Full	0.1.0	
LAS version 1.2	Full	0.1.0	
LAS version 2.0	Full	0.1.0	
LAS version 3.0	None		Little evidence that this is used by the industry.
WellLogML	None		No evidence that this is used by the industry.
RP66v1 “DLIS”	Full	0.3.0	Full.
RP66v2 “DLIS”	None		No evidence that this is used by the industry.
DAT	Full	0.4.0	An informal standard commonly used for mud logs.
Western Atlas BIT	Most	0.4.0	An informal standard not publicly documented.

1.2 TotalDepth Example Outputs

This shows some examples of the kind of thing that TotalDepth can do.

1.2.1 Wireline Plots

TotalDepth produced these time honoured plots from LIS and LAS wireline logs in SVG format that can be viewed in most browsers[1].

Plotting from LIS

Some examples of plots generated from LIS79 files:

- A collection of 9 LIS files where TotalDepth used their internal plot specifications to generate 22 separate plots.
- A High Resolution Dipmeter plot on a scale of 1:25 with an API header. Fast channel FC0 (red) overlaid on FC1.

Plotting From LAS

This shows off some examples of plots generated from the Canadian Well Logging Society's LAS formated files[2].

Single LAS Plot examples

This shows plots of a single LAS file that has 200 feet of 15 curves. TotalDepth can plot this with, linear and log scales and with an API header:

- Plotted with the Resistivity 3Track Logrithmic format.
- The same data plotted with the Triple Combo format.

The original LAS file is here.

A Collection of LAS Plots

Here is a directory of six LAS files that was used to make 31 individual plots complete with an index that summarises them. For each LAS file the plotting program automatically choose from 29 plot formats the formats that produce useful plots[3].

Making LAS Plots

The `PlotLogs.py` command line tool was used with the command:

```
$ python3 PlotLogs.py -A -j4 -r -X 4 Data/ Plot/
```

This searched for LAS files in directory `Data/` with the plots being written in directory `Plot/`.

The following options have been set:

- API headers on the top of each plot: `-A`
- Multiprocessing on with 4 simultaneous jobs: `-j4`
- Recursive search of input directory: `-r`

[1] There is good SVG support among current browsers such as Opera, Chrome and Safari. You can find a comparison of browser support for SVG at Wikipedia.

[2] Thanks to the University of Kansas [kgs.ku.edu] for the original data. For these examples that data has been edited or truncated or both.

[3] A *useful* plot format is one that can handle at least *n* curves where *n* is a number that is specified by the user. If the user specifies 4 then there will be at least 4 curves on each plot.

- Uses any available plot specifications from LgFormat XML files which result in 4 curves or more being plotted: `-X 4`

This took around six seconds to compute. More detail on the `PlotLogs.py` is here: *Plotting Well Logs with tdplotlogs*

1.2.2 LIS Log HTML Summaries

The program `LisToHtml.py` takes LIS file(s) and generates an LIS HTML summary for each one.

How This HTML was Made

The `LisToHtml.py` command line tool was used with the command:

```
$ python3 LisToHtml.py -k -j4 -r Data/ HTML/
```

This searched for LAS files in directory `Data/` with the files being written to directory `HTML/`.

The following options have been set:

- Keep going as far as possible: `-k`
- Multiprocessing on with 4 simultaneous jobs: `-j4`
- Recursive search of input directory: `-r`

More detail on the `LisToHtml.py` is here: *Summarise LIS Files in HTML with tdlistohtml*

1.2.3 RP66V1 Log HTML Summaries

The program `ScanHtml.py` takes RP66V1 file(s) and generates an RP66V1 HTML summary for each one.

1.3 Installation

1.3.1 Stable release

To install TotalDepth, run this command in your terminal:

```
$ pip install TotalDepth
```

This is the preferred method to install TotalDepth, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

1.3.2 From sources

If you are using a virtual environment in your *<PYTHONVENVS>*, say *~/pyvenvs*:

```
$ python3 -m venv <PYTHONVENVS>/TotalDepth
$ . <PYTHONVENVS>/TotalDepth/bin/activate
(TotalDepth) $
```

Or if you have a Conda environment:

```
$ conda create --name TotalDepth python=3.6 pip
$ source activate TotalDepth
```

Install the dependencies, `numpy` and `Cython`:

If you are using a virtual environment:

```
(TotalDepth) $ pip install numpy
(TotalDepth) $ pip install Cython
```

Or if you have a Conda environment:

```
(TotalDepth) $ conda install numpy
(TotalDepth) $ conda install Cython
```

The sources for TotalDepth can be downloaded from the Github repo.

You can either clone the public repository:

```
(TotalDepth) $ git clone git://github.com/paulross/TotalDepth.git
```

Or download the tarball:

```
(TotalDepth) $ curl  -OL https://github.com/paulross/TotalDepth/tarball/master
```

Once you have a copy of the source, you can install it with:

```
(TotalDepth) $ cd TotalDepth
(TotalDepth) $ python setup.py install
```

Install the test dependencies and run TotalDepth's tests:

```
(TotalDepth) $ pip install pytest
(TotalDepth) $ pip install pytest-runner
(TotalDepth) $ python setup.py test
```

1.3.3 Developing with TotalDepth

If you are developing with TotalDepth you need the test coverage and documentation tools.

Test Coverage

Install `pytest-cov`:

```
(TotalDepth) $ pip install pytest-cov
```

The most meaningful invocation that elimates the top level tools is:

```
(TotalDepth) $ pytest --cov=TotalDepth.LAS.core --cov=TotalDepth.LIS.core --
↪cov=TotalDepth.RP66.core --cov=TotalDepth.util --cov-report html tests/
```

Documentation

To build the documentation you need to:

```
(TotalDepth) $ pip install Sphinx
(TotalDepth) $ cd docs
(TotalDepth) $ make html
```

System Testing

See *Testing the Plot Package* for comprehensive testing of your installation to see if LIS/LAS files can be written, read and plotted. This pretty much executes all TotalDepth code.

Unit Testing

See *Unit Tests* for more information about testing and unit tests.

1.4 HOWTOs: What TotalDepth can do for you

This describes how TotalDepth can help you with various tasks when dealing with petrophysical data.

1.4.1 Working with Archives of Mixed Data

I have an archive of files and I want to know what is in there.

Sometimes you are given an archive of data and would like to know what file formats, the file sizes and so on. There is a command line tool `tdarchive` [References *`TotalDepth.util.archive`*] that can give you a summary of the files there, their binary file types and their sizes.

Read more here *Analysing a Directory of Files with tdarchive*.

I have an archive and I want copy specific file types

There is a command line tool `tdcopybinfiles` [References *TotalDepth.util.CopyBinFiles*] that can copy specific file types from one directory to another. It can also recursively deflate archives such as ZIP files.

Read more here *Copying a Directory of Files with tdcopybinfiles*.

I have an archive and I want report or remove duplicate files

There is a command line tool `tdremovedupefiles` [References *TotalDepth.util.RemoveDupeFiles*] that can find duplicate files based on their checksum and, optionally, remove the duplicates.

Read more here *Removing Duplicate Files with tdremovedupefiles*.

1.4.2 Working with Western Atlas BIT Data

I have an archive of data and I'd like a summary

There is a command line tool *tdbitread* (a wrapper around *TotalDepth.BIT.ReadBIT*) which can generate a summary of a body of BIT files. There is a tutorial here: *Summarise BIT Files with tdbitread*

I'd like to convert BIT files to LAS format files

There is a tutorial here : *Converting BIT Files to LAS Files with tdbittolas*.

Getting the Frame Data as a `numpy` Array

TotalDepth's BIT parser represents the channel data as Numpy arrays. There is a tutorial here on writing code that allows you to access the Numpy channel data directly: *Reading a BIT File Log Data*

I have some troublesome BIT files

This is a highly specialised area. Feel free to contact the author for advice.

1.4.3 Working with LAS Data

I have an archive of data and I'd like a summary in HTML

There is a command line tool *tdlastohtml* (a wrapper around `TotalDepth.LAS.LASToHtml`) which can generate an HTML summary of a body of LAS files. There is a tutorial here: *Summarise LAS Files in HTML with tdlastohtml*

I have an archive of LAS data and I want to plot it

There is a generalised well log plotting command line tool `tdplotlogs` that supports LAS described here: *Plotting Well Logs with tdplotlogs* (references `TotalDepth.PlotLogs`).

LAS files do not contain an internal plot specification. For files needing an external plot specification there is some background information on plotting in a technical note here *Wireline Files With External Plot Data*.

Getting the Frame Data as a `numpy` Array

TotalDepth's LAS represents the channel data as Numpy arrays. There is a tutorial here on writing code that allows you to access the Numpy channel data directly: *Reading a LAS File Log Data*

I have some troublesome LAS files

There are several problem areas for LAS files:

- The LAS specification is fairly weak and provides a lot of uncertainty. So LAS files from some producers are not readable by some other LAS consumers.
- LAS is a 'human readable' format, unfortunately that means it is a human writable format as well. This often means that LAS files can be mangled by well meaning, but mistaken intervention.
- Some LAS file archives have serious errors such as swapping value and description fields. These are not easily fixable by a rule based system.

The advantage, of course, with LAS files is that the can be hacked around with a simple text editor at will. This will often fix small local problems.

Feel free to contact the author for advice.

1.4.4 Working with LIS Data

I have an archive of data and I'd like a summary in HTML

There is a command line tool *tdlistohtml* (a wrapper around `TotalDepth.LIS.LisToHtml`) which can generate an HTML summary of a body of LIS files. There is a tutorial here: *Summarise LIS Files in HTML with tdlistohtml*

I have an archive of LIS data and I want to plot it

There is a generalised well log plotting command line tool *tdplotlogs* that supports LIS and is described here: *Plotting Well Logs with tdplotlogs* (references `TotalDepth.PlotLogs`).

LIS files may or may not contain an internal plot specification, if so `tdplotlogs` can take advantage of that, if not then external plot specifications can be used. For files needing an external plot specification there is some background information on plotting in a technical note here *Wireline Files With External Plot Data*.

I'd like to convert LIS files to LAS format files

There is a tutorial here : *Converting LIS Files to LAS Files with tdlistolas*.

Getting the Frame Data as a `numpy` Array

TotalDepth's LIS represents the channel data as Numpy arrays. There is a tutorial here on writing code that allows you to access the Numpy channel data directly: *Reading a LIS File Log Data*

I have some troublesome LIS files

This is a highly specialised area. Feel free to contact the author for advice.

1.4.5 Working with RP66V1 Data

TotalDepth version 0.3.0 (you are viewing 0.4.0rc0) onwards provides a RP66V1 standard compliant implementation. RP66V1 file handling is work-in-progress.

I have an archive of RP66V1 data and I'd like a summary in HTML

TotalDepth's `tdrp66v1scanhtml` command line tool can do this, it is a wrapper around *`TotalDepth.RP66V1.ScanHTML`* There is a tutorial here: *Creating HTML Pages from RP66V1 Files with tdrp66v1scanhtml*.

Here is an example of the HTML summary of a single RP66V1 file .

I'd like to create some well plots from RP66V1 data

Unlike LIS and like LAS, RP66V1 files do not specify a plot format. Some producers include some producer specific information in private EFLRs. TotalDepth version 0.4.0 will provide a simpler, universal, way of specifying plot formats in SVG from LAS, LIS and RP66V1 data.

I'd like to convert RP66V1 files to LAS format files

There is a tutorial here : *Converting RP66V1 Files to LAS Files with tdrp66v1tolas*.

Getting the Frame Data as a `numpy` Array

There is a tutorial here on writing code that allows you to access the numpy channel data directly: *Reading the Frame Data and Accessing the numpy Arrays*. There is some example code in `example_data/RP66V1/demo_read.py`

I have some troublesome RP66V1 files

On problem noticed in RP66V1 data from the wild is that it is often polluted with gratuitous TIF markers which makes the file unreadable by a RP66V1 standard compliant implementation. TotalDepth's `tddetif` command line tool can remove these TIF markers and make the files readable. There is a tutorial here: *Removing TIF Markers From Files with tddetif*

Other problems require special skills, feel free to contact the author for advice.

1.4.6 Converting Files to LAS

Converting LIS Files to LAS

There is a command line tool `tdlistolas` that can convert LIS files to LAS 2.0 files. There is a tutorial here: *Converting LIS Files to LAS Files with tdlistolas*. This command line tool is a wrapper round the `ToLAS` module, the reference documentation is here: *`TotalDepth.LIS.ToLAS`*.

Converting RP66V1 Files to LAS

There is a command line tool `tdrp66v1tolas` that can convert RP66V1 files to LAS 2.0 files. There is a tutorial here: *Converting RP66V1 Files to LAS Files with tdrp66v1tolas*. This command line tool is a wrapper round the `ToLAS` module, the reference documentation is here: *`TotalDepth.RP66V1.ToLAS`*.

There is a technical note about the performance of this conversion here *Converting RP66V1 to LAS*.

1.5 Command Line Tools

TotalDepth provides a number of tools run from the command line that can analyse and visualise petrophysical data. All the command line tools start with `td`.

1.5.1 TotalDepth Command Line Tools

This describes the command line tools that are available for processing any TotalDepth file.

The tools are located in `TotalDepth/`

Table 1: **LIS Command Line Tools**

Tool Name	Description
`tdarchive`	Scans a directory of files and provides an analysis by file type. *Link*
`tdcopybinfiles`	Selects files of a particular type and copies them to a directory. This can also expand compressed archives. *Link*
`tdremovedupefiles`	Removes duplicate files based on their checksum. *Link*
`tddetif`	Removes TIF markers from a file. *Link*
`tdplotlogs`	Plots logs from LIS and LAS data. *Link*

Analysing a Directory of Files with `tdarchive`

Scans a directory of files and provides an analysis by file type.

Usage

Usage:

```
usage: tdarchive [-h] [--file-type FILE_TYPE] [-b BYTES] [-r]
                 [--expand-and-delete] [--histogram] [-n] [-o] [-l LOG_LEVEL]
                 [-v]
                 path [path_out]
```

Arguments

1. The path to the input directory
2. [Optional] Path to the output directory to write the files to. The results are undefined if path_out conflicts with path_in.

Options

Option	Description
`-h`, `--help`	Show this help message and exit.
`--file-type`	Types of files to analyse, this option is is additive so can be used multiple times. Supported (and default) file types are: ASCII, LAS1.2, LAS2.0, LAS3.0, LIS, LISt, LIStr, PDF, PS, RP66V1, RP66V1t, RP66V1tr, RP66V2, SEGY, TIFF, XML, ZIP
`-b`, `--bytes=`	The number of initial bytes of the file to show [default: 0].
`-r`, `--recursive`	Process input recursively. [default: False]
`--expand-and-delete`	Expand and delete archive files, implies `--recurse`. [default: False]
`--histogram`	Plot a histogram of file sizes. [default: False]
`-n`, `--nervous`	Nervous mode, does not do anything but report.
`-o`, `--over-write`	Overwrite existing files, otherwise warns.
`-l`, `--log-level=`	Log Level as an integer or symbol. (0<->NOTSET, 10<->DEBUG, 20<->INFO, 30<->WARNING, 40<->ERROR, 50<->CRITICAL) [default: 30]
`-v`, `--verbose`	Increase verbosity, additive [default: 0]

Examples

Here is an example of scanning the `example_data` directory (the output is filleted for clarity):

```
$ tdarchive example_data/ -r --histogram
CMD: tdarchive example_data/ -r --histogram
Analysing archive.
Common prefix:
       108,707 .SVG       XML      example_data/LIS/Plot/DILLSON-1_WELL_LOGS_FILE-013.
↪LIS_0001_HDT.svg
           667 .HTML      XML      example_data/LIS/Plot/index.html
       163,169 .SVG       XML      example_data/LIS/Plot/DILLSON-1_WELL_LOGS_FILE-037.
↪LIS_0001_1.svg
           690 .CSS       ASCII    example_data/LIS/Plot/index.css
...
       184,084 .LIS       LIS      example_data/LIS/data/DILLSON-1_WELL_LOGS_FILE-037.
↪LIS
        96,376 .LIS       LIS      example_data/LIS/data/DILLSON-1_WELL_LOGS_FILE-013.
↪LIS
        98,508 .LIS       LIS      example_data/LIS/data/DILLSON-1_WELL_LOGS_FILE-049.
↪LIS
        10,306 .PY        ASCII    example_data/RP66V1/demo_read.py
       462,795 .HTML      XML      example_data/RP66V1/HTML/206_05a-_3_DWL_DWL_WIRE_
↪258276498.DLIS.HTML.html
         3,098 .HTML      XML      example_data/RP66V1/HTML/index.html
...
     1,018,327 .PKL               example_data/RP66V1/pickle/206_05a-_3_DWL_DWL_WIRE_
↪258276498.pkl
       276,045 .PKL               example_data/RP66V1/pickle/206_05a-_3_DWL_DWL_WIRE_
↪258276498.DLIS.pkl
           520 .PKL               example_data/RP66V1/pickle/MINIMAL_FILE.dlis.pkl
        53,137 .PKL               example_data/RP66V1/pickle/BASIC_FILE.dlis.pkl
           949 .PKL               example_data/RP66V1/pickle/BASIC_FILE_WITH_TWO_
↪VISIBLE_RECORDS_NO_IFLRS.dlis.pkl
         7,731 .LAS       LAS2.0   example_data/RP66V1/LAS/BASIC_FILE_WITH_TWO_
↪VISIBLE_RECORDS_NO_IFLRS_0_.las
     1,621,374 .LAS       LAS2.0   example_data/RP66V1/LAS/206_05a-_3_DWL_DWL_WIRE_
↪258276498_0_800T.las
...
        44,916 .DLIS      RP66V1   example_data/RP66V1/data/BASIC_FILE.dlis
       540,372 .DLIS      RP66V1   example_data/RP66V1/data/206_05a-_3_DWL_DWL_WIRE_
↪258276498.DLIS
           716 .DLIS      RP66V1   example_data/RP66V1/data/MINIMAL_FILE.dlis
         8,826 .DLIS      RP66V1   example_data/RP66V1/data/BASIC_FILE_WITH_TWO_
↪VISIBLE_RECORDS_NO_IFLRS.dlis
Total number of files 75, total bytes 12,981,766
File extensions:
.CSS  :        5
.DLIS :        4
.HTML :       13
.LAS  :        6
.LIS  :        3
.PKL  :        5
.PY   :        1
.PYC  :        1
.SVG  :       36
.XML  :        1
```

(continues on next page)

(continued from previous page)

```
Binary file types:
Binary type: ""
 Extensions: .PKL, .PYC
      Count: 6 [8.000%]
      Bytes: 1,354,629 [10.435%] from 520 to  1,018,327
>=2**9  [     2] |␣
↪++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
>=2**10 [     0] |
>=2**11 [     0] |
>=2**12 [     1] | ++++++++++++++++++++++++++++++++++++++++
>=2**13 [     0] |
>=2**14 [     0] |
>=2**15 [     1] | ++++++++++++++++++++++++++++++++++++++++
>=2**16 [     0] |
>=2**17 [     0] |
>=2**18 [     1] | ++++++++++++++++++++++++++++++++++++++++
>=2**19 [     1] | ++++++++++++++++++++++++++++++++++++++++

Binary type: "ASCII"
 Extensions: .CSS, .PY
      Count: 6 [8.000%]
      Bytes: 15,232 [0.117%] from 690 to  10,306
>=2**9  [     4] |␣
↪++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
>=2**10 [     0] |
>=2**11 [     1] | ++++++++++++++++++++
>=2**12 [     0] |
>=2**13 [     1] | ++++++++++++++++++++

Binary type: "LAS2.0"
 Extensions: .LAS
      Count: 6 [8.000%]
      Bytes: 1,785,653 [13.755%] from 1,279 to  1,621,374
>=2**10 [     1] |␣
↪++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
>=2**11 [     1] |␣
↪++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
>=2**12 [     1] |␣
↪++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
>=2**13 [     0] |
>=2**14 [     0] |
>=2**15 [     1] |␣
↪++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
>=2**16 [     1] |␣
↪++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
>=2**17 [     0] |
>=2**18 [     0] |
>=2**19 [     0] |
>=2**20 [     1] |␣
↪++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Binary type: "LIS"
 Extensions: .LIS
      Count: 3 [4.000%]
      Bytes: 378,968 [2.919%] from 96,376 to  184,084
>=2**16 [     2] |␣
↪++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

(continues on next page)

(continued from previous page)

```
>=2**17 [     1] | ++++++++++++++++++++++++++++++++++++++

Binary type: "RP66V1"
 Extensions: .DLIS
      Count: 4 [5.333%]
      Bytes: 594,830 [4.582%] from 716 to  540,372
>=2**9  [     1] |␣
↪++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
>=2**10 [     0] |
>=2**11 [     0] |
>=2**12 [     0] |
>=2**13 [     1] |␣
↪++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
>=2**14 [     0] |
>=2**15 [     1] |␣
↪++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
>=2**16 [     0] |
>=2**17 [     0] |
>=2**18 [     0] |
>=2**19 [     1] |␣
↪++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Binary type: "XML"
 Extensions: .HTML, .SVG, .XML
      Count: 50 [66.667%]
      Bytes: 8,852,454 [68.191%] from 667 to  961,863
>=2**9  [     1] | ++++
>=2**10 [     1] | ++++
>=2**11 [     2] | +++++++++
>=2**12 [     2] | +++++++++
>=2**13 [     0] |
>=2**14 [     0] |
>=2**15 [     3] | ++++++++++++++
>=2**16 [    16] |␣
↪++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
>=2**17 [    18] |␣
↪++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
>=2**18 [     5] | ++++++++++++++++++++++
>=2**19 [     2] | +++++++++

Execution time: 0.067 (s)
         Files: 75 rate 1,113.2 (files/s)
         Bytes: 1,350 rate 20,038.5 (bytes/s)
```

Copying a Directory of Files with `tdcopybinfiles`

Scans a directory of files and can copy particular file type to another directory.

Usage

Usage:

```
usage: TotalDepth.RP66V1.util.CopyBinFiles.main
       [-h] [--file-types FILE_TYPES] [-m] [-n] [-l LOG_LEVEL]
       path_in path_out
```

Arguments

1. The path to the input directory
2. Path to the output directory to write the files to. The results are undefined if path_out conflicts with path_in.

Options

Option	Description
`-h`, `--help`	Show this help message and exit.
`--file-type`	Types of files to copy, this option is is additive so can be used multiple times. Supported (and default) file types are: ASCII, LAS1.2, LAS2.0, LAS3.0, LIS, LISt, LIStr, PDF, PS, RP66V1, RP66V1t, RP66V1tr, RP66V2, SEGY, TIFF, XML, ZIP Use '?' or '??' to see what file types are available
`-m`, `--move`	Move rather than copy, Irrelevant for files in ZIP archives which are always copied. [default: False]
`-n`, `--nervous`	Nervous mode, does not do anything but report.
`-l`, `--log-level`	Log Level as an integer or symbol. (0<->NOTSET, 10<->DEBUG, 20<->INFO, 30<->WARNING, 40<->ERROR, 50<->CRITICAL) [default: 30]

Examples

To see what file types are supported use '?' and two dummy paths:

```
$ tdcopybinfiles --file-types=? '' ''
Cmd: tdcopybinfiles --file-types=?
Binary file types supported: ASCII, LAS1.2, LAS2.0, LAS3.0, LIS, LISt, LIStr, PDF, PS,
↪ RP66V1, RP66V1t, RP66V1tr, RP66V2, SEGY, TIFF, XML, ZIP
Execution time =    0.000 (S) 0 kb/s
```

To get the file type description as well use '??' and two dummy paths:

```
$ tdcopybinfiles --file-types=?? '' ''
Cmd: tdcopybinfiles --file-types=??
Binary file types supported:
```

(continues on next page)

(continued from previous page)

```
ASCII    - American Standard Code for Information Interchange
LAS1.2   - Canadian Well Logging Society Log ASCII Standard version 1.2
LAS2.0   - Canadian Well Logging Society Log ASCII Standard version 2.0
LAS3.0   - Canadian Well Logging Society Log ASCII Standard version 3.0
LIS      - Schlumberger LIS-79 well logging format
LISt     - Schlumberger LIS-79 well logging format with TIF markers
LIStr    - Schlumberger LIS-79 well logging format with reversed TIF markers
PDF      - Portable Document Format
PS       - Postscript
RP66V1   - American Petroleum Institute Recommended Practice 66 version 1
RP66V1t  - American Petroleum Institute Recommended Practice 66 version 1 with TIF␣
↪markers
RP66V1tr - American Petroleum Institute Recommended Practice 66 version 1 with␣
↪reversed TIF markers
RP66V2   - American Petroleum Institute Recommended Practice 66 version 2
SEGY     - Society of Exploration Geophysicists seismic format Y
TIFF     - Tagged Image File Format
XML      - eXtensible Markup Language
ZIP      - ZIP Compressed Archive
Execution time =    0.000 (S) 0 kb/s
```

Removing Duplicate Files with `tdremovedupefiles`

Scans a directory of files and identifies duplicate files by their checksum. It is **strongly recommended** to use `-n` (`--nervous`) first and look at the results before running this without `-n` which is potentially destructive.

Usage

Usage:

```
usage: TotalDepth.RP66V1.util.RemoveDupeFiles.main [-h] [--version] [-k] [-v]
                                                   [-r] [-l LOG_LEVEL] [-n]
                                                   path_in
```

Arguments

1. The path to the directory.

Options

Option	Description
-h, --help	Show this help message and exit.
-r, --recursive	Process input recursively. [default: False]
-n, --nervous	Nervous mode, does not do anything but report [default: False].
-l, --log-level=	Log Level as an integer or symbol. (0<->NOTSET, 10<->DEBUG, 20<->INFO, 30<->WARNING, 40<->ERROR, 50<->CRITICAL) [default: 20]
-v, --verbose	Increase verbosity, additive [default: 0]

Removing TIF Markers From Files with `tddetif`

Scans a directory of files and removes TIF markers. TIF markers are 12 bytes of data inserted in various places. They are not part of any standard and, except for one case, provide no value. This tool rewrites the input file without TIF markers if they are found.

Usage

Usage:

```
usage: tddetif [-h] [-r] [-n]
       [-l LOG_LEVEL] [-v] [-o]
       path_in [path_out]
```

Arguments

1. The path to the input directory
2. Path to the output directory to write the files to. The results are undefined if path_out conflicts with path_in.

Options

Option	Description
-h, --help	Show this help message and exit.
-r, --recurse	Process input recursively. [default: False]
-n, --nervous	Nervous mode, does not do anything but report [default: False].
-l, --log-level=	Log Level as an integer or symbol. (0<->NOTSET, 10<->DEBUG, 20<->INFO, 30<->WARNING, 40<->ERROR, 50<->CRITICAL) [default: 30]
-v, --verbose	Increase verbosity, additive [default: 0]
-o, --over-write	Overwrite existing files if found, otherwise warns of existing target file.

Examples

todo:

```
Put examples here.
```

These command line tools plot wireline data.

Plotting Well Logs with `tdplotlogs`

Produces SVG plots from LIS and LAS files.

Usage

Usage:

```
usage: tdplotlogs [-h] [--version] [-j JOBS] [-k] [-l LOGLEVEL] [-g] [-r]
                  [-A] [-x LGFORMAT] [-X LGFORMAT_MIN] [-s SCALE]
                  in out
```

Arguments

These are required arguments unless `-h` or `--version` options are specified (in which case no processing is done):

1. The path to the input LAS or LIS file or directory thereof.
2. The path to the output SVG file or directory, any directories will be created as necessary.

Options

Option	Description
`--version`	Show program's version number and exit
`-h, --help`	Show this help message and exit.
`-j JOBS, --jobs=JOBS`	Max processes when multiprocessing. Zero uses number of native CPUs [8]. -1 disables multiprocessing. [default: -1]
`-k, --keep-going`	Keep going as far as sensible. [default: False]
`-l LOGLEVEL, --loglevel=LOGLEVEL`	Log Level (debug=10, info=20, warning=30, error=40, critical=50) [default: 20]
`-g, --glob`	File pattern match. [default none]
`-r, --recursive`	Process input recursively. [default: False]
`-A, --API`	Put an API header on each plot. [default: False]
`-x LGFORMAT, --xml LGFORMAT`	Use XML LgFormat UniqueId to use for plotting (additive). Use -x? to see what LgFormats (UniqueID+Description) are available. Use -x?? to see what curves each format can plot. See also -X. This is additive so can used multiple times to get multiple plots from the same data.
`-X LGFORMAT_MIN, --XML LGFORMAT_MIN`	Use all available LgFormat XML plots that use LGFORMAT_MIN or more outputs. If -x option present limited by those LgFormats [default: 0]
`-s SCALE, --scale SCALE`	Scale of X axis to use (an integer). This overrides the scale(s) specified in the LgFormat file or FILM table. [default: 0].

Examples

LgFormat XML

Using `-x?` to see what formats are available:

```
$ python3 tdplotlogs -x? spam eggs
```

The output is something like:

```
Cmd: tdplotlogs -x? spam eggs
XML LgFormats available: [29]
UniqueId                                Description
-----------------------------------   --------------------------------
ADN_Image_Format                     : ADN Image Log
Azimuthal_Density_3Track.xml         : Azimuthal Density 3Track
Azimuthal_Resistivity_3Track.xml     : Azimuthal Resistivity 3Track
Blank_3Track_Depth                   : Blank 3Track
Blank_3Track_Time.xml                : Blank 3Track Time
FMI_IMAGE_ALIGNED                    : FMI Image Aligned
FMI_IMAGE_PROCESSED                  : FMI Image Processed
Formation_Test                       : Formation Test Time
HDT                                  : High Definition Dipmeter
Micro_Resistivity_3Track.xml         : Micro Resistivity 3 Track Format
```

(continues on next page)

(continued from previous page)

```
Natural_GR_Spectrometry_3Track.xml  : Natural GR Spectrometry 3Track
OBMI_IMAGE_EQUAL                    : OBMI Image Equalized
Porosity_GR_3Track                  : Standard Porosity Curves
Pulsed_Neutron_3Track.xml           : Pulsed Neutron 3Track
Pulsed_Neutron_Time.xml             : Pulsed Neutron Time
RAB_Image_Format_Deep               : Resistivity At the Bit Image
RAB_Image_Format_Medium             : Resistivity At the Bit Image
RAB_Image_Format_Shallow            : Resistivity At the Bit Image
RAB_Std_Format                      : Resistivity At the Bit
Resistivity_3Track_Correlation.xml  : Resistivity Linear Correlation Format
Resistivity_3Track_Logrithmic.xml   : Logrithmic Resistivity 3Track
Resistivity_Investigation_Image.xml : AIT Radial Investigation Image
Sonic_3Track.xml                    : Sonic DT Porosity 3 Track
Sonic_PWF4                          : SONIC Packed Waveform 4
Sonic_SPR1_VDL                      : SONIC Receiver Array Lower Dipole VDL
Sonic_SPR2_VDL                      : SONIC Receiver Array Upper Dipole VDL
Sonic_SPR3_VDL                      : SONIC Receiver Array Stonely VDL
Sonic_SPR4_VDL                      : SONIC Receiver Array P and S VDL
Triple_Combo                        : Resistivity Density Neutron GR 3Track Format
```

The first column is the UniqueID to be used in identifying plots for the `-x` option.

Using `-x??` to see what formats and what curves would be plotted by each plot specification:

```
$ python3 tdplotlogs -x?? a b
```

The output is something like:

```
Cmd: tdplotlogs -x?? a b
XML LgFormats available: [29]
UniqueId                              Description
-----------------------------------   --------------------------------
ADN_Image_Format                    : ADN Image Log
    DRHB, GR  , GR_RAB, ROBB, ROP5, TNPH
Azimuthal_Density_3Track.xml        : Azimuthal Density 3Track
    BS  , DCAL, DRHB, DRHL, DRHO, DRHR, DRHU, DTAB, HDIA, PEB , PEF , PEL
    PER , PEU , RHOB, ROBB, ROBL, ROBR, ROBU, ROP5, RPM , SCN2, SOAB, SOAL
    SOAR, SOAU, SONB, SOXB, VDIA
Azimuthal_Resistivity_3Track.xml    : Azimuthal Resistivity 3Track
    AAI , BS  , C1  , C2  , CALI, GR  , GRDN_RAB, GRLT_RAB, GRRT_RAB, GRUP_RAB, PCAL,␣
↪RDBD
    RDBL, RDBR, RDBU, RLA0, RLA1, RLA2, RLA3, RLA4, RLA5, RMBD, RMBL, RMBR
    RMBU, ROP5, RPM , RSBD, RSBL, RSBR, RSBU, SP  , TENS
Blank_3Track_Depth                  : Blank 3Track
Blank_3Track_Time.xml               : Blank 3Track Time
FMI_IMAGE_ALIGNED                   : FMI Image Aligned
    C1  , C2  , GR  , HAZIM, P1AZ, SP  , TENS
FMI_IMAGE_PROCESSED                 : FMI Image Processed
    C1  , C2  , GR  , HAZIM, P1AZ, SP  , TENS
Formation_Test                      : Formation Test Time
    B1TR, BFR1, BQP1, BQP1, BQP1, BQP1, BSG1, POHP
HDT                                 : High Definition Dipmeter
    C1  , C2  , DEVI, FC0 , FC1 , FC2 , FC3 , FC4 , GR  , HAZI, P1AZ, RB
Micro_Resistivity_3Track.xml        : Micro Resistivity 3 Track Format
    BMIN, BMNO, BS  , CALI, GR  , HCAL, HMIN, HMNO, MINV, MLL , MNOR, MSFL
    PROX, RXO , SP  , TENS
Natural_GR_Spectrometry_3Track.xml  : Natural GR Spectrometry 3Track
```

(continues on next page)

(continued from previous page)

```
    CGR , PCAL, POTA, ROP5, SGR , SIGM, TENS, THOR, URAN
OBMI_IMAGE_EQUAL                    : OBMI Image Equalized
    C1  , C1_OBMT, C2  , C2_OBMT, GR  , HAZIM, OBRA3, OBRB3, OBRC3, OBRD3, P1AZ, P1NO_
↪OBMT
    TENS
Porosity_GR_3Track                  : Standard Porosity Curves
    APDC, APLC, APSC, BS  , C1  , C2  , CALI, CALI_CDN, CMFF, CMRP, DPHB, DPHI
    DPHZ, DPOR_CDN, DRHO, ENPH, GR  , HCAL, NPHI, NPOR, PCAL, RHOB, RHOZ, ROP5
    SNP , SP  , SPHI, TENS, TNPB, TNPH, TNPH_CDN, TPHI
Pulsed_Neutron_3Track.xml           : Pulsed Neutron 3Track
    FBAC, GR  , INFD, SIGM, TAU , TCAF, TENS, TPHI, TSCF, TSCN
Pulsed_Neutron_Time.xml             : Pulsed Neutron Time
    FBAC_SL, GR_SL, INFD_SL, SIGM_SL, TAU_SL, TCAF_SL, TENS_SL, TPHI_SL, TSCF_SL,␣
↪TSCN_SL
RAB_Image_Format_Deep               : Resistivity At the Bit Image
    GR_RAB, RES_BD, RES_BM, RES_BS, RES_RING, ROP5
RAB_Image_Format_Medium             : Resistivity At the Bit Image
    GR_RAB, RES_BD, RES_BM, RES_BS, RES_RING, ROP5
RAB_Image_Format_Shallow            : Resistivity At the Bit Image
    GR_RAB, RES_BD, RES_BM, RES_BS, RES_RING, ROP5
RAB_Std_Format                      : Resistivity At the Bit
    AAI , BDAV, BDM3, BMAV, BMM2, BSAV, BSM1, BTAB, CALI, DEVI, GR_RAB, HAZI
    OBIT, RBIT, RING, ROP5, RPM , RTAB
Resistivity_3Track_Correlation.xml  : Resistivity Linear Correlation Format
    AHT20, AHT60, AHT90, ATR , BS  , CALI, CATR, CILD, CLLD, GR  , HCAL, ILD
    ILM , LLD , LLS , MSFL, PCAL, PSR , RLA0, ROP5, RT  , RXO , SFL , SP
    TENS
Resistivity_3Track_Logrithmic.xml   : Logrithmic Resistivity 3Track
    A22H, A34H, AHF10, AHF20, AHF30, AHF60, AHF90, AHO10, AHO20, AHO30, AHO60, AHO90
    AHT10, AHT20, AHT30, AHT60, AHT90, ATR , BS  , CALI, GR  , HCAL, ILD , ILM
    LLD , LLM , MSFL, P16H_RT, P28H_RT, P34H_RT, PCAL, PSR , RLA0, RLA1, RLA2, RLA3
    RLA4, RLA5, ROP5, RXO , SFL , SP  , TENS
Resistivity_Investigation_Image.xml : AIT Radial Investigation Image
    AHT10, AHT20, AHT30, AHT60, AHT90, BS  , GR  , HCAL, SP
Sonic_3Track.xml                    : Sonic DT Porosity 3 Track
    BS  , CALI, DT  , DT0S, DT1R, DT2 , DT2R, DT4S, DTBC, DTCO, DTCU, DTL
    DTLF, DTLN, DTR2, DTR5, DTRA, DTRS, DTSH, DTSM, DTST, DTTA, GR  , HCAL
    PCAL, ROP5, SP  , SPHI, TENS
Sonic_PWF4                          : SONIC Packed Waveform 4
    CALI, DT1 , DT2 , DTCO, DTSM, DTST, GR  , HCAL, TENS
Sonic_SPR1_VDL                      : SONIC Receiver Array Lower Dipole VDL
    CALI, DT1 , DT1 , DT2 , DTCO, DTSM, DTST, GR  , HCAL, TENS
Sonic_SPR2_VDL                      : SONIC Receiver Array Upper Dipole VDL
    CALI, DT1 , DT2 , DT2 , DTCO, DTSM, DTST, GR  , HCAL, TENS
Sonic_SPR3_VDL                      : SONIC Receiver Array Stonely VDL
    CALI, DT1 , DT2 , DT3R, DTCO, DTSM, DTST, GR  , HCAL, TENS
Sonic_SPR4_VDL                      : SONIC Receiver Array P and S VDL
    CALI, DT1 , DT2 , DTCO, DTRP, DTRS, DTSM, DTST, GR  , HCAL, TENS
Triple_Combo                        : Resistivity Density Neutron GR 3Track Format
    AHT10, AHT20, AHT30, AHT60, AHT90, APDC, APLC, APSC, ATR , BS  , C1  , C2
    CALI, CMFF, CMRP, DPHB, DPHI, DPHZ, DPOR_CDN, DSOZ, ENPH, GR  , HCAL, HMIN
    HMNO, ILD , ILM , LLD , LLM , MSFL, NPHI, NPOR, PCAL, PEFZ, PSR , RLA0
    RLA1, RLA2, RLA3, RLA4, RLA5, ROP5, RSOZ, RXO , RXOZ, SFL , SNP , SP
    SPHI, TENS, TNPB, TNPH, TNPH_CDN, TPHI
```

Plotting Logs

Here is an example of plotting LIS and LAS files in directory `in/` with the plots in directory `out/`. The following options have been invoked:

- API headers on the top of each plot: `-A`
- Multiprocessing on with 4 simultaneous jobs: `-j4`
- Recursive search of input directory: `-r`
- Uses any available plot specifications from LgFormat XML files which result in 4 curves or more being plotted: `-X 4`

The command line is:

```
$ python3 tdplotlogs -A -j4 -r -X 4 in/ out/
```

First tdplotlogs echos the command:

```
Cmd: tdplotlogs -A -j4 -r -X 4 in/ out/
```

When complete tdplotlogs writes out a summary, first the number of files read (output is wrapped here with '' for clarity):

```
plotLogInfo PlotLogInfo <__main__.PlotLogInfo object at 0x101e0da90> \
    Files=23 \
    Bytes=10648531 \
    LogPasses=23 \
    Plots=8 \
    Curve points=229991
```

Then as summary of each plot in detail (output is wrapped here with '' for clarity):

```
('in/1003578128.las', \
    0, \
    'Natural_GR_Spectrometry_3Track.xml', \
    IndexTableValue( \
        scale=100, \
        evFirst='800.5 (FEET)', \
        evLast='3019.5 (FEET)', \
        evInterval='2219.0 (FEET)', \
        curves='CGR_2, POTA, SGR_1, TENS_16, THOR, URAN', \
        numPoints=26213, \
        outPath='out//1003578128.las_0000_Natural_GR_Spectrometry_3Track.xml.svg' \
    )
)
('in/1003578128.las', \
    0, \
    'Porosity_GR_3Track', \
    IndexTableValue( \
        scale=100, \
        evFirst='800.5 (FEET)', \
        evLast='3019.5 (FEET)', \
        evInterval='2219.0 (FEET)', \
        curves='Cali, DRHO, DensityPorosity, GammaRay, NeutronPorosity,␣
↪OLDESTNeutronPorosity, OLDNeutronPorosity, RHOB, SP, SonicPorosity, Tension', \
        numPoints=46170, \
        outPath='out//1003578128.las_0000_Porosity_GR_3Track.svg' \
```

(continues on next page)

(continued from previous page)

```
    )
)

... 8<-------------- Snip -------------->8

('in/1006346987.las', \
    0, 'Sonic_3Track.xml', \
    IndexTableValue(
        scale=100, \
        evFirst='4597.5 (FEET)', \
        evLast='5799.5 (FEET)', \
        evInterval='1202.0 (FEET)', \
        curves='Caliper, DT, DTL_DDBHC, GammaRay, SonicPorosity, TENSION', \
        numPoints=14430, \
        outPath='out//1006346987.las_0000_Sonic_3Track.xml.svg' \
    )
)
```

The fields in each tuple are:

- Input file name.
- LogPass number in the file. For example "Repeat Section" might be 0 and "Main Log" 1.
- LgFormat used for the plot (several plots my be generated from one LogPass).
- **An IndexTableValue object (used to generate the index.html file) that has the following fields:**
  - Plot scale as an integer.
  - First reading and units as an Engineering Value.
  - Last reading and units as an Engineering Value.
  - Log interval and units as an Engineering Value.
  - List of curve names plotted.
  - Total number of data points plotted.
  - The ouput file.

Finally the total number of curve feet plotted and the time it took:

```
Interval*curves: EngVal: 121020.000 (FEET)
  CPU time =    0.043 (S)
Exec. time =   25.119 (S)
Bye, bye!
```

In this case (under Unix) the "CPU Time" is the cumulative amount of CPU time used. As we are using multiprocessing it is the CPU time of the parent process which is very small since it just invokes child processes. The Exec. time is the wall clock time between starting and finishing tdplotlogs.

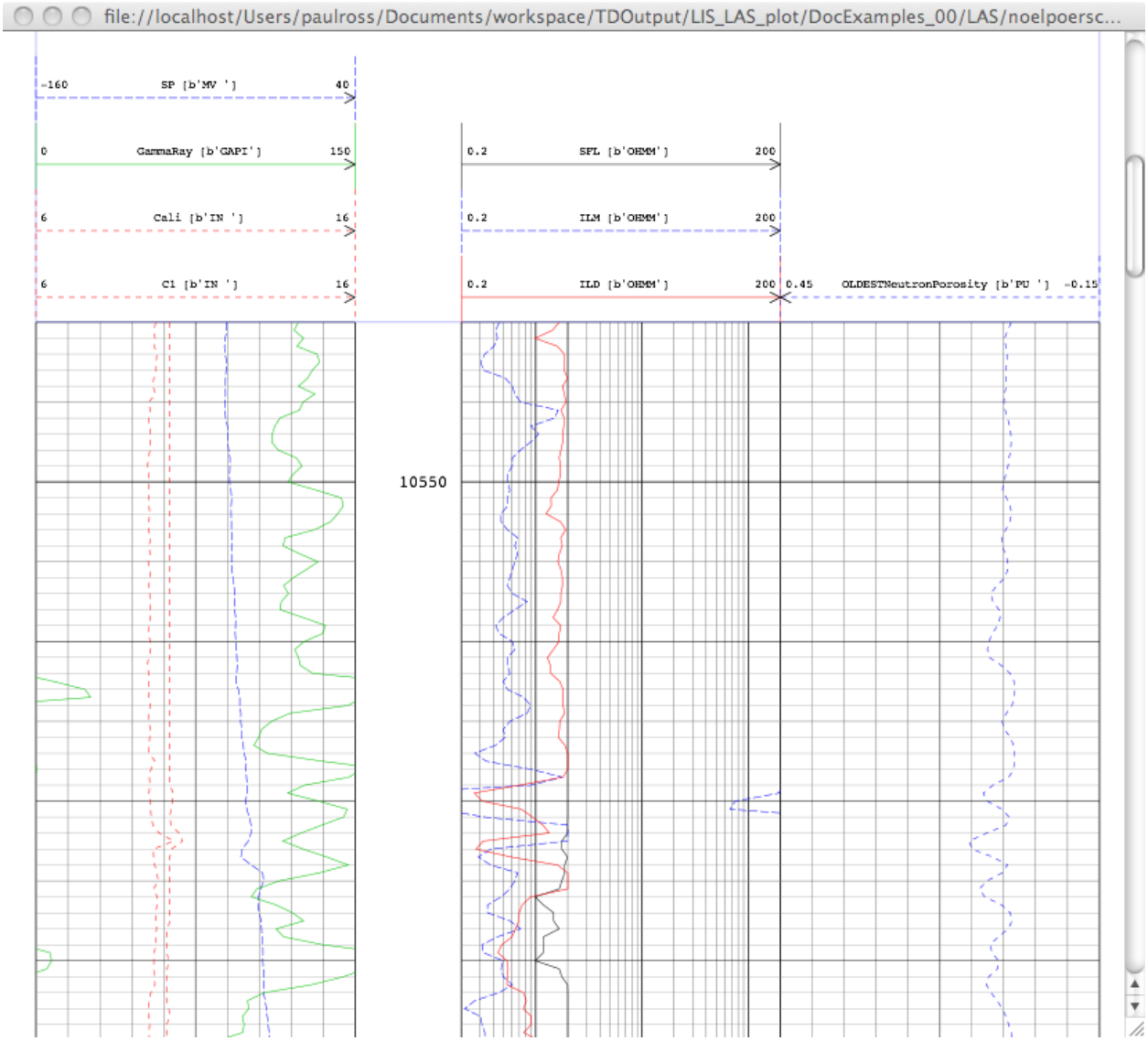
In the output directory will be an index.html file that has a table with the fields that duplicate those on the command line output. It looks like this:

LIS plots in SVG

PlotLogPasses: ../../../TDTestData/LAS/uz_small/

Input	Pass	Film	Scale	From	To	Interval	Curves	Points	Plot
1003578128.las	0	Natural_GR_Spectrometry_3Track.xml	100	800.5 (FEET)	3019.5 (FEET)	2219.0 (FEET)	CGR_2, POTA, SGR_1, TENS_16, THOR, URAN	26213	1003578128.las_0000_Natural_GR_Spectrometry_3Track.xml.svg
		Porosity_GR_3Track	100	800.5 (FEET)	3019.5 (FEET)	2219.0 (FEET)	Cali, DRHO, DensityPorosity, GammaRay, NeutronPorosity, OLDESTNeutronPorosity, OLDNeutronPorosity, RHOB, SP, SonicPorosity, Tension	46170	1003578128.las_0000_Porosity_GR_3Track.svg
		Resistivity_3Track_Correlation.xml	100	800.5 (FEET)	3019.5 (FEET)	2219.0 (FEET)	CALI_8, CILD, GR_9, SP_10, TENS_6	22012	1003578128.las_0000_Resistivity_3Track_Correlation.xml.svg
		Resistivity_3Track_Logrithmic.xml	100	800.5 (FEET)	3019.5 (FEET)	2219.0 (FEET)	CALI_8, GR_9, ILD, ILM, SFL, SP_10, TENS_6	30878	1003578128.las_0000_Resistivity_3Track_Logrithmic.xml.svg
		Sonic_3Track.xml	100	800.5 (FEET)	3019.5 (FEET)	2219.0 (FEET)	Caliper, DTLF_DDBHC, DTLN_DDBHC, GammaRay, SP, SonicPorosity, TENSION	24935	1003578128.las_0000_Sonic_3Track.xml.svg
		Triple_Combo	100	800.5 (FEET)	3019.5 (FEET)	2219.0 (FEET)	Cali, DensityPorosity, GammaRay, ILD, ILM, NeutronPorosity, OLDESTNeutronPorosity, OLDNeutronPorosity, SFL, SP, SonicPorosity, Tension	50755	1003578128.las_0000_Triple_Combo.svg
1003349986.las	0	Porosity_GR_3Track	100	4592.0 (FEET)	5808.0 (FEET)	1216.0 (FEET)	Cali, DRHO, DensityPorosity, GammaRay, OLDESTNeutronPorosity, RHOB	14598	1003349986.las_0000_Porosity_GR_3Track.svg
1003349987.las	0	Sonic_3Track.xml	100	4597.5 (FEET)	5799.5 (FEET)	1202.0 (FEET)	Caliper, DT, DTL_DDBHC, GammaRay, SonicPorosity, TENSION	14430	1003349987.las_0000_Sonic_3Track.xml.svg

The links in the last column are to the SVG plots. Her is a screen shot of one:



Sample Plots

Here is an actual plot from a LAS file and there are many more examples here: *Wireline Plots*.

1.5.2 BIT Command Line Tools

This describes the command line tools that are available for processing BIT files.

Table 2: **BIT Command Line Tools**

Tool Name	Description
`tdbitread`	Generates a summary of an archive of BIT file(s). *Link*
`tdbittolas`	Converts BIT file(s) to LAS file(s). *Link*

Summarise BIT Files with `tdbitread`

Generates a summary from input BIT file or directory.

Arguments

1. The path to the input BIT file or directory.

Options

Option	Description
`-h, --help`	Show this help message and exit.
`--version`	Show program's version number and exit
`-k, --keep-going`	Keep going as far as sensible. [default: False]
`-v, --verbose`	Verbose output, this outputs a representation of table data and DFSRs.
`-r, --recurse`	Process input recursively. [default: False]
`-l LOGLEVEL, --loglevel=LOGLEVEL`	Log Level (debug=10, info=20, warning=30, error=40, critical=50) [default: 20]
`--summary`	Summarise the Frame Data. [default False]

Examples

Command to process a directory of BIT:

```
$ tdbitread -r data/DresserAtlasBIT/special/
```

Output:

```
Cmd: src/TotalDepth/BIT/ReadBIT.py data/DresserAtlasBIT/special/ --summary -r
2021-02-04 13:19:42,290 - ReadBIT.py       -  631 -  8000 - (MainThread) - INFO     -␣
↪Processing: data/DresserAtlasBIT/special/29_10-_3/DWL_FILE/29_10-_3_dwl_DWL_WIRE_
↪1646632.bit
2021-02-04 13:19:42,762 - ReadBIT.py       -  600 -  8000 - (MainThread) - WARNING  -␣
↪The block length 276 does not have equal data for the channels 10. at tell=922504.␣
↪Ignoring rest of file.
2021-02-04 13:19:42,946 - ReadBIT.py       -  631 -  8000 - (MainThread) - INFO     -␣
↪Processing: data/DresserAtlasBIT/special/29_10-_3/DWL_FILE/29_10-_3_dwl_DWL_WIRE_
↪1646636.bit
2021-02-04 13:19:43,175 - ReadBIT.py       -  600 -  8000 - (MainThread) - WARNING  -␣
↪The block length 276 does not have equal data for the channels 10. at tell=441328.␣
↪Ignoring rest of file.
2021-02-04 13:19:43,279 - ReadBIT.py       -  631 -  8000 - (MainThread) - INFO     -␣
↪Processing: data/DresserAtlasBIT/special/29_10-_3Z/DWL_FILE/29_10-_3Z_dwl_DWL_WIRE_
↪1644659.bit
2021-02-04 13:19:43,364 - ReadBIT.py       -  631 -  8000 - (MainThread) - INFO     -␣
↪Processing: data/DresserAtlasBIT/special/29_10-_3Z/DWL_FILE/29_10-_3Z_dwl_DWL_WIRE_
↪1644660.bit
2021-02-04 13:19:43,582 - ReadBIT.py       -  690 -  8000 - (MainThread) - INFO     -␣
↪Count of success=4 errors=0
        Size     Time (s) Rate (ms/Mb) File
     1004032        0.654      682.705 data/DresserAtlasBIT/special/29_10-_3/DWL_FILE/
↪29_10-_3_dwl_DWL_WIRE_1646632.bit
```

(continues on next page)

(continued from previous page)

```
      513536        0.330      673.674 data/DresserAtlasBIT/special/29_10-_3/DWL_FILE/
↪29_10-_3_dwl_DWL_WIRE_1646636.bit
      119276        0.084      736.193 data/DresserAtlasBIT/special/29_10-_3Z/DWL_
↪FILE/29_10-_3Z_dwl_DWL_WIRE_1644659.bit
      339916        0.217      667.974 data/DresserAtlasBIT/special/29_10-_3Z/DWL_
↪FILE/29_10-_3Z_dwl_DWL_WIRE_1644660.bit
Total size 1976760 bytes, total time 1.284 (s)
Rate 681.053 (ms/MB) 1.468 Mb/s
Use -v, --verbose to see more information about each BIT file.
Execution time =    1.293 (S)
Bye, bye!
```

Adding the `-v` flag for more verbosity gives the following, here for one file:

```
$ tdbitread data/DresserAtlasBIT/special/29_10-_3Z/DWL_FILE/29_10-_3Z_dwl_DWL_WIRE_
↪1644659.bit -v
Cmd: src/TotalDepth/BIT/ReadBIT.py data/DresserAtlasBIT/special/29_10-_3Z/DWL_FILE/29_
↪10-_3Z_dwl_DWL_WIRE_1644659.bit -v
2021-02-04 13:21:41,402 - ReadBIT.py       -  631 -  8016 - (MainThread) - INFO     -␣
↪Processing: data/DresserAtlasBIT/special/29_10-_3Z/DWL_FILE/29_10-_3Z_dwl_DWL_WIRE_
↪1644659.bit
File size: 119276 0x1d1ec: data/DresserAtlasBIT/special/29_10-_3Z/DWL_FILE/29_10-_3Z_
↪dwl_DWL_WIRE_1644659.bit
==================== 29_10-_3Z_dwl_DWL_WIRE_1644659.bit ====================
------------------------------ Frame Array [0] ------------------------------
BITFrameArray: ident="0"
   Unknown head: b'\x00\x02\x00\x00'
    Description: b'SHELL EXPRO U.K.      24 OCT 84      MANSFIELD/DODDS                ␣
↪      '
      Unknown A: b'\x00\n\x00\x18\x00'
      Unknown B: b'T  2 9 / 1 0 - 3                                                   ␣
↪         '
      Unknown C: b'\x00\x12\x00\x0b\x00\x06  '
  Channels [10]: ['COND', 'SN  ', 'SP  ', 'GR  ', 'CAL ', 'TEN ', 'SPD ', 'ACQ ', 'AC␣
↪ ', 'RT  ']
   BIT Log Pass: LogPassRange(depth_from=14950.000891089492, depth_to=14590.
↪000869631818, spacing=0.2500000149011621, unknown_a=0.0, unknown_b=16.
↪000000953674373)
   Unknown tail: b'MN239J 1'
    Frame count: 1472
    Frame array:       FrameArray: ID: 0 b'SHELL EXPRO U.K.      24 OCT 84      ␣
↪MANSFIELD/DODDS                '
       <FrameChannel: 'X   ' "Computed X-axis" units: 'b''' count: 1 dimensions: (1,
↪) frames: 1472>
       <FrameChannel: 'COND' "COND" units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1472>
       <FrameChannel: 'SN  ' "SN  " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1472>
       <FrameChannel: 'SP  ' "SP  " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1472>
       <FrameChannel: 'GR  ' "GR  " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1472>
       <FrameChannel: 'CAL ' "CAL " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1472>
       <FrameChannel: 'TEN ' "TEN " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1472>
       <FrameChannel: 'SPD ' "SPD " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1472>
```

(continues on next page)

(continued from previous page)

```
        <FrameChannel: 'ACQ ' "ACQ " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1472>
        <FrameChannel: 'AC  ' "AC  " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1472>
        <FrameChannel: 'RT  ' "RT  " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1472>
------------------------- DONE: Frame Array [0] -------------------------
---------------------------- Frame Array [1] ----------------------------
BITFrameArray: ident="1"
   Unknown head: b'\x00\x02\x00\x00'
    Description: b'SHELL EXPRO U.K.      24 OCT 84      MANSFIELD/DODDS                ␣
↪      '
      Unknown A: b'\x00\n\x00\x18\x00'
      Unknown B: b'T  2 9 / 1 0 - 3                                                   ␣
↪         '
      Unknown C: b'\x00\x11\x00/\x00\r  '
  Channels [10]: ['COND', 'SN  ', 'SP  ', 'GR  ', 'CAL ', 'TEN ', 'SPD ', 'ACQ ', 'AC␣
↪ ', 'RT  ']
   BIT Log Pass: LogPassRange(depth_from=14948.000890970283, depth_to=0.0, spacing=0.
↪2500000149011621, unknown_a=0.0, unknown_b=16.000000953674373)
   Unknown tail: b'MN239J 4'
    Frame count: 1440
    Frame array:       FrameArray: ID: 1 b'SHELL EXPRO U.K.      24 OCT 84      ␣
↪MANSFIELD/DODDS                    '
        <FrameChannel: 'X   ' "Computed X-axis" units: 'b''' count: 1 dimensions: (1,
↪) frames: 1440>
        <FrameChannel: 'COND' "COND" units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1440>
        <FrameChannel: 'SN  ' "SN  " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1440>
        <FrameChannel: 'SP  ' "SP  " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1440>
        <FrameChannel: 'GR  ' "GR  " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1440>
        <FrameChannel: 'CAL ' "CAL " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1440>
        <FrameChannel: 'TEN ' "TEN " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1440>
        <FrameChannel: 'SPD ' "SPD " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1440>
        <FrameChannel: 'ACQ ' "ACQ " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1440>
        <FrameChannel: 'AC  ' "AC  " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1440>
        <FrameChannel: 'RT  ' "RT  " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1440>
------------------------- DONE: Frame Array [1] -------------------------
================= DONE 29_10-_3Z_dwl_DWL_WIRE_1644659.bit =================
Result:       119276        0.085      743.316 data/DresserAtlasBIT/special/29_10-_3Z/
↪DWL_FILE/29_10-_3Z_dwl_DWL_WIRE_1644659.bit
Execution time =    0.085 (S)
Bye, bye!
```

Adding the `--summary` flag for the frame data gives the following:

```
$ tdbitread data/DresserAtlasBIT/special/29_10-_3Z/DWL_FILE/29_10-_3Z_dwl_DWL_WIRE_
↪1644659.bit -v --summary
```

(continues on next page)

(continued from previous page)

```
Cmd: src/TotalDepth/BIT/ReadBIT.py data/DresserAtlasBIT/special/29_10-_3Z/DWL_FILE/29_
↪10-_3Z_dwl_DWL_WIRE_1644659.bit -v --summary
2021-02-04 13:24:12,756 - ReadBIT.py       -  631 -  8140 - (MainThread) - INFO     -␣
↪Processing: data/DresserAtlasBIT/special/29_10-_3Z/DWL_FILE/29_10-_3Z_dwl_DWL_WIRE_
↪1644659.bit
File size: 119276 0x1d1ec: data/DresserAtlasBIT/special/29_10-_3Z/DWL_FILE/29_10-_3Z_
↪dwl_DWL_WIRE_1644659.bit
==================== 29_10-_3Z_dwl_DWL_WIRE_1644659.bit ==================
------------------------------ Frame Array [0] ------------------------------
BITFrameArray: ident="0"
   Unknown head: b'\x00\x02\x00\x00'
    Description: b'SHELL EXPRO U.K.      24 OCT 84      MANSFIELD/DODDS                ␣
↪      '
      Unknown A: b'\x00\n\x00\x18\x00'
      Unknown B: b'T  2 9 / 1 0 - 3                                                    ␣
↪         '
      Unknown C: b'\x00\x12\x00\x0b\x00\x06  '
  Channels [10]: ['COND', 'SN  ', 'SP  ', 'GR  ', 'CAL ', 'TEN ', 'SPD ', 'ACQ ', 'AC␣
↪ ', 'RT  ']
   BIT Log Pass: LogPassRange(depth_from=14950.000891089492, depth_to=14590.
↪000869631818, spacing=0.2500000149011621, unknown_a=0.0, unknown_b=16.
↪000000953674373)
   Unknown tail: b'MN239J 1'
    Frame count: 1472
    Frame array:       FrameArray: ID: 0 b'SHELL EXPRO U.K.      24 OCT 84      ␣
↪MANSFIELD/DODDS                    '
       <FrameChannel: 'X    ' "Computed X-axis" units: 'b''' count: 1 dimensions: (1,
↪) frames: 1472>
       <FrameChannel: 'COND' "COND" units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1472>
       <FrameChannel: 'SN   ' "SN   " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1472>
       <FrameChannel: 'SP   ' "SP   " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1472>
       <FrameChannel: 'GR   ' "GR   " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1472>
       <FrameChannel: 'CAL  ' "CAL  " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1472>
       <FrameChannel: 'TEN  ' "TEN  " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1472>
       <FrameChannel: 'SPD  ' "SPD  " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1472>
       <FrameChannel: 'ACQ  ' "ACQ  " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1472>
       <FrameChannel: 'AC   ' "AC   " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1472>
       <FrameChannel: 'RT   ' "RT   " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1472>
ID   Length          Shape  Count          Min          Max         Mean     Std.Dev.␣
↪      Median  Equal   Inc.   Dec.     Activity        Drift        First ->        ␣
↪Last
X      1472        (1472,)   1472      14582.3        14950      14766.1      106.232␣
↪     14766.1      0      0   1471   2.4427e-05        -0.25        14950 ->      ␣
↪14582.3
COND   1472        (1472,)   1472       0.0001      2249.65      1016.02      520.471␣
↪     1032.29     36    741    694    0.0695624    -0.690478      1015.69 ->       0.
↪0001
```

(continues on next page)

(continued from previous page)

```
SN      1472        (1472,)   1472     -3.18704       24.465       16.718       3.78744␣
↪      17.4128      35    762    674          nan   -0.0109108       16.0499 ->       0.
↪0001
SP      1472        (1472,)   1472     -249.709       0.0001      -244.45       35.8542␣
↪    -249.709   1470      1      0          nan     0.169755     -249.709 ->       0.
↪0001
GR      1472        (1472,)   1472       0.0001      128.979      64.2715      17.9234␣
↪      65.8888    168    637    666    0.0418426   -0.0556673      81.8867 ->       0.
↪0001
CAL     1472        (1472,)   1472     -2.44161       0.0001     -2.36168       0.3465␣
↪    -2.41034    216    633    622          nan   0.00163825     -2.40976 ->       0.
↪0001
TEN     1472        (1472,)   1472       0.0001      4385.71      3619.19      543.754␣
↪      3680.34     44    780    647    0.0198009     -2.18931      3220.48 ->       0.
↪0001
SPD     1472        (1472,)   1472       0.0001       43.074      30.9595      4.88865␣
↪      31.8997     52    682    737    0.0201738   -0.0197124      28.9971 ->       0.
↪0001
ACQ     1472        (1472,)   1472            0            4     0.058426     0.390943␣
↪           0   1433     26     12          nan   6.7981e-08            0 ->       0.
↪0001
AC      1472        (1472,)   1472       0.0001      91.9285      73.2971      15.5518␣
↪      76.4215    116    630    725    0.0242162   -0.0254478      37.4338 ->       0.
↪0001
RT      1472        (1472,)   1472       0.0001      22.6922      1.81547      3.05078␣
↪    0.952113     36    693    742    0.0621832 -0.000669238     0.984549 ->       0.
↪0001
--------------------------- DONE: Frame Array [0] ---------------------------
----------------------------- Frame Array [1] -----------------------------
BITFrameArray: ident="1"
   Unknown head: b'\x00\x02\x00\x00'
    Description: b'SHELL EXPRO U.K.      24 OCT 84      MANSFIELD/DODDS                ␣
↪     '
      Unknown A: b'\x00\n\x00\x18\x00'
      Unknown B: b'T  2 9 / 1 0 - 3                                                    ␣
↪        '
      Unknown C: b'\x00\x11\x00/\x00\r  '
  Channels [10]: ['COND', 'SN  ', 'SP  ', 'GR  ', 'CAL ', 'TEN ', 'SPD ', 'ACQ ', 'AC␣
↪ ', 'RT  ']
   BIT Log Pass: LogPassRange(depth_from=14948.000890970283, depth_to=0.0, spacing=0.
↪2500000149011621, unknown_a=0.0, unknown_b=16.000000953674373)
   Unknown tail: b'MN239J 4'
    Frame count: 1440
    Frame array:       FrameArray: ID: 1 b'SHELL EXPRO U.K.      24 OCT 84      ␣
↪MANSFIELD/DODDS                    '
       <FrameChannel: 'X   ' "Computed X-axis" units: 'b''' count: 1 dimensions: (1,
↪) frames: 1440>
       <FrameChannel: 'COND' "COND" units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1440>
       <FrameChannel: 'SN  ' "SN  " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1440>
       <FrameChannel: 'SP  ' "SP  " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1440>
       <FrameChannel: 'GR  ' "GR  " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1440>
       <FrameChannel: 'CAL ' "CAL " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1440>
```

(continues on next page)

(continued from previous page)

```
        <FrameChannel: 'TEN ' "TEN " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1440>
        <FrameChannel: 'SPD ' "SPD " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1440>
        <FrameChannel: 'ACQ ' "ACQ " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1440>
        <FrameChannel: 'AC  ' "AC  " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1440>
        <FrameChannel: 'RT  ' "RT  " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1440>
ID   Length          Shape  Count          Min          Max         Mean     Std.Dev.␣
↪      Median  Equal   Inc.   Dec.     Activity        Drift        First ->         ␣
↪Last
X      1440        (1440,)   1440      14588.3        14948      14768.1      103.923␣
↪     14768.1      0      0   1439  2.44237e-05        -0.25        14948 ->       ␣
↪14588.3
COND   1440        (1440,)   1440       0.0001       2261.8      1016.27      509.346␣
↪     1053.65     20    757    662    0.0685063    -0.708206      1019.11 ->        0.
↪0001
SN     1440        (1440,)   1440       0.0001      91.6549      18.5481      4.23407␣
↪     19.0641     20    756    663    0.0256921   -0.0636934      91.6549 ->        0.
↪0001
SP     1440        (1440,)   1440     -249.709       0.0001     -239.304      29.3708␣
↪    -242.363     54    708    677          nan     0.164374     -236.534 ->        0.
↪0001
GR     1440        (1440,)   1440       0.0001      131.253      65.1255      17.9535␣
↪     66.5528    155    631    653    0.0393008    -0.055561      79.9523 ->        0.
↪0001
CAL    1440        (1440,)   1440     -2.43995       0.0001     -2.37981     0.289624␣
↪    -2.41493     80    673    686          nan   0.00167468     -2.40976 ->        0.
↪0001
TEN    1440        (1440,)   1440       0.0001      6564.66      3783.73      721.861␣
↪     3678.22     21    786    632    0.0212608     -1.99216      2866.71 ->        0.
↪0001
SPD    1440        (1440,)   1440            0      39.5108      29.5586      4.05069␣
↪     29.9188     26    683    730          inf  6.94927e-08            0 ->        0.
↪0001
ACQ    1440        (1440,)   1440            0            2   0.00972368      0.12322␣
↪           0   1426      8      5          nan  6.94927e-08            0 ->        0.
↪0001
AC     1440        (1440,)   1440       0.0001      91.9285      76.2796      10.8951␣
↪     76.8658     91    633    715     0.023214   -0.0523857      75.3831 ->        0.
↪0001
RT     1440        (1440,)   1440       0.0001      21.3526      1.79013      2.73227␣
↪     0.92928     20    661    758     0.060755 -0.000681828      0.98125 ->        0.
↪0001
-------------------------- DONE: Frame Array [1] --------------------------
================= DONE 29_10-_3Z_dwl_DWL_WIRE_1644659.bit =================
Result:       119276        0.097      848.759 data/DresserAtlasBIT/special/29_10-_3Z/
↪DWL_FILE/29_10-_3Z_dwl_DWL_WIRE_1644659.bit
Execution time =    0.097 (S)
Bye, bye!
```

Converting BIT Files to LAS Files with `tdbittolas`

This takes a BIT file or directory of them and writes out a set of LAS files. A single LAS file is written for each Log Pass so a single BIT file produces one or more LAS files.

The frames in the log pass can be sub-sampled by using `--frame-slice` which speeds things up when processing large files. The `--channels` option can be used to limit channels.

BIT does not allow multiple values per channel.

As BIT files contain very little other than the frame data the generated LAS files are very simple and are missing what many processors would regard as essential data such as well name. These LAS files may have to be edited with data from other sources than the original BIT file to be useful.

LAS File Naming Convention

One BIT file produces one or more LAS files. LAS file names are of the form:

```
{BIT_File}_{logical_file_number:04d}.las
```

Processing a Single BIT File

Given the path out the LAS files will be named `{path_out}_{logical_file_number}.las`

For example `tdbittolas foo.bit bar/baz` might create:

```
bar/baz.bit_0000.las
bar/baz.bit_0001.las
```

and so on.

Processing a Directory of BIT Files

Given the path out the LAS files will be named:

```
{path_out}/{BIT_File}_{logical_file_number}.las
```

For example `tdbittolas foo/ bar/baz` might create:

```
bar/baz.bit_0000.las
bar/baz.bit_0001.las
```

and so on.

The output directory structure will mirror the input directory structure.

Arguments

The first argument is the path to a BIT file or directory. The second argument is the path to write the output to.

Options

-h, --help show this help message and exit

--version show program's version number and exit

-k, --keep-going Keep going as far as sensible. Default: False.

-v, --verbose Increase verbosity, additive [default: 0]

-r, --recurse Process the input recursively. Default: False.

-l LOG_LEVEL, --log-level LOG_LEVEL Log Level as an integer or symbol. (0<->NOTSET, 10<->DEBUG, 20<->INFO, 30<->WARNING, 40<->ERROR, 50<->CRITICAL) [default: 20]

-j JOBS, --jobs JOBS Max processes when multiprocessing.Zero uses number of native CPUs [8]. Negative value disables multiprocessing code. Default: -1.

--frame-slice FRAME_SLICE Do not process all frames but sample or slice the frames. SAMPLE: Sample is of the form "N" so a maximum of N frames, roughly regularly spaced, will be processed. N must be +ve, non-zero integer. Example: "64" - process a maximum of 64 frames. SLICE: Slice the frames is of the form start,stop,step as a comma separated list. Values can be absent or "None". Examples: ",," - every frame, ",,2" - every other frame, ",10," - frames 0 to 9, "4,10,2" - frames 4, 6, 8, "40,-1,4" - every fourth frame from 40 to the end. Results will be truncated by frame array length. Use '?' to see what frames are available [default: ",," i.e. all frames]

--log-process LOG_PROCESS Writes process data such as memory usage as a log INFO line every LOG_PROCESS seconds. If 0.0 no process data is logged. [default: 0.0]

--gnuplot GNUPLOT Directory to write the gnuplot data.

--array-reduction ARRAY_REDUCTION Method to reduce multidimensional channel data to a single value. One of {first,max,mean,median,min} [default: first]

--channels CHANNELS Comma separated list of channels to write out (X axis is always included). Use '?' to see what channels exist without writing anything. [default: ""]

--field-width FIELD_WIDTH Field width for array data [default: 16].

--float-format FLOAT_FORMAT Floating point format for array data [default: ".3f"].

Examples

Finding out what Channels and Frames Exist:

Use `--channels=?` and/or `--frame-slice=?` to see what channels and frames exist in the original BIT file.

```
$ tdbittolas --channels=? --frame-slice=? example_data/BIT/data/29_10-_3Z_dwl_DWL_
↪WIRE_1644659.bit example_data/BIT/LIS
======= File example_data/BIT/data/29_10-_3Z_dwl_DWL_WIRE_1644659.bit =======

  Frame Array: 0
  Channels: "X   ","COND","SN  ","SP  ","GR  ","CAL ","TEN ","SPD ","ACQ ","AC  ","RT␣
↪ "
  X axis: <FrameChannel: 'X   ' "Computed X-axis" units: 'b''' count: 1 dimensions:␣
↪(1,) frames: 1472>
  Frames: 1472 from 14950.000891089492 to 14582.250869169884 interval -0.
↪2500000149011612 [b'']

  Frame Array: 1
  Channels: "X   ","COND","SN  ","SP  ","GR  ","CAL ","TEN ","SPD ","ACQ ","AC  ","RT␣
↪ "
  X axis: <FrameChannel: 'X   ' "Computed X-axis" units: 'b''' count: 1 dimensions:␣
↪(1,) frames: 1440>
  Frames: 1440 from 14948.000890970283 to 14588.250869527512 interval -0.
↪2500000149011612 [b'']


===== END File example_data/BIT/data/29_10-_3Z_dwl_DWL_WIRE_1644659.bit =====
```

Processing a Single File

```
$ tdbittolas example_data/BIT/data/29_10-_3Z_dwl_DWL_WIRE_1644659.bit example_data/
↪BIT/LAS/29_10-_3Z_dwl_DWL_WIRE_1644659.bit
Cmd: /Users/paulross/pyenvs/TotalDepth_3.8_v0.3/bin/tdbittolas example_data/BIT/data/
↪29_10-_3Z_dwl_DWL_WIRE_1644659.bit example_data/BIT/LAS/29_10-_3Z_dwl_DWL_WIRE_
↪1644659.bit
gnuplot version: "b'gnuplot 5.4 patchlevel 1'"
2021-02-05 12:58:18,749 - WriteLAS.py      -  191 - 28222 - (MainThread) - INFO     -␣
↪process_to_las(): Namespace(array_reduction='first', channels='', field_width=16,␣
↪float_format='.3f', frame_slice=',,', gnuplot=None, jobs=-1, keepGoing=False, log_
↪level=20, log_process=0.0, path_in='example_data/BIT/data/29_10-_3Z_dwl_DWL_WIRE_
↪1644659.bit', path_out='example_data/BIT/LAS/29_10-_3Z_dwl_DWL_WIRE_1644659.bit',␣
↪recurse=False, verbose=0)
2021-02-05 12:58:18,749 - WriteLAS.py      -  167 - 28222 - (MainThread) - INFO     -␣
↪index_dir_or_file(): "example_data/BIT/data/29_10-_3Z_dwl_DWL_WIRE_1644659.bit" to
↪"example_data/BIT/LAS/29_10-_3Z_dwl_DWL_WIRE_1644659.bit" recurse: False
2021-02-05 12:58:18,750 - ToLAS.py         -  117 - 28222 - (MainThread) - INFO     -␣
↪Found file type BIT on path example_data/BIT/data/29_10-_3Z_dwl_DWL_WIRE_1644659.bit
2021-02-05 12:58:18,750 - ToLAS.py         -  119 - 28222 - (MainThread) - INFO     -␣
↪Reading BIT file example_data/BIT/data/29_10-_3Z_dwl_DWL_WIRE_1644659.bit
2021-02-05 12:58:18,846 - ToLAS.py         -  125 - 28222 - (MainThread) - INFO     -␣
↪Writing frame array 0 to example_data/BIT/LAS/29_10-_3Z_dwl_DWL_WIRE_1644659.bit_
↪0000.las
2021-02-05 12:58:18,848 - WriteLAS.py      -  521 - 28222 - (MainThread) - INFO     -␣
↪Writing array section with 1,472 frames, 11 channels and 11 values per frame,␣
↪total: 16,192 input values.
```

(continues on next page)

(continued from previous page)

```
2021-02-05 12:58:18,994 - ToLAS.py         -  125 - 28222 - (MainThread) - INFO     -␣
↪Writing frame array 1 to example_data/BIT/LAS/29_10-_3Z_dwl_DWL_WIRE_1644659.bit_
↪0001.las
2021-02-05 12:58:18,995 - WriteLAS.py      -  521 - 28222 - (MainThread) - INFO     -␣
↪Writing array section with 1,440 frames, 11 channels and 11 values per frame,␣
↪total: 15,840 input values.
  Input     Type  Output LAS Count  Time  Ratio  ms/Mb Exception                    ␣
↪                                Path
------- -------- ------- --------- ----- ------ ------ --------- --------------------
↪-------------------------------------
119,276 BIT      549,613         2 0.370 460.8% 3249.8     False "example_data/BIT/
↪data/29_10-_3Z_dwl_DWL_WIRE_1644659.bit"
Writing results returned: 0 files failed.
Execution time =    0.370 (S)
Out of 1 processed 1 files of total size 119,276 input bytes
Wrote 549,613 output bytes, ratio: 460.791% at 3256.2 ms/Mb
Execution time: 0.370 (s)
Bye, bye!
```

The LAS files look like this:

```
~Version Information Section
VERS.           2.0                                   : CWLS Log ASCII Standard -␣
↪VERSION 2.0
WRAP.           NO                                    : One Line per depth step
PROD.           TotalDepth                            : LAS Producer
PROG.           TotalDepth.BIT.ToLAS 0.1.1            : LAS Program name and version
CREA.           2021-02-05 12:58:18.847493 UTC        : LAS Creation date [YYYY-mm-dd␣
↪HH MM SS.us UTC]
SOURCE.         29_10-_3Z_dwl_DWL_WIRE_1644659.bit    : Source File Name
LOGICAL-FILE.   0                                     : Logical File number in the␣
↪Source file
SOURCE_FORMAT.  WESTERN ATLAS BIT FORMAT              : File format of Source file.
#
# Binary block A: b'SHELL EXPRO U.K.      24 OCT 84      MANSFIELD/DODDS                ␣
↪      '
# Binary block B: b'T  2 9 / 1 0 - 3                                                   ␣
↪         '
# BIT Log Pass (claimed): LogPassRange(depth_from=14950.000891089492, depth_to=14590.
↪000869631818, spacing=0.2500000149011621, unknown_a=0.0, unknown_b=16.
↪000000953674373)
#
~Well Information Section
#MNEM.UNIT  DATA                 DESCRIPTION
#----.----  ----                 -----------
STRT.       14950.000891089492   : START
STOP.       14582.250869169884   : STOP
STRP.       -0.2500000149011612  : STEP
~Curve Information Section
#MNEM.UNIT  Curve Description
#---------  -----------------
X    .      : Computed X-axis Dimensions (1,)
COND.       : COND Dimensions (1,)
SN   .      : SN   Dimensions (1,)
SP   .      : SP   Dimensions (1,)
GR   .      : GR   Dimensions (1,)
CAL  .      : CAL  Dimensions (1,)
```

(continues on next page)

(continued from previous page)

```
TEN .        : TEN  Dimensions (1,)
SPD .        : SPD  Dimensions (1,)
ACQ .        : ACQ  Dimensions (1,)
AC  .        : AC   Dimensions (1,)
RT  .        : RT   Dimensions (1,)
# Array processing information:
# Frame Array: ID: 0 description: b'SHELL EXPRO U.K.      24 OCT 84      MANSFIELD/
↪DODDS                      '
# All [11] original channels reproduced here.
# Where a channel has multiple values the reduction method is by "first" value.
# Maximum number of original frames: 1472
# Requested frame slicing: <Slice on length=1472 start=0 stop=1472 step=1>, total␣
↪number of frames presented here: 1472
~A          X                COND             SN               SP               GR    ␣
↪            CAL              TEN              SPD              ACQ               AC   ␣
↪             RT
       14950.001         1015.693          16.050         -249.709           81.887 ␣
↪          -2.410          3220.477          28.997            0.000           37.434␣
↪            0.985
       14949.751         1015.693          16.050         -249.709           81.887 ␣
↪          -2.410          3220.477          28.997            0.000           37.434␣
↪            0.985
       14949.501         1015.693          16.050         -249.709           81.887 ␣
↪          -2.410          3220.477          28.997            0.000           37.434␣
↪            0.985
       14949.251         1015.693          16.050         -249.709           81.887 ␣
↪          -2.410          3220.477          28.997            0.000           37.434␣
↪            0.985
```

Processing a Directory

Use the `-r` option to process recursively. The output directory will mirror the input directory.

```
$ tdbittolas -r example_data/BIT/data example_data/BIT/LAS
Cmd: /Users/paulross/pyenvs/TotalDepth_3.8_v0.3/bin/tdbittolas -r example_data/BIT/
↪data example_data/BIT/LAS
gnuplot version: "b'gnuplot 5.4 patchlevel 1'"
2021-02-05 13:00:32,879 - WriteLAS.py      -  191 - 28324 - (MainThread) - INFO     -␣
↪process_to_las(): Namespace(array_reduction='first', channels='', field_width=16,␣
↪float_format='.3f', frame_slice=',,', gnuplot=None, jobs=-1, keepGoing=False, log_
↪level=20, log_process=0.0, path_in='example_data/BIT/data', path_out='example_data/
↪BIT/LAS', recurse=True, verbose=0)
2021-02-05 13:00:32,879 - WriteLAS.py      -  167 - 28324 - (MainThread) - INFO     -␣
↪index_dir_or_file(): "example_data/BIT/data" to "example_data/BIT/LAS" recurse: True
2021-02-05 13:00:32,880 - ToLAS.py         -  117 - 28324 - (MainThread) - INFO     -␣
↪Found file type BIT on path example_data/BIT/data/29_10-_3Z_dwl_DWL_WIRE_1644659.bit
2021-02-05 13:00:32,880 - ToLAS.py         -  119 - 28324 - (MainThread) - INFO     -␣
↪Reading BIT file example_data/BIT/data/29_10-_3Z_dwl_DWL_WIRE_1644659.bit
2021-02-05 13:00:32,962 - ToLAS.py         -  125 - 28324 - (MainThread) - INFO     -␣
↪Writing frame array 0 to example_data/BIT/LAS/29_10-_3Z_dwl_DWL_WIRE_1644659.bit_
↪0000.las
2021-02-05 13:00:32,964 - WriteLAS.py      -  521 - 28324 - (MainThread) - INFO     -␣
↪Writing array section with 1,472 frames, 11 channels and 11 values per frame,␣
↪total: 16,192 input values.
2021-02-05 13:00:33,076 - ToLAS.py         -  125 - 28324 - (MainThread) - INFO     -␣
↪Writing frame array 1 to example_data/BIT/LAS/29_10-_3Z_dwl_DWL_WIRE_1644659.bit_
↪0001.las
```

(continues on next page)

(continued from previous page)

```
2021-02-05 13:00:33,076 - WriteLAS.py      -  521 - 28324 - (MainThread) - INFO     -␣
↪Writing array section with 1,440 frames, 11 channels and 11 values per frame,␣
↪total: 15,840 input values.
  Input     Type  Output LAS Count  Time  Ratio  ms/Mb Exception                    ␣
↪                                Path
------- -------- ------- --------- ----- ------ ------ --------- --------------------
↪-------------------------------------
119,276 BIT      549,613         2 0.300 460.8% 2634.7     False "example_data/BIT/
↪data/29_10-_3Z_dwl_DWL_WIRE_1644659.bit"
Writing results returned: 0 files failed.
Execution time =    0.301 (S)
Out of 1 processed 1 files of total size 119,276 input bytes
Wrote 549,613 output bytes, ratio: 460.791% at 2644.1 ms/Mb
Execution time: 0.301 (s)
Bye, bye!
```

1.5.3 LAS Command Line Tools

This describes the command line tools that are available for processing LAS files.

Table 3: **LAS Command Line Tools**

Tool Name	Description
`tdlastohtml`	Generates a HTML page(s) about LAS file(s). *Link*
`tdlasreadlasfiles`	Summarises LAS file(s). *Link*

Summarise LAS Files in HTML with `tdlastohtml`

Generates HTML from input LAS file or directory to an output destination.

Arguments

1. The path to the input LAS file or directory.
2. The path to the output file or directory, any directories will be created as necessary.

Options

-h, --help show this help message and exit

--version show program's version number and exit

-k, --keep-going Keep going as far as sensible. Default: False.

-v, --verbose Increase verbosity, additive [default: 0]

-r, --recurse Process the input recursively. Default: False.

-l LOG_LEVEL, --log-level LOG_LEVEL Log Level as an integer or symbol. (0<->NOTSET, 10<->DEBUG, 20<->INFO, 30<->WARNING, 40<->ERROR, 50<->CRITICAL) [default: 20]

-j JOBS, --jobs JOBS Max processes when multiprocessing.Zero uses number of native CPUs [8]. Negative value disables multiprocessing code. Default: -1.

--frame-slice FRAME_SLICE Do not process all frames but sample or slice the frames. SAMPLE: Sample is of the form "N" so a maximum of N frames, roughly regularly spaced, will be processed. N must be +ve, non-zero integer. Example: "64" - process a maximum of 64 frames. SLICE: Slice the frames is of the form start,stop,step as a comma separated list. Values can be absent or "None". Examples: ",," - every frame, ",,2" - every other frame, ",10," - frames 0 to 9, "4,10,2" - frames 4, 6, 8, "40,-1,4" - every fourth frame from 40 to the end. Results will be truncated by frame array length. Use '?' to see what frames are available [default: ",," i.e. all frames]

--log-process LOG_PROCESS Writes process data such as memory usage as a log INFO line every LOG_PROCESS seconds. If 0.0 no process data is logged. [default: 0.0]

--gnuplot GNUPLOT Directory to write the gnuplot data.

-g, --glob File match pattern. Default: None.

Examples

Command to process a directory of LAS:

```
$ tdlastohtml example_data/LAS/data/ example_data/LAS/HTML/
```

Output:

```
$ tdlastohtml example_data/LAS/data/ example_data/LAS/HTML/
Cmd: /Users/paulross/pyenvs/TotalDepth_3.8_v0.3/bin/tdlastohtml example_data/LAS/data/
 ↪ example_data/LAS/HTML/
gnuplot version: "b'gnuplot 5.4 patchlevel 1'"
2021-02-06 11:13:20,527 - LASToHTML.py      -  440 - 41351 - (MainThread) - INFO     -␣
 ↪scan_dir_or_file(): "example_data/LAS/data" to "example_data/LAS/HTML" recurse:␣
 ↪False
2021-02-06 11:13:20,529 - LASToHTML.py      -  351 - 41351 - (MainThread) - INFO     -␣
 ↪Scanning file type "ASCII" from "example_data/LAS/data/.DS_Store" to "example_data/
 ↪LAS/HTML/.DS_Store.html"
2021-02-06 11:13:20,530 - LASToHTML.py      -  351 - 41351 - (MainThread) - INFO     -␣
 ↪Scanning file type "LAS2.0" from "example_data/LAS/data/1000079714.las" to "example_
 ↪data/LAS/HTML/1000079714.las.html"
2021-02-06 11:13:20,530 - LASToHTML.py      -  353 - 41351 - (MainThread) - INFO     -␣
 ↪scan_a_single_file(): "example_data/LAS/data/1000079714.las" to "example_data/LAS/
 ↪HTML/1000079714.las.html"
2021-02-06 11:13:20,614 - LASToHTML.py      -  351 - 41351 - (MainThread) - INFO     -␣
 ↪Scanning file type "LAS2.0" from "example_data/LAS/data/206_05a-_3_DWL_DWL_WIRE_
 ↪258276498_0_2000T.las" to "example_data/LAS/HTML/206_05a-_3_DWL_DWL_WIRE_258276498_
 ↪0_2000T.las.html"
2021-02-06 11:13:20,614 - LASToHTML.py      -  353 - 41351 - (MainThread) - INFO     -␣
 ↪scan_a_single_file(): "example_data/LAS/data/206_05a-_3_DWL_DWL_WIRE_258276498_0_
 ↪2000T.las" to "example_data/LAS/HTML/206_05a-_3_DWL_DWL_WIRE_258276498_0_2000T.las.
 ↪html"
2021-02-06 11:13:20,679 - LASToHTML.py      -  351 - 41351 - (MainThread) - INFO     -␣
 ↪Scanning file type "LAS2.0" from "example_data/LAS/data/BASIC_FILE_0_50.las" to
 ↪"example_data/LAS/HTML/BASIC_FILE_0_50.las.html"
```

(continues on next page)

(continued from previous page)

```
2021-02-06 11:13:20,680 - LASToHTML.py      -  353 - 41351 - (MainThread) - INFO     -␣
↪scan_a_single_file(): "example_data/LAS/data/BASIC_FILE_0_50.las" to "example_data/
↪LAS/HTML/BASIC_FILE_0_50.las.html"
2021-02-06 11:13:20,724 - LASToHTML.py      -  382 - 41351 - (MainThread) - INFO     -␣
↪_write_indexes(): result map size 4
2021-02-06 11:13:20,725 - ToHTML.py         -  207 - 41351 - (MainThread) - INFO     -␣
↪Opening index file at /Users/paulross/PycharmProjects/TotalDepth/example_data/LAS/
↪HTML/index.html
2021-02-06 11:13:20,730 - ToHTML.py         -  240 - 41351 - (MainThread) - INFO     -␣
↪Completed index file at /Users/paulross/PycharmProjects/TotalDepth/example_data/LAS/
↪HTML/index.html
2021-02-06 11:13:20,730 - LASToHTML.py      -  399 - 41351 - (MainThread) - INFO     -␣
↪Wrote indexes: ['/Users/paulross/PycharmProjects/TotalDepth/example_data/LAS/HTML/
↪index.html']
Common path: example_data/LAS/data
         Size In   Size Out     Time  Ratio %    ms/Mb Fail? Path
---------------- ---------- -------- -------- -------- ----- ----
           6,148          0    0.000   0.000%      0.0 False ".DS_Store"
          80,697     12,892    0.083  15.976%   1078.0 False "1000079714.las"
          87,448     53,444    0.065  61.115%    779.8 False "206_05a-_3_DWL_DWL_WIRE_
↪258276498_0_2000T.las"
          62,494     26,021    0.044  41.638%    736.1 False "BASIC_FILE_0_50.las"
Processed 4 files and 236,787 bytes in 0.203 s, 900.0 ms/Mb
Bye, bye!
```

For each file the output lists:

- Input file.
- Output HTML file.
- File size.
- Execution time.

In the output directory there will be an index.html file which has the columns:

Path File Type Sections Channels Frames STRT STOP STEP Size Time

- The name of the LAS file.
- LAS file type.
- Number of sections.
- Recorded channels.
- Number of data frames.
- Start of log pass.
- End of log pass.
- Frame step.
- The size of the LAS file.
- Execution time.

In the linked HTML file is a summary of the content of the LAS file.

Summarise LAS Files with `tdlasreadlasfiles`

Reads an input LAS file or directory and summarises it by showing mnemonics, curves and well site data.

Arguments

1. The path to the input LAS file or directory.

Options

-h, --help show this help message and exit

--version show program's version number and exit

-k, --keep-going Keep going as far as sensible. Default: False.

-v, --verbose Increase verbosity, additive [default: 0]

-r, --recurse Process the input recursively. Default: False.

-l LOG_LEVEL, --log-level LOG_LEVEL Log Level as an integer or symbol. (0<->NOTSET, 10<->DEBUG, 20<->INFO, 30<->WARNING, 40<->ERROR, 50<->CRITICAL) [default: 20]

--log-process LOG_PROCESS Writes process data such as memory usage as a log INFO line every LOG_PROCESS seconds. If 0.0 no process data is logged. [default: 0.0]

--gnuplot GNUPLOT Directory to write the gnuplot data.

-m, --mnemonic Output Mnemonic map. Default: False.

-c, --curve Output Curve map. Default: False.

-u, --unit Output Units map. Default: False.

-w, --wsd Output Well Site Data map. Default: False.

-p, --param Output Parameter section mnemonics and their most popular description and a map of themnemonic frequency. Default: False.

-s, --size-time Output parser's size vs time performance. Default: False.

-a, --all Output all, equivalent to -mcuwps. Default: False.

Examples

Listing Menmonics

Use the `-m` option to summarise the menmonics and their descriptions:

```
$ tdlasreadlasfiles example_data/LAS/data/ -m
Cmd: /Users/paulross/pyenvs/TotalDepth_3.8_v0.3/bin/tdlasreadlasfiles example_data/
↪LAS/data/ -m
gnuplot version: "b'gnuplot 5.4 patchlevel 1'"
----------- All  mnemonics and their (most popular) description -----------
{
"ALTDPCHAN"                      : "Name Of Alternate Depth Channel
↪              ", # Out of 1
```

(continues on next page)

(continued from previous page)

```
"AMD"                              : "Azimuth Of Maximum Deviation                                ␣
↪               ", # Out of 1
"AOFF"                             : "Alphanumeric To Film Flag                                   ␣
↪               ", # Out of 1
"APD"                              : "Depth Above Pd                                              ␣
↪               ", # Out of 2
"API"                              : "                                                            ␣
↪               ", # Out of 2
"APIN"                             : "Api S/N                                                     ␣
↪               ", # Out of 2
"BG"                               : "Gas Formation Volume Factor, Bg                             ␣
↪               ", # Out of 1
"BHS"                              : "Borehole Status                                             ␣
↪               ", # Out of 1
"BHT"                              : "Bottom Hole Temperature (Used In Calculations)              ␣
↪               ", # Out of 1
"BLI"                              : "Bottom Log Interval                                         ␣
↪               ", # Out of 1
"BO"                               : "Oil Formation Volume Factor, Bo                             ␣
↪               ", # Out of 1
"BPP"                              : "Bubble Point Pressure                                       ␣
↪               ", # Out of 1
...
```

Listing Curves

Use the `-c` option to summarise the curves and their descriptions:

```
$ tdlasreadlasfiles example_data/LAS/data/ -c
Cmd: /Users/paulross/pyenvs/TotalDepth_3.8_v0.3/bin/tdlasreadlasfiles example_data/
↪LAS/data/ -c
gnuplot version: "b'gnuplot 5.4 patchlevel 1'"
--------- All Curve mnemonics and their (most popular) description --------
{
"DEPT"                             : "Depth Curve                                                 ␣
↪               ", # Out of 2
"DEPT_SL"                          : "Station Logging Depth Dimensions (1,)                       ␣
↪               ", # Out of 1
"DHTN"                             : "Dhtn/Ch Tension Dimensions (1,)                             ␣
↪               ", # Out of 1
"ETIM"                             : "Etim/Elapsed Time Dimensions (1,)                           ␣
↪               ", # Out of 1
"GR"                               : "Gamma Ray                                                   ␣
↪               ", # Out of 2
"TDEP"                             : "Second River Depth Dimensions (1,)                          ␣
↪               ", # Out of 1
"TENS"                             : "Tens/Tension Dimensions (1,)                                ␣
↪               ", # Out of 1
"TENS_SL"                          : "Cable Tension Dimensions (1,)                               ␣
↪               ", # Out of 1
"TIME"                             : "Second River Time Dimensions (1,)                           ␣
↪               ", # Out of 1
}
------ DONE: All Curve mnemonics and their (most popular) description -----
```

Listing Units

Use the `-u` option to summarise the channels and their units:

```
$ tdlasreadlasfiles example_data/LAS/data/ -u
Cmd: /Users/paulross/pyenvs/TotalDepth_3.8_v0.3/bin/tdlasreadlasfiles example_data/
↪LAS/data/ -u
gnuplot version: "b'gnuplot 5.4 patchlevel 1'"
------------------------- Channels and their Units ------------------------
{
"DEPT"     : "Counter({'F': 1, 'm': 1})",
"DEPT_SL"  : "Counter({'0.1': 1})",
"DHTN"     : "Counter({'lbs': 1})",
"ETIM"     : "Counter({'min': 1})",
"GR"       : "Counter({'GAPI': 1, 'api': 1})",
"TDEP"     : "Counter({'0.1': 1})",
"TENS"     : "Counter({'lbs': 1})",
"TENS_SL"  : "Counter({'lbf': 1})",
"TIME"     : "Counter({'ms': 1})",
}
---------------------- DONE: Channels and their Units ----------------------
```

Listing Well Site Data

Use the `-w` option to summarise the well site data and its frequency:

```
$ tdlasreadlasfiles example_data/LAS/data/ -w
Cmd: /Users/paulross/pyenvs/TotalDepth_3.8_v0.3/bin/tdlasreadlasfiles example_data/
↪LAS/data/ -w
gnuplot version: "b'gnuplot 5.4 patchlevel 1'"
------ Count of well site mnemonics and the % of files that have them -----
{
"API"                              : ""
↪             , #        3  100.00%
"CNTY"                             : ""
↪             , #        2   66.67%
"COMP"                             : ""
↪             , #        3  100.00%
"CORN"                             : "Reference Section Corner For Footage"
↪             , #        1   33.33%
"COUN"                             : "County"
↪             , #        1   33.33%
"CTRY"                             : ""
↪             , #        2   66.67%
"DATE"                             : ""
↪             , #        3  100.00%
"FLD"                              : ""
↪             , #        3  100.00%
"FTE"                              : "Feet East From Reference Section Corner"
↪             , #        1   33.33%
"FTN"                              : "Feet North From Reference Section Corner"
↪             , #        1   33.33%
"LAT"                              : "Latitude North (Kgs,Leo3.6)"
↪             , #        1   33.33%
"LEAS"                             : "Lease Name"
↪             , #        1   33.33%
```

(continues on next page)

(continued from previous page)

```
"LOC"                                : ""
↪               , #         3  100.00%
"LON"                                : "Longitude West (Kgs, Leo3.6)"
↪               , #         1   33.33%
"NULL"                               : ""
↪               , #         3  100.00%
"PM"                                 : "Principal Meridian"
↪               , #         1   33.33%
"PROV"                               : ""
↪               , #         2   66.67%
"RANG"                               : "Range"
↪               , #         1   33.33%
"SECT"                               : "Section"
↪               , #         1   33.33%
"SPOT"                               : "Spot Location"
↪               , #         1   33.33%
"SRVC"                               : ""
↪               , #         2   66.67%
"STAT"                               : "State Name"
↪               , #         3  100.00%
"STEP"                               : "Step (Average)"
↪               , #         3  100.00%
"STOP"                               : "Stop Depth"
↪               , #         3  100.00%
"STRT"                               : "Start X"
↪               , #         3  100.00%
"TOWN"                               : "Township"
↪               , #         1   33.33%
"UWI"                                : ""
↪               , #         2   66.67%
"WELL"                               : ""
↪               , #         3  100.00%
}
--- DONE: Count of well site mnemonics and the % of files that have them --
```

1.5.4 LIS Command Line Tools

This describes the command line tools that are available for processing LIS files.

Table 4: **LIS Command Line Tools**

Tool Name	Description
`tdlistohtml`	Generates a HTML page(s) about LIS file(s). *Link*
`tdlisplotlogpasses`	Plots LIS log data as SVG pages. *Link*
`tdlisdumpframeset`	Writes out the frame data as a CSV file. *Link*
`tdlisindex`	Indexes a LIS file. *Link*
`tdlistablehistogram`	Analyses the contents of table Logical Records. *Link*
`tdlisscanphysrec`	Scans all the Physical Records. *Link*
`tdlisscanlogidata`	Scans the logical data. *Link*
`tdlisscanlogirecord`	Scans all Logical records. *Link*

Summarise LIS Files in HTML with `tdlistohtml`

Generates HTML from input LIS file or directory to an output destination.

Arguments

1. The path to the input LIS file or directory.
2. The path to the output file or directory, any directories will be created as necessary.

Options

Option	Description
`--version`	Show program's version number and exit
`-h, --help`	Show this help message and exit.
`-g, --glob`	File pattern match. [default none]
`-j JOBS, --jobs=JOBS`	Max processes when multiprocessing. Zero uses number of native CPUs [8]. -1 disables multiprocessing. [default: -1]
`-k, --keep-going`	Keep going as far as sensible. [default: False]
`-l LOGLEVEL, --loglevel=LOGLEVEL`	Log Level (debug=10, info=20, warning=30, error=40, critical=50) [default: 20]
`-v, --verbose`	Verbose output, this outputs a representation of table data and DFSRs.
`-r, --recursive`	Process input recursively. [default: False]

Examples

Command to process a directory of LIS:

```
$ ``tdlistohtml`` Simple/ LIS_plot/Simple_00/
```

Output:

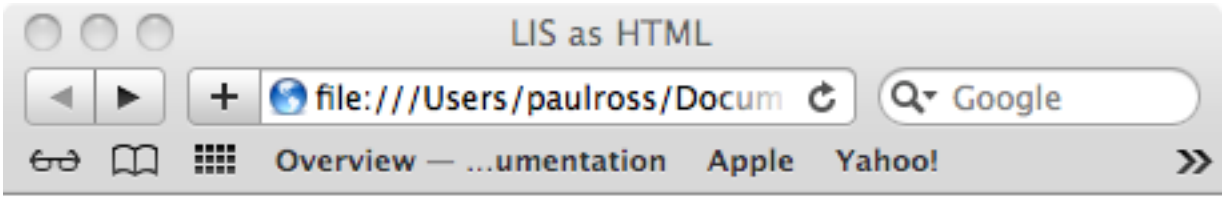
```
Cmd: ``tdlistohtml`` Simple/ LIS_plot/Simple_00/
plotLogInfo:
FileInfo: "Simple/LIS.lis" -> "LIS_plot/Simple_00/LIS.lis.html" 17 (kb) LR count=4␣
↪t=0.070
FileInfo: "Simple/RW.lis" -> "LIS_plot/Simple_00/RW.lis.html" 843 (kb) LR count=12␣
↪t=3.206
FileInfo: "Simple/RW_No_TIF.lis" -> "LIS_plot/Simple_00/RW_No_TIF.lis.html" 833 (kb)␣
↪LR count=12 t=3.200
  CPU time =     6.568 (S)
Exec. time =     6.568 (S)
Bye, bye!
```

For each file the output lists:

- Input file.
- Output HTML file.
- File size.
- Count of Logical Records.

- Execution time.

In the output directory there will be an index.html file, for example:



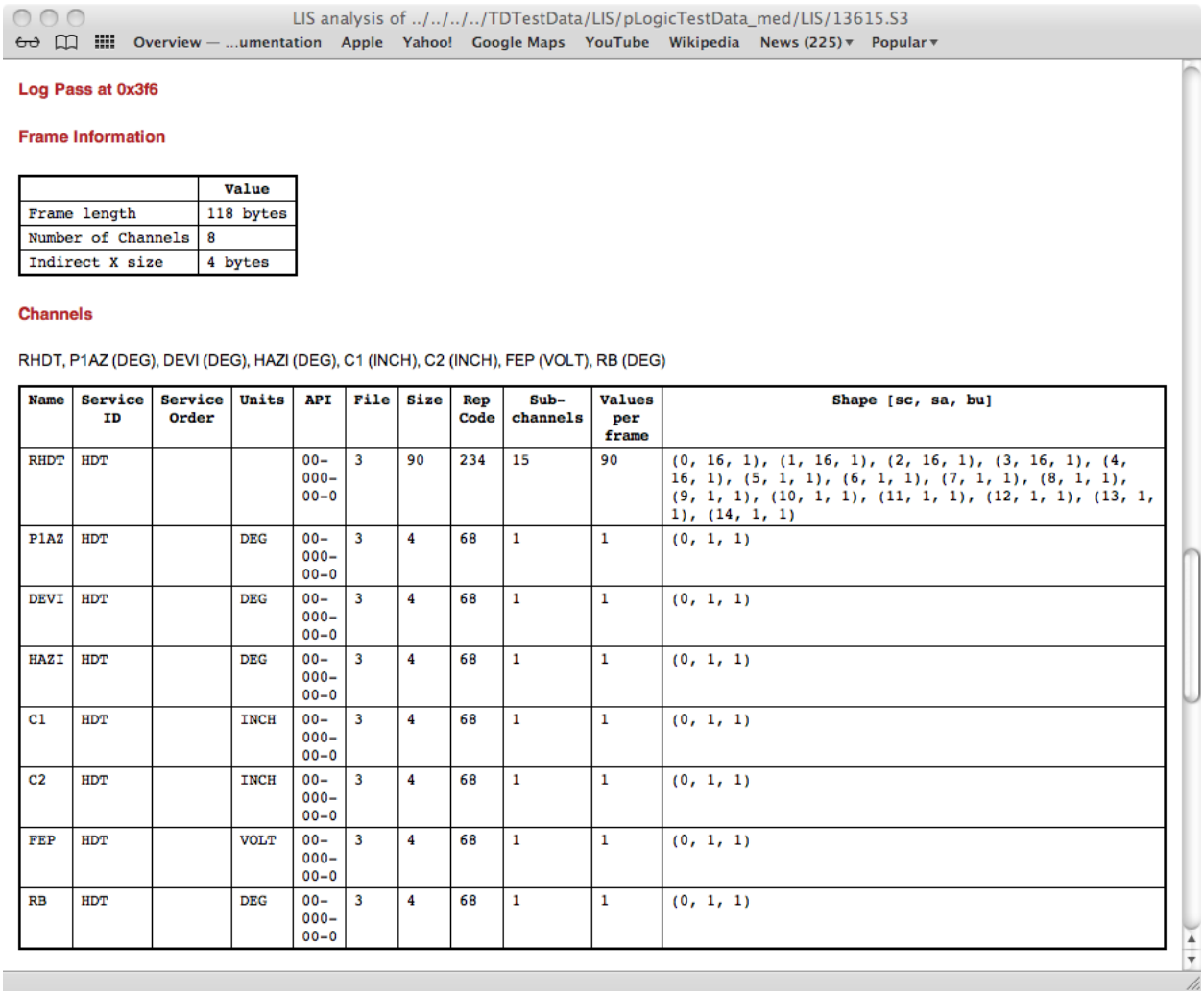
LIS File	Size (MB)	Record Entries	CPU Time (s)	Rate (MB/s)
2851.S1	0.026	14	0.117	0.219
2851.S2	0.176	11	1.110	0.159
2851.S3	0.161	11	1.012	0.159
2851.S4	0.052	11	0.310	0.169
2851.S5	0.058	11	0.346	0.169
2851.S6	0.028	11	0.149	0.186
2851.S7	0.236	11	1.504	0.157
2951.S1	0.182	18	0.659	0.277
2951.S2	0.097	15	0.367	0.264
2953.S10	0.041	9	0.230	0.179
2953.S12	0.016	10	0.068	0.236
2953.S13	0.084	10	0.582	0.145
2953.S14	0.031	19	0.136	0.227
2953.S15	0.015	11	0.065	0.232
2953.S16	0.071	11	0.484	0.148
2953.S17	0.031	17	0.133	0.233
2953.S18	0.015	10	0.065	0.234
2953.S19	0.074	10	0.437	0.169
2955.S1	0.028	10	0.117	0.238
2955.S2	1.487	12	9.463	0.157
Totals	2.910	242	17.352	0.168

The columns are:

- The name of the LIS file.
- The size of the LIS file.
- Count of Logical Records.
- Execution time.
- Processing rate.

In the linked HTML file is a summary of the content of the LIS file.

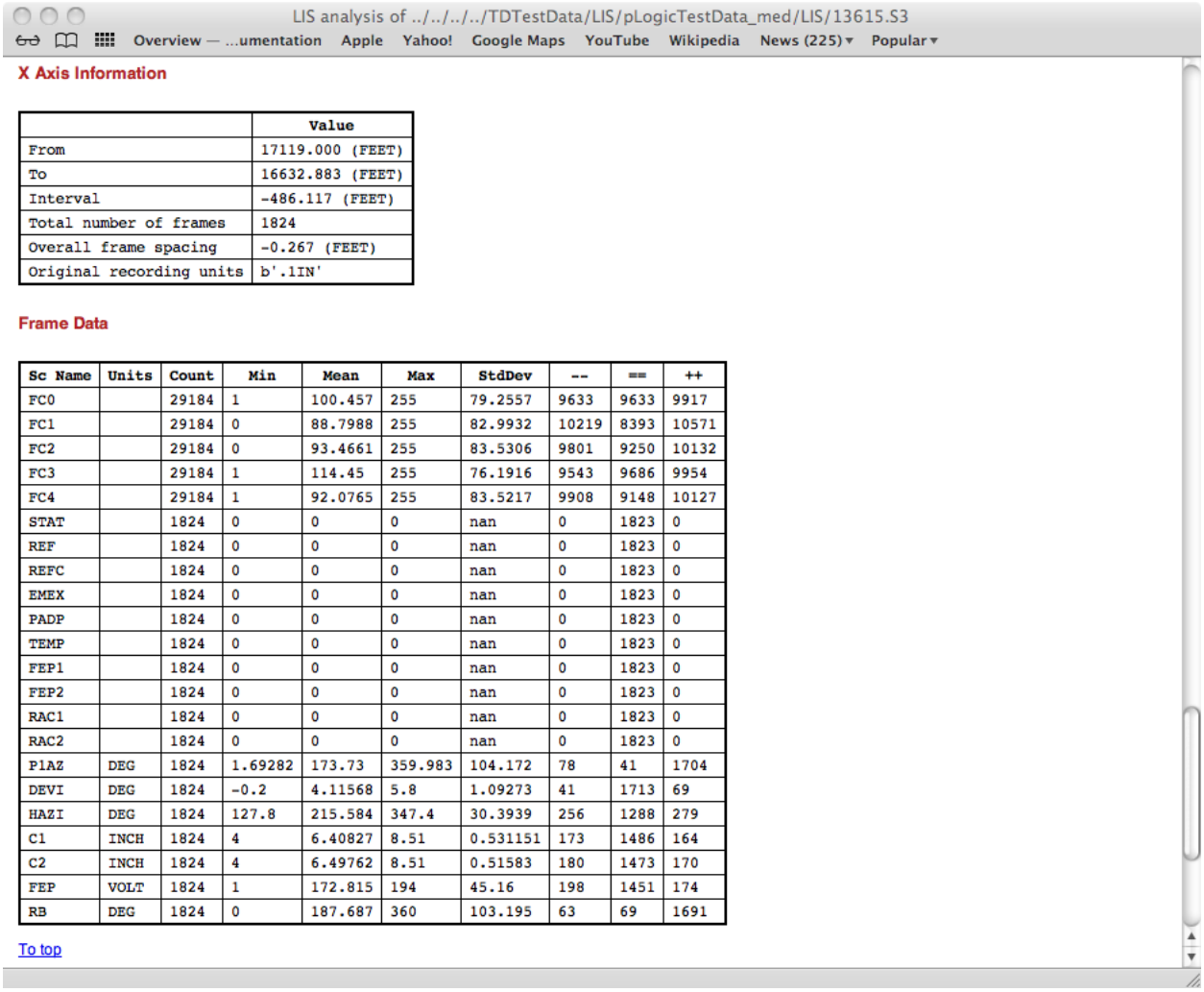
The Log Pass merits several entries, the first summarises the frame shape and the shape of each channel, for example:



	Value
Frame length	118 bytes
Number of Channels	8
Indirect X size	4 bytes

Name	Service ID	Service Order	Units	API	File	Size	Rep Code	Sub-channels	Values per frame	Shape [sc, sa, bu]
RHDT	HDT			00-000-00-0	3	90	234	15	90	(0, 16, 1), (1, 16, 1), (2, 16, 1), (3, 16, 1), (4, 16, 1), (5, 1, 1), (6, 1, 1), (7, 1, 1), (8, 1, 1), (9, 1, 1), (10, 1, 1), (11, 1, 1), (12, 1, 1), (13, 1, 1), (14, 1, 1)
P1AZ	HDT		DEG	00-000-00-0	3	4	68	1	1	(0, 1, 1)
DEVI	HDT		DEG	00-000-00-0	3	4	68	1	1	(0, 1, 1)
HAZI	HDT		DEG	00-000-00-0	3	4	68	1	1	(0, 1, 1)
C1	HDT		INCH	00-000-00-0	3	4	68	1	1	(0, 1, 1)
C2	HDT		INCH	00-000-00-0	3	4	68	1	1	(0, 1, 1)
FEP	HDT		VOLT	00-000-00-0	3	4	68	1	1	(0, 1, 1)
RB	HDT		DEG	00-000-00-0	3	4	68	1	1	(0, 1, 1)

Then there is a couple of tables, the first summarises the X axis and the second summarises each channel (min, max mean etc.), for example:



Plots LIS log data as SVG pages.

TODO:

Generates HTML from input LIS file or directory to an output destination.

Arguments

1. The path to the input LIS file or directory.
2. The path to the output file or directory, any directories will be created as necessary.

Options

Option	Description
`--version`	Show program's version number and exit
`-h, --help`	Show this help message and exit.
`-g, --glob`	File pattern match. [default none]
`-j JOBS, --jobs=JOBS`	Max processes when multiprocessing. Zero uses number of native CPUs [8]. -1 disables multiprocessing. [default: -1]
`-k, --keep-going`	Keep going as far as sensible. [default: False]
`-l LOGLEVEL, --loglevel=LOGLEVEL`	Log Level (debug=10, info=20, warning=30, error=40, critical=50) [default: 20]
`-v, --verbose`	Verbose output, this outputs a representation of table data and DFSRs.
`-r, --recursive`	Process input recursively. [default: False]

Examples

Command to process a directory of LIS:

```
$ ``tdlistohtml`` Simple LIS_plot/Simple_00
```

Output:

```
x
```

Scanning Physical Records in LIS Files with `tdlisscanphysrec`

Scans a LIS79 file and reports the Physical Record structure.

Arguments

One argument that will be treated as a path to a LIS file.

Options

Option	Description
`--version`	Show program's version number and exit
`-h, --help`	Show this help message and exit.
`-k, --keep-going`	Keep going as far as sensible. [default: False]
`-l LOGLEVEL, --loglevel=LOGLEVEL`	Log Level (debug=10, info=20, warning=30, error=40, critical=50) [default: 20]

Examples

Example of scanning a non-TIF encoded file:

```
$ ``tdlisscanphysrec`` LIS.lis
Cmd: ScanPhysRec.py LIS.lis
PR:     tell()  Length    Attr  LD_len  RecNum  FilNum  ChkSum   LR Attr [Total LD]
------------------------------------ start -------------------------------------
PR: 0x       0      62  0x   0      58  ------  ------  ------ 0x80 0x00 [      58]
PR: 0x      3e    1024  0x   1    1020  ------  ------  ------ 0x22 0x00
PR: 0x     43e    1024  0x   3    1020  ------  ------  ------ + --   --
PR: 0x     83e    1024  0x   3    1020  ------  ------  ------ + --   --
PR: 0x     c3e    1024  0x   3    1020  ------  ------  ------ + --   --
PR: 0x    103e    1024  0x   3    1020  ------  ------  ------ + --   --
PR: 0x    143e    1024  0x   3    1020  ------  ------  ------ + --   --
PR: 0x    183e    1024  0x   3    1020  ------  ------  ------ + --   --
PR: 0x    1c3e    1024  0x   3    1020  ------  ------  ------ + --   --
PR: 0x    203e      34  0x   2      30  ------  ------  ------ + --   -- [    8190]
PR: 0x    2060     304  0x   0     300  ------  ------  ------ 0x40 0x00 [     300]
PR: 0x    2190    1014  0x   0    1010  ------  ------  ------ 0x00 0x00 [    1010]
PR: 0x    2586    1014  0x   0    1010  ------  ------  ------ 0x00 0x00 [    1010]
PR: 0x    297c    1014  0x   0    1010  ------  ------  ------ 0x00 0x00 [    1010]
PR: 0x    2d72    1014  0x   0    1010  ------  ------  ------ 0x00 0x00 [    1010]
PR: 0x    3168    1014  0x   0    1010  ------  ------  ------ 0x00 0x00 [    1010]
PR: 0x    355e    1014  0x   0    1010  ------  ------  ------ 0x00 0x00 [    1010]
PR: 0x    3954    1014  0x   0    1010  ------  ------  ------ 0x00 0x00 [    1010]
PR: 0x    3d4a    1014  0x   0    1010  ------  ------  ------ 0x00 0x00 [    1010]
PR: 0x    4140     942  0x   0     938  ------  ------  ------ 0x00 0x00 [     938]
PR: 0x    44ee      62  0x   0      58  ------  ------  ------ 0x81 0x00 [      58]
PR: EOF
------------------------------------- EOF --------------------------------------
PR Count: 21
Histogram of Physical Record lengths:
Bytes
   34 [1] | ++++++++++
   62 [2] | ++++++++++++++++++++
  304 [1] | ++++++++++
  942 [1] | ++++++++++
 1014 [8] |␣
↪++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 1024 [8] |␣
↪++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
CPU time =    0.001 (S)
Bye, bye!
```

First `tdlisscanphysrec` echo's the command line then it scans the Physical Records an writes out a table that has the following columns:

Heading	Description
`tell()`	The file position of the start of the Physical Record as a hex integer.
`Length`	The length of the Physical Record as a decimal integer.
`Attr`	The Physical Record Header attributes as a hex integer.
`LD_len`	The length of the logical data payload contained in this Physical Record.
`RecNum`	A record number from the Physical Record trailer if present, otherwise: `------`
`FilNum`	A file number from the Physical Record trailer if present, otherwise: `------`
`ChkSum`	A checksum from the Physical Record trailer if present, otherwise: `------`
`LR`	Logical Record type from the Logical Record Header as a hex integer.
`Attr`	Logical Record attributes from the Logical Record Header as a hex integer. This is (almost?) always 0x00
`[Total LD]`	The total length of the logical data in the Logical Record if a terminator Physical Record, otherwise blank.

This is followed by an ASCII histogram of the lengths of all Physical Records with the following columns:

1. The size in bytes.
2. The frequency count.
3. A series of + that is proportionate to the frequency count.

If TIF markers are detected then the output adds TIF columns thus:

```
TIF     ?  :        Type        Back        Next  PR:     tell()  Length    Attr  LD_
↪len  RecNum  FilNum  ChkSum   LR Attr [Total LD]
------------------------------------------------------------- start ----------------
↪----------------------------------------------
TIF  True >:  0x       0  0x       0  0x      4a  PR: 0x       0      62  0x   0
↪58  ------  ------  ------ 0x80 0x00 [     58]
TIF  True >:  0x       0  0x       0  0x     456  PR: 0x      4a    1024  0x   1
↪1020  ------  ------  ------ 0x22 0x00
TIF  True >:  0x       0  0x      4a  0x     862  PR: 0x     456    1024  0x   3
↪1020  ------  ------  ------ + --   --
TIF  True >:  0x       0  0x     456  0x     c6e  PR: 0x     862    1024  0x   3
↪1020  ------  ------  ------ + --   --
TIF  True >:  0x       0  0x     862  0x    107a  PR: 0x     c6e    1024  0x   3
↪1020  ------  ------  ------ + --   --
TIF  True >:  0x       0  0x     c6e  0x    1486  PR: 0x    107a    1024  0x   3
↪1020  ------  ------  ------ + --   --
TIF  True >:  0x       0  0x    107a  0x    1892  PR: 0x    1486    1024  0x   3
↪1020  ------  ------  ------ + --   --
TIF  True >:  0x       0  0x    1486  0x    1c9e  PR: 0x    1892    1024  0x   3
↪1020  ------  ------  ------ + --   --
TIF  True >:  0x       0  0x    1892  0x    20aa  PR: 0x    1c9e    1024  0x   3
↪1020  ------  ------  ------ + --   --
TIF  True >:  0x       0  0x    1c9e  0x    20ec  PR: 0x    20aa      54  0x   2
↪50  ------  ------  ------ + --   -- [    8210]
```

The additional columns are:

Heading	Description
`?`	?
`Type`	TIF marker type, 0 for in-file record, 1 for EOF.
`Back`	The file position of the precious TIF marker as a hex integer.
`Next`	The file position of the next TIF marker as a hex integer.

Scanning Logical Records in LIS Files with `tdlisscanlogirecord`

Scans a LIS79 file and reports the Logical Record structure.

Arguments

One argument that will be treated as a path to a LIS file.

Options

Option	Description
`--version`	Show program's version number and exit
`-h, --help`	Show this help message and exit.
`-k, --keep-going`	Keep going as far as sensible. [default: False]
`-l LOGLEVEL, --loglevel=LOGLEVEL`	Log Level (debug=10, info=20, warning=30, error=40, critical=50) [default: 20]
`-v, --verbose`	Verbose output, this outputs a representation of table data and DFSRs.

Examples

Example of scanning a LIS file:

```
$ tdlisscanlogirecord RW.lis
Cmd: ScanLogiRec.py RW.lis
0x00000000 <TotalDepth.LIS.core.LogiRec.LrFileHeadRead object at 0x1007981d0>: "File␣
↪header"
2012-02-08 17:43:45,078 WARNING  LrTableRead(): Discarding duplicate row b'BS7 ' in␣
↪table b'CONS'
2012-02-08 17:43:45,087 WARNING  LrTableRead.__init__(): Tell: 0x4a LD index: 0x32␣
↪Error: FileRead.unpack(): Bytes: b'\x00' not enough for struct that needs: 12 bytes.
0x0000004a <TotalDepth.LIS.core.LogiRec.LrTableRead object at 0x100798210>: "Well␣
↪site data"
0x000020ec <TotalDepth.LIS.core.LogiRec.LrDFSRRead object at 0x1007981d0>: "Data␣
↪format specification record"
0x0006141c <TotalDepth.LIS.core.LogiRec.LrFileTailRead object at 0x10058e7d0>: "File␣
↪trailer"
0x00061466 <TotalDepth.LIS.core.LogiRec.LrFileHeadRead object at 0x10058e850>: "File␣
↪header"
2012-02-08 17:43:45,103 WARNING  LrTableRead(): Discarding duplicate row b'BS7 ' in␣
↪table b'CONS'
0x000614b0 <TotalDepth.LIS.core.LogiRec.LrTableRead object at 0x10058e7d0>: "Well␣
↪site data"
0x0006353e <TotalDepth.LIS.core.LogiRec.LrDFSRRead object at 0x10058e850>: "Data␣
↪format specification record"
0x00065a44 <TotalDepth.LIS.core.LogiRec.LrFileTailRead object at 0x10058e850>: "File␣
↪trailer"
0x00065a8e <TotalDepth.LIS.core.LogiRec.LrFileHeadRead object at 0x10058e7d0>: "File␣
↪header"
2012-02-08 17:43:45,116 WARNING  LrTableRead(): Discarding duplicate row b'BS7 ' in␣
↪table b'CONS'
2012-02-08 17:43:45,124 WARNING  LrTableRead.__init__(): Tell: 0x65ad8 LD index: 0x32␣
↪Error: FileRead.unpack(): Bytes: b'\x00' not enough for struct that needs: 12 bytes.
```

(continues on next page)

(continued from previous page)

```
0x00065ad8 <TotalDepth.LIS.core.LogiRec.LrTableRead object at 0x10058e850>: "Well␣
↪site data"
0x00067b7a <TotalDepth.LIS.core.LogiRec.LrDFSRRead object at 0x10058e7d0>: "Data␣
↪format specification record"
0x000d2c44 <TotalDepth.LIS.core.LogiRec.LrFileTailRead object at 0x10058e7d0>: "File␣
↪trailer"
CPU time =    0.064 (S)
Bye, bye!
```

Scanning Logical Data in LIS Files with `tdlisscanlogidata`

Scans a LIS79 file and reports the Logical Record structure.

Arguments

One argument that will be treated as a path to a LIS file.

Options

Option	Description
`--version`	Show program's version number and exit
`-h, --help`	Show this help message and exit.
`-k, --keep-going`	Keep going as far as sensible. [default: False]
`-d DUMP, --dump=DUMP`	Dump complete data at these integer positions (ws separated, hex/dec). [default:]
`-l LOGLEVEL, --loglevel=LOGLEVEL`	Log Level (debug=10, info=20, warning=30, error=40, critical=50) [default: 20]
`-v, --verbose`	Verbose output, this outputs a representation of table data and DFSRs.

Examples

Example of scanning a LIS file:

```
$ ``tdlisscanlogidata`` LIS.lis
Cmd: ScanLogiData.py LIS.lis
Offset        Length  Type  Logical Data
0x00000000        58   128  b'\x80\x00RUN1R .S01\x00\x00DAT2TF            '...
0x0000003E      8190    34  b'"\x00IA\x04\x00TYPE    CONS\x00A\x04\x00MNEM    HI'...
0x00002060       300    64  b'@\x00\x01\x02O\x00\x00\x02\x02O\x00\x00\x03\x04I\x00\
↪x00\x00\x18\x04\x02O\x00\x01\x08\x04D?N\x07_\t'...
0x00002190      1010     0  b"\x00\x00F@'\xde\xbe76\xfb@\xd6\x1a\xc0@\xd0\xdc\xc7D\
↪xe0\xa6P\xba\x83\x18\x00F@&\xa6\xbe-"...
0x00002586      1010     0  b'\x00\x00E\xff\xe9S\xbe:\x1f\x82@\xfe%\xc9@\xf7\xd5\
↪xb7EA\x90\xda\xba\x83\x18\x00E\xff\xe6\xe3\xbe\x8a'...
0x0000297C      1010     0  b'\x00\x00E\xff\x82\xea\xbe-\xe1\xa8@\xd83\xb6@\xf0\x0f\
↪x0fET\x149D\xc8\x08\xc5E\xff\x80y\xbe-'...
0x00002D72      1010     0  b'\x00\x00E\xff\x1c\x80\xbd\xba\x7f\x19@\xc4\xbf\xe8@y\
↪x0b\xb3E\xc0\x08\x03D\xd5\xednE\xff\x1a\x10\xbd\xb4'...
0x00003168      1010     0  b'\x00\x00E\xfe\xb6\x16\xbe\x12\xde\xf0@\xcb1\xe7@zF\
↪xc2Ew\xba/D\xd0\xca\xd9E\xfe\xb3\xa6\xbe\x17'...
```

(continues on next page)

(continued from previous page)

```
0x0000355E      1010     0  b'\x00\x00E\xfeO\xac\xbe40\x85@\xcc4\x8d@of\xd9E\xc1F\
↪xd8D\xd4\xaa+E\xfeM<\xbe6'...
0x00003954      1010     0  b"\x00\x00E\xfd\xe9C\xbd\xb2\x19\xf0\xba\x83\x18\x00AK'%D\
↪xed\x9d\xdbD\xd0\x17RE\xfd\xe6\xd3\xbd\xab"...
0x00003D4A      1010     0  b'\x00\x00E\xfd\x82\xd9\xba\x83\x18\x00\xba\x83\x18\x00\
↪xba\x83\x18\x00\xba\x83\x18\x00D\xd0\xad\xf3E\xfd\x80i\xba\x83'...
0x00004140       938     0  b'\x00\x00E\xfd\x1co\xba\x83\x18\x00\xba\x83\x18\x00\xba\
↪x83\x18\x00\xba\x83\x18\x00D\xd8\x8c\xb5E\xfd\x19\xff\xba\x83'...
0x000044EE        58     0  b'\x81\x00RUN1R .S01\x00\x00DAT2TF            '...
Histogram of Logical Data lengths:
Bytes
   58 [1] | ++++++++++
  300 [1] | ++++++++++
  938 [1] | ++++++++++
 1010 [8] |␣
↪++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 8190 [1] | ++++++++++
Histogram of Logical Record types:
  0 [9] |␣
↪++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 34 [1] | ++++++++++
 64 [1] | ++++++++++
128 [1] | ++++++++++
CPU time =    0.001 (S)
Bye, bye!
```

First `tdlisscanlogidata` echo's the command line then it scans the file an writes out a table that has the following columns:

Heading	Description
`Offset`	The file position of the start of the Physical Record as a hex integer.
`Length`	The length of the Logical Record as a decimal integer.
`Type`	The Logical Record type as a decimal integer.
`Logical Data`	The logical data payload. Only the first 32 bytes are shown. `...` is shown if the payload is longer than 32 bytes. If the verbose or dump options are given then all bytes are shown.

This is followed by an ASCII histogram of the lengths of all logical data with the following columns:

1. The size in bytes.
2. The frequency count.
3. A series of + that is proportionate to the frequency count.

This is followed by an ASCII histogram of the lengths of all Logical Record types with the following columns:

1. The size in bytes.
2. The frequency count.
3. A series of + that is proportionate to the frequency count.

Using the -d option expands the output when the file position value matches. So given the above then adding `-d 0x44EE` changes this:

```
...
0x000044EE        58     0  b'\x81\x00RUN1R .S01\x00\x00DAT2TF            '...
...
```

To this:

```
...
0x000044EE        58     0  b'\x81\x00RUN1R .S01\x00\x00DAT2TF                \x00␣
↪1024\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
...
```

Extracting Data from LIS with `tdlisdumpframeset`

Reads a LIS file and writes out tab separated values of each frame.

Arguments

1. The path to the LIS file.

Options

Option	Description
`--version`	Show program's version number and exit
`-h, --help`	Show this help message and exit.
`-k, --keep-going`	Keep going as far as sensible. [default: False]
`-l LOGLEVEL, --loglevel=LOGLEVEL`	Log Level (debug=10, info=20, warning=30, error=40, critical=50) [default: 20]
`-s, --summary`	Display summary only [default: False].

Examples

```
$ ``tdlisdumpframeset`` LIS.lis
Cmd: DumpFrameSet.py LIS.lis
2012-02-09 08:41:38,372 INFO     Index.indexFile(): LIS.lis
<TotalDepth.LIS.core.LogPass.LogPass object at 0x101a0c510>
b'DEPT' [b'M   ']   b'SP  ' [b'MV  ']   b'SN  ' [b'OHMM']   b'ILD ' [b'OHMM']   b'CILD
↪' [b'MMHO']   b'DT  ' [b'US/M']
2052.98 -4.54908    1.34538 1.26348 386.599 -999.25
2052.83 -5.1372     1.36062 1.29521 500.511 -999.25
2052.68 -6.66747    1.38543 1.45786 595.623 -999.25
2052.53 -6.69616    1.43226 1.61085 592.447 -999.25
2052.37 -4.93782    1.51647 1.6622  590.846 -999.25
2052.22 -4.38823    1.66883 1.70584 586.092 -999.25
2052.07 -4.70347    1.8102  1.70607 577.873 -999.25
...
1996.44 -999.25     -999.25 -999.25 -999.25 -999.25
1996.29 -999.25     -999.25 -999.25 -999.25 -999.25
1996.14 -999.25     -999.25 -999.25 -999.25 -999.25
1995.99 -999.25     -999.25 -999.25 -999.25 -999.25

Sc Name          Count      Min     Mean      Max Std Dev.       --       ==       ++␣
↪    Bias    Drift Activity
DEPT [M   ]        375    2e+03 2.02e+03 2.05e+03     16.5      374        0        0␣
↪       1   -0.152 0.000144
```

(continues on next page)

(continued from previous page)

```
SP    [MV  ]         262     -13.7     -5.67    -0.769      2.66       124          0       137␣
↪ -0.0498    0.0144     0.678
SN    [OHMM]         252     0.866      1.36      1.98     0.277       123          0       128␣
↪ -0.0199 -0.000719    0.0425
ILD   [OHMM]         253     0.361      1.31      2.35     0.412        95          0       157␣
↪  -0.246   0.00429     0.134
CILD [MMHO]         253       387       787 1.75e+03       236       130          0       122␣
↪  0.0317     0.205     0.101
DT    [US/M]         292       133       320       460      42.5       139          0       152␣
↪ -0.0447    -0.451     0.106
CPU time =     0.047 (S)
Bye, bye!
```

The summary table at the end has the following columns:

Heading	Description
`Sc Name`	The sub-channel name and units of measure.
`Count`	The number of non-null values.
`Min`	Minimum value.
`Mean`	Arithmetic mean of values.
`Max`	Maximum value.
`Std Dev.`	Standard deviation of values.
`--`	Number of values that are a decrease over the previous value.
`==`	Number of values that are equal to the previous value.
`++`	Number of values that are an increase over the previous value.
`Bias`	(`--` - `++`) / total
`Drift`	(last value - first value) / number of values
`Activity`	The RMS exponent change.

Analysing Table Data in LIS Files with `tdlistablehistogram`

Provides a count of elements in LIS tables.

Arguments

1. A path to a LIS file or directory of LIS files.

Options

Option	Description
`--version`	Show program's version number and exit
`-h, --help`	Show this help message and exit.
`-k, --keep-going`	Keep going as far as sensible. [default: False]
`-r, --recursive`	Process input recursively. [default: False]
`-s, --structure`	Display table structure (row/col range). [default: False]
`--type=LRTYPE`	Logical record table type e.g. 34. [default: 34]
`--name=NAME`	Logical record table name e.g. PRES. [default:]
`--row=ROW`	Logical record table row e.g. "GR ". [default:]
`--col=COL`	Logical record table column e.g. "LEDG". [default:]
`-l LOGLEVEL, --loglevel=LOGLEVEL`	Log Level (debug=10, info=20, warning=30, error=40, critical=50) [default: 20]

Examples

Count of all entries regardless of the table/row/column that they appear in:

```
$ ``tdlistablehistogram`` -l 40 Simple/
Cmd: TableHistogram.py -l 40 Simple/
======================== Count of all table entries ========================
{"(34, b'    ')": 1414,
 "(34, b'0.445')": 5,
 "(34, b'0.621')": 5,
 "(34, b'013529700231')": 7,
 "(34, b'1')": 5,
 "(34, b'1.22')": 5,
 "(34, b'1.70')": 2,
 "(34, b'116')": 2,
 "(34, b'12.25')": 2,
 "(34, b'15')": 5,
 "(34, b'15-4-76')": 5,
 "(34, b'17')": 5,
 "(34, b'17.5')": 5,
 "(34, b'19')": 5,
 "(34, b'1976')": 7,
 "(34, b'2')": 2,
 "(34, b'2055.0')": 2,
 "(34, b'2071.2')": 4,
 "(34, b'25')": 2,
 "(34, b'25-6-76')": 2,
 "(34, b'257.0')": 7,
...
 "(34, b'WN  ')": 7,
 "(34, b'YEAR')": 7,
 '(34,)': 443}
====================== Count of all table entries END ======================
CPU time =    0.205 (S)
Bye, bye!
```

The result is a dictionary that has the key as a pair `(lr_type, cell_value)` and the value as a count of the number of occurrences.

If the `-s` option is used then an additional summary is provided:

```
============================== Row entries ==============================
{(34, b'CONS', b'APIN'): 7,
 (34, b'CONS', b'BLI '): 7,
 (34, b'CONS', b'BS1 '): 7,
 (34, b'CONS', b'BS2 '): 7,
 (34, b'CONS', b'BS3 '): 7,
...
 (34, b'CONS', b'WN  '): 7,
 (34, b'CONS', b'YEAR'): 7}
============================== Row entries END ==========================
============================ Column entries ============================
{(34, b'CONS', b'ALLO'): 707,
 (34, b'CONS', b'MNEM'): 707,
 (34, b'CONS', b'PUNI'): 707,
 (34, b'CONS', b'TUNI'): 707,
 (34, b'CONS', b'VALU'): 707}
============================ Column entries END ========================
```

This are dictionaries that have the key as a tripple `(lr_type, table_name, row_name)` and `(lr_type, table_name, column_name)` respectively and the value as a count of the number of occurrences.

Filtering by Logical Record type, table name, row name and column name (note quoting of spaces):

```
$ ``tdlistablehistogram`` -l 40 --type=34 --name=CONS --row="WN  " --col=VALU Simple/
Cmd: TableHistogram.py -l 40 --type=34 --name=CONS --row=WN   --col=VALU Simple/
======================== Count of all table entries ========================
{"(34, b'CONS', b'WN  ', b'VALU', b'B897 - 14')": 1,
 "(34, b'CONS', b'WN  ', b'VALU', b'DIEKSAND 111A')": 3,
 "(34, b'CONS', b'WN  ', b'VALU', b'VOELKERSEN AZ4')": 3}
====================== Count of all table entries END ======================
CPU time =    0.174 (S)
Bye, bye!
```

The result is a dictionary that has the key as a quadruple `(lr_type, table_name, row_name, column_name, cell_value)` and the value as a count of the number of occurrences.

Indexing LIS Files with `tdlisindex`

This indexes a LIS file and prints out the result. It can also provide some performance measurements of the indexing operation. See *Indexing LIS Files* for more information about the design and performance of LIS indexing.

Arguments

1. The path to a LIS file or a directory of LIS files.

Options

Option	Description
`--version`	Show program's version number and exit
`-h, --help`	Show this help message and exit.
`-k, --keep-going`	Keep going as far as sensible. [default: False]
`-l LOGLEVEL, --loglevel=LOGLEVEL`	Log Level (debug=10, info=20, warning=30, error=40, critical=50) [default: 20]
`-j JOBS, --jobs=JOBS`	Max processes when multiprocessing. Zero uses number of native CPUs [8]. -1 disables multiprocessing. [default: -1]
`-t TIMES, --times=TIMES`	Number of times to repeat the read [default: 1]
`-s, --statistics`	Dump timing statistics. [default: False]
`-v, --verbose`	Verbose output, this outputs a representation of table data and DFSRs.
`-r, --recursive`	Process input recursively. [default: False]

Examples

Simple tif_scan_path of a single file:

```
$ ``tdlisindex`` Simple/LIS.lis
Cmd: Index.py Simple/LIS.lis
2012-02-09 09:36:28,039 INFO     Index.indexFile(): Simple/LIS.lis
File size: 17708 (0.017 MB) Reference Time: 0.002459 (s) for Simple/LIS.lis␣
↪pickleLen=4351 jsonLen=-1
Summary:
Results:        1
 Errors:        0
  Total:        1
CPU time =    0.004 (S)
Bye, bye!
```

Simple tif_scan_path of a single file with verbose output:

```
$ ``tdlisindex`` -v Simple/LIS.lis
Cmd: Index.py -v Simple/LIS.lis
2012-02-09 09:39:29,493 INFO     Index.indexFile(): Simple/LIS.lis
<TotalDepth.LIS.core.FileIndexer.FileIndex object at 0x10197fdd0> "Simple/LIS.lis"␣
↪[4]:
  tell: 0x00000000 type=128 <TotalDepth.LIS.core.FileIndexer.IndexFileHead object at␣
↪0x10197fe10>
  tell: 0x0000003e type= 34 name=b'CONS' <TotalDepth.LIS.core.FileIndexer.IndexTable␣
↪object at 0x10197fe90>
  <TotalDepth.LIS.core.LogPass.LogPass object at 0x101b071d0>
  tell: 0x000044ee type=129 <TotalDepth.LIS.core.FileIndexer.IndexFileTail object at␣
↪0x101b07790>
============================== All records ==============================
tell: 0x00000000 type=128 <TotalDepth.LIS.core.FileIndexer.IndexFileHead object at␣
↪0x10197fe10>
```

(continues on next page)

(continued from previous page)

```
tell: 0x0000003e type= 34 name=b'CONS' <TotalDepth.LIS.core.FileIndexer.IndexTable␣
↪object at 0x10197fe90>
<TotalDepth.LIS.core.LogPass.LogPass object at 0x101b071d0>
tell: 0x000044ee type=129 <TotalDepth.LIS.core.FileIndexer.IndexFileTail object at␣
↪0x101b07790>
============================ All records DONE ============================
============================== Log Passes ==============================
LogPass <TotalDepth.LIS.core.LogPass.LogPass object at 0x101b071d0>:
       DFSR: <TotalDepth.LIS.core.LogiRec.LrDFSRRead object at 0x10197ff90>: "Data␣
↪format specification record"
 Frame plan: <TotalDepth.LIS.core.Type01Plan.FrameSetPlan object at 0x101b07210>:␣
↪indr=0 frame length=24 channels=6
   Channels: [b'DEPT', b'SP  ', b'SN  ', b'ILD ', b'CILD', b'DT  ']
        RLE: <TotalDepth.LIS.core.Rle.RLEType01 object at 0x101b07250>: func=None:␣
↪[RLEItemType01: datum=8592 stride=1014 repeat=7 frames=42, RLEItemType01:␣
↪datum=16704 stride=None repeat=0 frames=39]
     X axis: first=2052.983 last=1995.986 frames=375 overall spacing=-0.1524 in␣
↪optical units=b'M   ' (actual units=b'M   ')
  Frame set: None

============================ Log Passes DONE ============================
============================== Plot Records ==============================
============================ Plot Records DONE ============================
   Min: 0.003 (s)
   Max: 0.003 (s)
  Mean: 0.003 (s)
File size: 17708 (0.017 MB) Reference Time: 0.002529 (s) for Simple/LIS.lis␣
↪pickleLen=4351 jsonLen=-1
Summary:
Results:        1
 Errors:        0
  Total:        1
CPU time =    0.004 (S)
Bye, bye!
```

Scan of a directory (recursively) indexing each file 11 times and writing out statistics:

```
$ ``tdlisindex`` -t11 -s -l 40 Simple/
Cmd: Index.py -t11 -s -l 40 ../../../../TDTestData/LIS/Simple
File size: 17708 (0.017 MB) Reference Time: 0.001670 (s) for Simple/LIS.lis␣
↪pickleLen=4351 jsonLen=-1
File size: 863374 (0.823 MB) Reference Time: 0.043411 (s) for Simple/RW.lis␣
↪pickleLen=18231 jsonLen=-1
File size: 853030 (0.814 MB) Reference Time: 0.039238 (s) for Simple/RW_No_TIF.lis␣
↪pickleLen=18238 jsonLen=-1
Summary:
Size (kb)   Time (s)
17.293      0.001670
843.139     0.043411
833.037     0.039238

Files: 3
Errors: 0
CPU time =    0.938 (S)
Bye, bye!
```

tdXlisrandomframesetread

For developers only. This may not be present in some distributions. This is designed to measure the performance of loading and iterating across a frame-set.

Converting LIS Files to LAS Files with tdlistolas

This takes a LIS file or directory of them and writes out a set of LAS files. A single LAS file is written for each Log Pass so a single LIS file produces one or more LAS files.

The frames in the log pass can be sub-sampled by using `--frame-slice` which speeds things up when processing large files. The `--channels` option can be used to limit channels.

Where a channel has multiple values, and LAS can only record a single value, then the `--array-reduction` flag can be used to specify how the single value is computed. The allowable values are `{first,max,mean,median,min}` and the default is `mean`.

LAS File Naming Convention

One LIS file produces one or more LAS files. LAS file names are of the form:

```
{LIS_File_no_extension}_{logical_file_number}.las
```

Processing a Single LIS File

Given the path out the LAS files will be named `{path_out}_{logical_file_number}.las`

For example `tdlistolas foo.lis bar/baz` might create:

```
bar/baz_0.las
bar/baz_1.las
```

and so on.

Processing a Directory of LIS Files

Given the path out the LAS files will be named:

```
{path_out}/{LIS_File}_{logical_file_number}.las
```

For example `tdlistolas foo/ bar/baz` might create:

```
bar/baz/bit_0.las
bar/baz/bit_1.las
```

and so on.

The output directory structure will mirror the input directory structure.

Arguments

The first argument is the path to a LIS file or directory. The second argument is the path to write the output to.

Options

-h, --help show this help message and exit

--version show program's version number and exit

-k, --keep-going Keep going as far as sensible. Default: False.

-v, --verbose Increase verbosity, additive [default: 0]

-r, --recurse Process the input recursively. Default: False.

-l LOG_LEVEL, --log-level LOG_LEVEL Log Level as an integer or symbol. (0<->NOTSET, 10<->DEBUG, 20<->INFO, 30<->WARNING, 40<->ERROR, 50<->CRITICAL) [default: 20]

-j JOBS, --jobs JOBS Max processes when multiprocessing.Zero uses number of native CPUs [8]. Negative value disables multiprocessing code. Default: -1.

--frame-slice FRAME_SLICE Do not process all frames but sample or slice the frames. SAMPLE: Sample is of the form "N" so a maximum of N frames, roughly regularly spaced, will be processed. N must be +ve, non-zero integer. Example: "64" - process a maximum of 64 frames. SLICE: Slice the frames is of the form start,stop,step as a comma separated list. Values can be absent or "None". Examples: ",," - every frame, ",,2" - every other frame, ",10," - frames 0 to 9, "4,10,2" - frames 4, 6, 8, "40,-1,4" - every fourth frame from 40 to the end. Results will be truncated by frame array length. Use '?' to see what frames are available [default: ",," i.e. all frames]

--log-process LOG_PROCESS Writes process data such as memory usage as a log INFO line every LOG_PROCESS seconds. If 0.0 no process data is logged. [default: 0.0]

--gnuplot GNUPLOT Directory to write the gnuplot data.

--array-reduction ARRAY_REDUCTION Method to reduce multidimensional channel data to a single value. One of {first,max,mean,median,min} [default: first]

--channels CHANNELS Comma separated list of channels to write out (X axis is always included). Use '?' to see what channels exist without writing anything. [default: ""]

--field-width FIELD_WIDTH Field width for array data [default: 16].

--float-format FLOAT_FORMAT Floating point format for array data [default: ".3f"].

Examples

Finding out what Channels and Frames Exist:

Use `--channels=?` and/or `--frame-slice=?` to see what channels and frames exist in the LIS file.

```
$ tdlistolas --channels=? --frame-slice=? example_data/LIS/data/DILLSON-1_WELL_LOGS_
↪FILE-049.LIS tmp/scrap/
======= File example_data/LIS/data/DILLSON-1_WELL_LOGS_FILE-049.LIS =======
Log pass 1:
Available channels: ['FC0 ', 'FC1 ', 'FC2 ', 'FC3 ', 'FC4 ', 'STAT', 'REF ', 'REFC',
↪'EMEX', 'PADP', 'TEMP', 'FEP1', 'FEP2', 'RAC1', 'RAC2', 'P1AZ', 'DEVI', 'HAZI', 'C1␣
↪ ', 'C2  ', 'FEP ', 'RB  ']
X axis: first=5280.792 last=5079.725 frames=755 overall spacing=-0.2667 in optical␣
↪units=b'FEET' (actual units=b'.1IN')
===== END File example_data/LIS/data/DILLSON-1_WELL_LOGS_FILE-049.LIS =====
```

Processing a Single File

```
$ tdlistolas example_data/LIS/data/DILLSON-1_WELL_LOGS_FILE-049.LIS tmp/scrap/
Cmd: /Users/engun/venvs/TotalDepth36_00/bin/tdlistolas example_data/LIS/data/DILLSON-
↪1_WELL_LOGS_FILE-049.LIS tmp/scrap/
gnuplot version: "b'gnuplot 5.2 patchlevel 6'"
2020-09-01 13:31:24,159 - WriteLAS.py      -  117 - 39415 - (MainThread) - INFO     -␣
↪process_to_las(): Namespace(array_reduction='first', channels='', field_width=16,␣
↪float_format='.3f', frame_slice=',,', gnuplot=None, jobs=-1, keepGoing=False, log_
↪level=20, log_process=0.0, path_in='example_data/LIS/data/DILLSON-1_WELL_LOGS_FILE-
↪049.LIS', path_out='tmp/scrap/', recurse=False, verbose=0)
2020-09-01 13:31:24,159 - WriteLAS.py      -   93 - 39415 - (MainThread) - INFO     -␣
↪index_dir_or_file(): "example_data/LIS/data/DILLSON-1_WELL_LOGS_FILE-049.LIS" to
↪"tmp/scrap/" recurse: False
2020-09-01 13:31:24,159 - ToLAS.py         -  328 - 39415 - (MainThread) - INFO     -␣
↪single_lis_file_to_las(): path_in: example_data/LIS/data/DILLSON-1_WELL_LOGS_FILE-
↪049.LIS path_out: tmp/scrap/
2020-09-01 13:31:24,161 - File.py          -  254 - 39415 - (MainThread) - INFO     -␣
↪Finding best PR settings for: <_io.BufferedReader name='example_data/LIS/data/
↪DILLSON-1_WELL_LOGS_FILE-049.LIS'>
2020-09-01 13:31:24,171 - File.py          -  260 - 39415 - (MainThread) - INFO     -␣
↪Best pad options, first of 5: <PhysicalRecordSettings(pad_modulo=0, pad_non_null:␣
↪False> giving 100 Physical Records.
2020-09-01 13:31:24,177 - ToLAS.py         -  337 - 39415 - (MainThread) - INFO     -␣
↪Reading LIS in example_data/LIS/data/DILLSON-1_WELL_LOGS_FILE-049.LIS
2020-09-01 13:31:24,177 - ToLAS.py         -  338 - 39415 - (MainThread) - INFO     -␣
↪Index.indexFile(): example_data/LIS/data/DILLSON-1_WELL_LOGS_FILE-049.LIS
2020-09-01 13:31:24,189 - ToLAS.py         -  358 - 39415 - (MainThread) - INFO     -␣
↪LIS Logical Files: [<TotalDepth.LIS.ToLAS.LisLogicalFile object at 0x111a2e390>]
2020-09-01 13:31:24,190 - ToLAS.py         -  291 - 39415 - (MainThread) - INFO     -␣
↪write_las_file(): path_in: example_data/LIS/data/DILLSON-1_WELL_LOGS_FILE-049.LIS␣
↪path_out: tmp/scrap/
2020-09-01 13:31:24,190 - ToLAS.py         -  294 - 39415 - (MainThread) - INFO     -␣
↪Writing to LAS tmp/scrap/_0.las
 Input  Output LAS Count  Time  Ratio  ms/Mb Exception                               ␣
↪                    Path
------ ------- --------- ----- ------ ------ --------- --------------------------------
↪-------------------------
```

(continues on next page)

(continued from previous page)

```
98,508 285,487         1 0.201 289.8% 2139.3     False "example_data/LIS/data/DILLSON-
↪1_WELL_LOGS_FILE-049.LIS"
 Total files: 1
Failed files: 0
Execution time =    0.220 (S)
Out of  1 processed 1 files of total size 98,508 input bytes
Wrote 285,487 output bytes, ratio: 289.811% at 2344.7 ms/Mb
Execution time: 0.220 (s)
Bye, bye!
```

The LAS files look like this:

```
$ head -n20  tmp/scrap/_0.las
~Version Information Section
VERS.          2.0                                : CWLS Log ASCII Standard - VERSION␣
↪2.0
WRAP.          NO                                 : One Line per depth step
PROD.          TotalDepth                         : LAS Producer
PROG.          TotalDepth.LIS.ToLAS 0.1.1         : LAS Program name and version
CREA.          2020-09-01 12:31:24.190938 UTC     : LAS Creation date [YYYY-mm-dd␣
↪HH:MM:SS.us UTC]
SOURCE.        DILLSON-1_WELL_LOGS_FILE-049.LIS   : LIS File Name
LOGICAL-FILE.  0                                  : Logical File number in the LIS file
~Well Information Section
#MNEM.UNIT  Value           Description
#---------  -----           -----------
STRT.FEET   5280.792        : START
STOP.FEET   5079.725        : STOP
STEP.FEET   -0.267          : STEP
NULL.       -999.250        : NULL VALUE
COUN.       N.A.            : County
CTRY.       AUSTRALIA       : COUNTRY
LATI.       21 23 06.314    : Latitude
LONG.       115 10 56.336   : Longitude
STAT.       WEST AUSTRALIA  : STATE
```

Processing a Directory

Use the `-r` option to process recursively. The output directory will mirror the input directory.

```
$ tdlistolas -r example_data/LIS/data tmp/LAS
  Input  Output LAS Count  Time  Ratio  ms/Mb Exception                                 ␣
↪                      Path
------- ------- --------- ----- ------ ------ --------- ------------------------------
↪--------------------------
 96,376 341,102         1 0.151 353.9% 1645.0     False "example_data/LIS/data/
↪DILLSON-1_WELL_LOGS_FILE-013.LIS"
184,084 741,708         1 0.300 402.9% 1709.6     False "example_data/LIS/data/
↪DILLSON-1_WELL_LOGS_FILE-037.LIS"
 98,508 285,487         1 0.154 289.8% 1639.0     False "example_data/LIS/data/
↪DILLSON-1_WELL_LOGS_FILE-049.LIS"
 Total files: 3
Failed files: 0
```

1.5.5 RP66V1 Command Line Tools

This describes the command line tools that are available for processing RP66V1 files. They are:

Tool Name	Description
`tdrp66v1scanhtml`	Scans RP66V1 file(s) and writes out a summary in HTML.
`tdrp66v1tolas`	Converts RP66V1 file(s) to a set of LAS files.
`tdrp66v1indexpickle`	Indexes RP66V1 file(s) and writes the indexes for future use as Python pickle files.
`tdrp66v1indexxml`	Indexes RP66V1 file(s) and writes the indexes as XML files.
`tdrp66v1scan`	Scans RP66V1 file at various levels of structure.

Creating HTML Pages from RP66V1 Files with `tdrp66v1scanhtml`

This takes a RP66V1 file or directory of them and writes out an HTML summary of each Logical File. The summary includes each non-encrypted EFLR and Log Pass. The frames in the log pass can be sub-sampled by using `--frame-slice` which speeds things up when processing large files.

Arguments

The first argument is the path to a RP66V1 file or directory. The second argument is the path to write the output to.

Options

-h, --help show this help message and exit

--version show program's version number and exit

-k, --keep-going Keep going as far as sensible. Default: False.

-v, --verbose Increase verbosity, additive [default: 0]

-r, --recurse Process the input recursively. Default: False.

-l LOG_LEVEL, --log-level LOG_LEVEL Log Level as an integer or symbol. (0<->NOTSET, 10<->DEBUG, 20<->INFO, 30<->WARNING, 40<->ERROR, 50<->CRITICAL) [default: 20]

-j JOBS, --jobs JOBS Max processes when multiprocessing.Zero uses number of native CPUs [8]. Negative value disables multiprocessing code. Default: -1.

-e, --encrypted Output encrypted Logical Records as well. [default: False]

--frame-slice FRAME_SLICE Do not process all frames but sample or slice the frames. SAMPLE: Sample is of the form "N" so a maximum of N frames, roughly regularly spaced, will be processed. N must be +ve, non-zero integer. Example: "64" - process a maximum of 64 frames. SLICE: Slice the frames is of the form start,stop,step as a comma separated list. Values can be absent or "None". Examples: ",," - every frame, ",,2" - every other frame, ",10," - frames 0 to 9, "4,10,2" - frames 4, 6, 8, "40,-1,4" - every fourth frame from 40 to the end. Results will be truncated by frame array length. [default: ",," i.e. all frames]

--log-process LOG_PROCESS Writes process data such as memory usage as a log INFO line every LOG_PROCESS seconds. If 0.0 no process data is logged. [default: 0.0]

--gnuplot GNUPLOT Directory to write the gnuplot data.

Here is an example of the HTML summary of a single RP66V1 file .

Converting RP66V1 Files to LAS Files with `tdrp66v1tolas`

This takes a RP66V1 file or directory of them and writes out a set of LAS files. A single LAS file is written for each Log Pass in each Logical Record.

The frames in the log pass can be sub-sampled by using `--frame-slice` which speeds things up when processing large files. The `--channels` option can be used to limit channels.

Where a channel has multiple values, and LAS con only record a single value, then the `--array-reduction` flag can be used to specify how the single value is computed. The allowable values are `{first,max,mean,median,min}` and the default is `mean`.

LAS File Naming Convention

One RP66V1 file produces one or more LAS files. LAS file names are of the form:

```
{RP66V1_File_no_extension}_{logical_file_number}_{frame_array_name}
```

Processing a Single RP66V1 File

Given the path out the LAS files will be named `{path_out}_{logical_file_number}_{frame_array_name}.las`

For example `tdrp66v1tolas foo.dlis bar/baz` might create:

```
bar/baz_0_2000T.las
bar/baz_0_800T.las
bar/baz_1_2000T.las
bar/baz_1_800T.las
```

and so on.

Processing a Directory of RP66V1 Files

Given the path out the LAS files will be named:

```
{path_out}/{RP66V1_File}_{logical_file_number}_{frame_array_name}.las
```

For example `tdrp66v1tolas foo/ bar/baz` might create:

```
bar/baz/bit_0_2000T.las
bar/baz/bit_0_800T.las
bar/baz/bit_1_2000T.las
bar/baz/bit_1_800T.las
```

and so on.

The output directory structure will mirror the input directory structure.

Arguments

The first argument is the path to a RP66V1 file or directory. The second argument is the path to write the output to.

Options

-h, --help show this help message and exit

--version show program's version number and exit

-k, --keep-going Keep going as far as sensible. Default: False.

-v, --verbose Increase verbosity, additive [default: 0]

-r, --recurse Process the input recursively. Default: False.

-l LOG_LEVEL, --log-level LOG_LEVEL Log Level as an integer or symbol. (0<->NOTSET, 10<->DEBUG, 20<->INFO, 30<->WARNING, 40<->ERROR, 50<->CRITICAL) [default: 20]

-j JOBS, --jobs JOBS Max processes when multiprocessing.Zero uses number of native CPUs [8]. Negative value disables multiprocessing code. Default: -1.

--frame-slice FRAME_SLICE Do not process all frames but sample or slice the frames. SAMPLE: Sample is of the form "N" so a maximum of N frames, roughly regularly spaced, will be processed. N must be +ve, non-zero integer. Example: "64" - process a maximum of 64 frames. SLICE: Slice the frames is of the form start,stop,step as a comma separated list. Values can be absent or "None". Examples: ",," - every frame, ",,2" - every other frame, ",10," - frames 0 to 9, "4,10,2" - frames 4, 6, 8, "40,-1,4" - every fourth frame from 40 to the end. Results will be truncated by frame array length. Use '?' to see what frames are available [default: ",," i.e. all frames]

--log-process LOG_PROCESS Writes process data such as memory usage as a log INFO line every LOG_PROCESS seconds. If 0.0 no process data is logged. [default: 0.0]

--gnuplot GNUPLOT Directory to write the gnuplot data.

--array-reduction ARRAY_REDUCTION Method to reduce multidimensional channel data to a single value. One of {first,max,mean,median,min} [default: first]

--channels CHANNELS Comma separated list of channels to write out (X axis is always included). Use '?' to see what channels exist without writing anything. [default: ""]

--field-width FIELD_WIDTH Field width for array data [default: 16].

--float-format FLOAT_FORMAT Floating point format for array data [default: ".3f"].

Examples

Finding out what Channels and Frames Exist:

Use `--channels=?` and/or `--frame-slice=?` to see what channels and frames exist in the RP66V1 file.

```
$ tdrp66v1tolas --channels=? --frame-slice=? example_data/RP66V1/206_05a-_3_DWL_DWL_
↪WIRE_258276498.DLIS example_data/LAS/206_05a-_3_DWL_DWL_WIRE_258276498
Logical file [0000]: <TotalDepth.RP66V1.core.LogicalFile.LogicalFile object at␣
↪0x109fd50f0>
  Frame Array: b'2000T'
  Channels: b'TIME,TDEP,TENS_SL,DEPT_SL'
  X axis: FrameChannel: OBNAME: O: 2 C: 4 I: b'TIME'            Rc:   2 Co:    1 Un: b
↪'ms'        Di: [1] b'1 second River Time'
  Frames: 921 from 16677259.0 to 17597260.0 interval 1000.0010869565217 [b'ms']

  Frame Array: b'800T'
  Channels: b'TIME,TDEP,ETIM,LMVL,UMVL,CFLA,OCD,RCMD,RCPP,CMRT,RCNU,DCFL,DFS,DZER,
↪RHMD,HMRT,RHV,RLSW,MNU,S1CY,S2CY,RSCU,RSTS,UCFL,CARC,CMDV,CMPP,CNU,HMDV,HV,LSWI,
↪SCUR,SSTA,RCMP,RHPP,RRPP,CMPR,HPPR,RPPV,SMSC,CMCU,HMCU,CMLP'
  X axis: FrameChannel: OBNAME: O: 2 C: 5 I: b'TIME'            Rc:   2 Co:    1 Un: b
↪'ms'        Di: [1] b'400 milli-second time channel'
  Frames: 2301 from 16677259.0 to 17597260.0 interval 400.0004347826087 [b'ms']
```

Processing a Single File

```
$ tdrp66v1tolas example_data/RP66V1/206_05a-_3_DWL_DWL_WIRE_258276498.DLIS example_
↪data/LAS/206_05a-_3_DWL_DWL_WIRE_258276498
  Input    Output LAS Count  Time  Ratio  ms/Mb Exception                          ␣
↪                          Path
------- --------- --------- ----- ------ ------ --------- ----------------------------
↪---------------------------------
540,372 1,812,131         2 1.816 335.3% 3524.1     False "example_data/RP66V1/206_
↪05a-_3_DWL_DWL_WIRE_258276498.DLIS"
Execution time =    1.819 (S)
Out of  1 processed 1 files of total size 540,372 input bytes
Wrote 1,812,131 output bytes, ratio: 335.349% at 3529.3 ms/Mb
$ ll example_data/LAS/
total 4600
-rw-r--r--  1 xxxxxxxx  staff    94317 28 Oct 10:19 206_05a-_3_DWL_DWL_WIRE_258276498_
↪0_2000T.las
-rw-r--r--  1 xxxxxxxx  staff  1717814 28 Oct 10:20 206_05a-_3_DWL_DWL_WIRE_258276498_
↪0_800T.las
```

The LAS files look like this:

```
$ head -n20 example_data/LAS/206_05a-_3_DWL_DWL_WIRE_258276498_0_2000T.las
~Version Information Section
VERS.          2.0                                   : CWLS Log ASCII Standard -␣
↪VERSION 2.0
WRAP.          NO                                    : One Line per depth step
PROD.          TotalDepth                            : LAS Producer
PROG.          TotalDepth.RP66V1.ToLAS 0.1.1         : LAS Program name and version
CREA.          2019-10-28 10:30                      : LAS Creation date [YYYY-mm-
↪dd HH:MM]
```

(continues on next page)

(continued from previous page)

```
DLIS_CREA.     2011-08-20 22:48                          : DLIS Creation date and time␣
↪[YYYY-mm-dd HH:MM]
SOURCE.        206_05a-_3_DWL_DWL_WIRE_258276498.DLIS    : DLIS File Name
FILE-ID.       MSCT_197LTP                               : File Identification Number
LOGICAL-FILE.  0                                         : Logical File number in the␣
↪DLIS file
FRAME-ARRAY.   2000T                                     : Identity of the Frame Array␣
↪in the Logical File
~Well Information Section
#MNEM.UNIT  DATA                          DESCRIPTION
#----.----  ----                          -----------
STRT.ms     16677259.0                    : Start X
STOP.ms     17597260.0                    : Stop X, frames: 921
STEP.ms     1000.0010869565217            : Step (average)
NULL.                                     :
COMP.       Faroe Petroleum               :
WELL.       206/05a-3                     :

$ head -n20 example_data/LAS/206_05a-_3_DWL_DWL_WIRE_258276498_0_800T.las
~Version Information Section
VERS.          2.0                                       : CWLS Log ASCII Standard -␣
↪VERSION 2.0
WRAP.          NO                                        : One Line per depth step
PROD.          TotalDepth                                : LAS Producer
PROG.          TotalDepth.RP66V1.ToLAS 0.1.1             : LAS Program name and version
CREA.          2019-10-28 10:30                          : LAS Creation date [YYYY-mm-
↪dd HH:MM]
DLIS_CREA.     2011-08-20 22:48                          : DLIS Creation date and time␣
↪[YYYY-mm-dd HH:MM]
SOURCE.        206_05a-_3_DWL_DWL_WIRE_258276498.DLIS    : DLIS File Name
FILE-ID.       MSCT_197LTP                               : File Identification Number
LOGICAL-FILE.  0                                         : Logical File number in the␣
↪DLIS file
FRAME-ARRAY.   800T                                      : Identity of the Frame Array␣
↪in the Logical File
~Well Information Section
#MNEM.UNIT  DATA                          DESCRIPTION
#----.----  ----                          -----------
STRT.ms     16677259.0                    : Start X
STOP.ms     17597260.0                    : Stop X, frames: 2,301
STEP.ms     400.0004347826087             : Step (average)
NULL.                                     :
COMP.       Faroe Petroleum               :
WELL.       206/05a-3                     :
```

Processing a Directory

Use the `-r` option to process recursively. The output directory will mirror the input directory.

```
$ tdrp66v1tolas -r example_data/ tmp/LAS
  Input    Output LAS Count  Time  Ratio  ms/Mb Exception                                ␣
↪                          Path
------- --------- --------- ----- ------ ------ --------- ----------------------------
↪--------------------------------
540,372 1,812,131         2 1.874 335.3% 3636.8     False "example_data/RP66V1/206_
↪05a-_3_DWL_DWL_WIRE_258276498.DLIS"
```

(continues on next page)

(continued from previous page)

```
Execution time =    1.884 (S)
Out of  6 processed 1 files of total size 540,372 input bytes
Wrote 1,812,131 output bytes, ratio: 335.349% at 3655.1 ms/Mb
$ find tmp/LAS -name '*.las'
tmp/LAS/RP66V1/206_05a-_3_DWL_DWL_WIRE_258276498_0_800T.las
tmp/LAS/RP66V1/206_05a-_3_DWL_DWL_WIRE_258276498_0_2000T.las
```

Indexing RP66V1 Files with `tdrp66v1indexpickle`

`tdrp66v1indexpickle` reads a RP66V1 file and dumps the index to a pickle file.

Arguments

The first argument is the path to a RP66V1 file or directory. The second argument is the path to write the output to.

Options

-h, --help show this help message and exit

-r, --recurse Process recursively. [default: False]

--read-back Read and time the output. [default: False]

-l LOG_LEVEL, --log-level LOG_LEVEL Log Level as an integer or symbol. (0<->NOTSET, 10<->DEBUG, 20<->INFO, 30<->WARNING, 40<->ERROR, 50<->CRITICAL) [default: 30]

--log-process LOG_PROCESS Writes process data such as memory usage as a log INFO line every LOG_PROCESS seconds. If 0.0 no process data is logged. [default: 0.0]

-v, --verbose Increase verbosity, additive [default: 0]

--gnuplot GNUPLOT Directory to write the gnuplot data.

Examples

Processing a Single File

```
$ tdrp66v1indexpickle --read-back example_data/RP66V1/206_05a-_3_DWL_DWL_WIRE_
↪258276498.DLIS example_data/pickle/206_05a-_3_DWL_DWL_WIRE_258276498
Common path prefix: example_data/RP66V1/206_05a-_3_DWL_DWL_WIRE_258276498.DLIS
Size (b) | Index (b) | Ratio (%) | Index (s) | Index (ms/Mb) | Read (s) | Read (ms/
↪Mb) | Except | Path
-------- | --------- | --------- | --------- | ------------- | -------- | ------------
↪ | ------ | ----
 540,372 | 1,018,327 |  188.449% |     0.330 |         639.9 |    0.041 |        78.
↪96 |  False |
Execution time =    0.379 (S)
Out of  1 processed 1 files of total size 540,372 input bytes
Wrote 1,018,327 output bytes, ratio: 188.449% at 651.0 ms/Mb
$ ll example_data/pickle/
```

(continues on next page)

(continued from previous page)

```
total 1992
-rw-r--r--  1 xxxxxxxx  staff  1018327 28 Oct 12:11 206_05a-_3_DWL_DWL_WIRE_258276498.
↪pkl
```

Processing a Directory

Use the `-r` option to process recursively. The output directory will mirror the input directory.

Indexing RP66V1 Files with `tdrp66v1indexxml`

`tdrp66v1indexxml` reads a RP66V1 file and dumps the index to an XML file.

Arguments

The first argument is the path to a RP66V1 file or directory. The second argument is the path to write the output to.

Options

optional arguments:

-h, --help show this help message and exit

-r, --recurse Process files recursively. [default: False]

-p, --private Also write out private EFLRs. [default: False]

-l LOG_LEVEL, --log-level LOG_LEVEL Log Level as an integer or symbol. (0<->NOTSET, 10<->DEBUG, 20<->INFO, 30<->WARNING, 40<->ERROR, 50<->CRITICAL) [default: 20]

--log-process LOG_PROCESS Writes process data such as memory usage as a log INFO line every LOG_PROCESS seconds. If 0.0 no process data is logged. [default: 0.0]

-v, --verbose Increase verbosity, additive [default: 0]

--gnuplot GNUPLOT Directory to write the gnuplot data.

Examples

Processing a Single File

```
$ tdrp66v1indexxml example_data/RP66V1/206_05a-_3_DWL_DWL_WIRE_258276498.DLIS example_
↪data/XML/206_05a-_3_DWL_DWL_WIRE_258276498
2019-10-28 11:58:55,498 - 74153 - MainThread - INFO     - IndexXML.py      - index_
↪dir_or_file(): "example_data/RP66V1/206_05a-_3_DWL_DWL_WIRE_258276498.DLIS" to
↪"example_data/XML/206_05a-_3_DWL_DWL_WIRE_258276498" recurse: False
2019-10-28 11:58:55,499 - 74153 - MainThread - INFO     - IndexXML.py      - Making␣
↪directory: example_data/XML
2019-10-28 11:58:55,499 - 74153 - MainThread - INFO     - IndexXML.py      - Indexing␣
↪example_data/RP66V1/206_05a-_3_DWL_DWL_WIRE_258276498.DLIS to example_data/XML/206_
↪05a-_3_DWL_DWL_WIRE_258276498
```

(continues on next page)

(continued from previous page)

```
2019-10-28 11:58:55,939 - 74153 - MainThread - INFO     - IndexXML.py      - Length␣
↪of XML: 428622
         Size In         Size Out     Time  Ratio %    ms/Mb Fail? Path
---------------- ---------------- -------- -------- -------- ----- ----
         540,372          428,622    0.440  79.320%    854.6 False "example_data/
↪RP66V1/206_05a-_3_DWL_DWL_WIRE_258276498.DLIS"
Execution time =    0.443 (S)
Out of  1 processed 1 files of total size 540,372 input bytes
Wrote 428,622 output bytes, ratio:  79.320% at 860.4 ms/Mb
```

The XML looks something like this:

```
<?xml version='1.0' encoding="utf-8"?>
<RP66V1FileIndex creator="TotalDepth.RP66V1.core.Index" path="example_data/RP66V1/206_
↪05a-_3_DWL_DWL_WIRE_258276498.DLIS" schema_version="0.1.0" size="540372" utc_file_
↪mtime="2019-06-22 09:10:59.512253" utc_now="2019-10-28 11:58:55.799047">
  <StorageUnitLabel dlis_version="V1.00" maximum_record_length="8192" sequence_number=
↪"1" storage_set_identifier="Default Storage Set                                    ␣
↪     " storage_unit_structure="RECORD"/>
  <LogicalFiles count="1">
    <LogicalFile has_log_pass="True" index="0">
      <EFLR lr_type="0" lrsh_position="0x54" object_count="1" set_name="" set_type=
↪"FILE-HEADER" vr_position="0x50">
        <Object C="0" I="5" O="2">
          <Attribute count="1" label="SEQUENCE-NUMBER" rc="20" rc_ascii="ASCII" units=
↪"">
            <Value type="bytes" value="       197"/>
          </Attribute>
          <Attribute count="1" label="ID" rc="20" rc_ascii="ASCII" units="">
            <Value type="bytes" value="MSCT_197LTP                                  ␣
↪                  "/>
          </Attribute>
        </Object>
      </EFLR>
      <!-- More EFLRs ... -->
      <LogPass count="2">
        <FrameArray C="0" I="2000T" O="2" description="" x_axis="TIME" x_units="ms">
          <Channels count="4">
            <Channel C="4" I="TIME" O="2" count="1" dimensions="1" long_name="1␣
↪second River Time" rep_code="2" units="ms"/>
            <Channel C="4" I="TDEP" O="2" count="1" dimensions="1" long_name="1␣
↪second River Depth" rep_code="2" units="0.1 in"/>
            <Channel C="0" I="TENS_SL" O="2" count="1" dimensions="1" long_name=
↪"Cable Tension" rep_code="2" units="lbf"/>
            <Channel C="0" I="DEPT_SL" O="2" count="1" dimensions="1" long_name=
↪"Station logging depth" rep_code="2" units="0.1 in"/>
          </Channels>
          <IFLR count="921">
            <FrameNumbers count="921" rle_len="1">
              <RLE datum="1" repeat="920" stride="1"/>
            </FrameNumbers>
            <LRSH count="921" rle_len="400">
              <RLE datum="0x13254" repeat="1" stride="0x190"/>
              <!-- ... -->
              <RLE datum="0x83ba4" repeat="1" stride="0x198"/>
            </LRSH>
            <Xaxis count="921" rle_len="2">
```

(continues on next page)

(continued from previous page)

```
              <RLE datum="16677259.0" repeat="99" stride="1000.0"/>
              <RLE datum="16777260.0" repeat="820" stride="1000.0"/>
            </Xaxis>
          </IFLR>
        </FrameArray>
        <FrameArray C="0" I="800T" O="2" description="" x_axis="TIME" x_units="ms">
          <Channels count="43">
            <Channel C="5" I="TIME" O="2" count="1" dimensions="1" long_name="400␣
↪milli-second time channel" rep_code="2" units="ms"/>
            <Channel C="5" I="TDEP" O="2" count="1" dimensions="1" long_name="MSCT␣
↪depth channel" rep_code="2" units="0.1 in"/>
            <Channel C="1" I="ETIM" O="2" count="1" dimensions="1" long_name="Elapsed␣
↪Logging Time" rep_code="2" units="s"/>
            <!-- ... -->
            <Channel C="0" I="HMCU" O="2" count="1" dimensions="1" long_name=
↪"Hydrailic Motor Current" rep_code="2" units="mA"/>
            <Channel C="0" I="CMLP" O="2" count="1" dimensions="1" long_name="Coring␣
↪Motor Linear Position" rep_code="2" units="in"/>
          </Channels>
          <IFLR count="2301">
            <FrameNumbers count="2301" rle_len="1">
              <RLE datum="1" repeat="2300" stride="1"/>
            </FrameNumbers>
            <LRSH count="2301" rle_len="937">
              <RLE datum="0x13274" repeat="1" stride="0xb8"/>
              <!-- ... -->
              <RLE datum="0x83d5c" repeat="1" stride="0xbc"/>
            </LRSH>
            <Xaxis count="2301" rle_len="2">
              <RLE datum="16677259.0" repeat="249" stride="400.0"/>
              <RLE datum="16777260.0" repeat="2050" stride="400.0"/>
            </Xaxis>
          </IFLR>
        </FrameArray>
      </LogPass>
    </LogicalFile>
  </LogicalFiles>
  <VisibleRecords count="66" rle_len="15">
    <RLE datum="0x50" repeat="3" stride="0x2000"/>
    <!-- ... -->
    <RLE datum="0x81f70" repeat="0" stride="0x0"/>
  </VisibleRecords>
</RP66V1FileIndex>
```

Processing a Directory

Use the `-r` option to process recursively. The output directory will mirror the input directory.

Scanning RP66V1 Files with `tdrp66v1scan`

`tdrp66v1scan` scans a RP66V1 file and dumps data about the file to stdout. This is useful for examining the details of RP66V1 files and can dump data at various levels of encapsulation, from the lowest level upwards:

- `--VR` - Visible Records only.
- `--LRSH` - Logical Record segments.
- `--LD` - Logical data i.e. all Logical Record segments concatenated for each Logical Record.
- `--EFLR` - Explicitly Formatted Logical Records.
- `--IFLR` - Implicitly Formatted Logical Records.
- `--LR` - All data, including the numerical analysis of frame data.

If these options are combined then the input is scanned, and reported, multiple times.

Arguments

The first argument is the path to a RP66V1 file. An optional second argument is the path to write the output to. If absent then output is written to stdout.

Options

-h, --help show this help message and exit

-V, --VR Dump the Visible Records. [default: False]

-L, --LRSH Summarise the Visible Records and the Logical Record Segment Headers, use -v to dump records. [default: False]

-D, --LD Summarise logical data, use -v to dump records. See also –dump-bytes, –dump-raw-bytes. [default: False]

-E, --EFLR Dump EFLR Set. [default: False]

--eflr-set-type EFLR_SET_TYPE List of EFLR Set Types to output, additive, if absent then dump all. [default: []]

-I, --IFLR Dump IFLRs. [default: False]

--iflr-set-type IFLR_SET_TYPE List of IFLR Set Types to output, additive, if absent then dump all. [default: []]

-R, --LR Dump all data, including frame data from Logical Records. [default: False]

-d DUMP_BYTES, --dump-bytes DUMP_BYTES Dump X leading raw bytes for certain options, if -1 all bytes are dumped. [default: 0]

--dump-raw-bytes Dump the raw bytes for certain options in raw format, otherwise Hex format is used. [default: False]

-r, --recurse Process files recursively. [default: False]

-e, --encrypted Output encrypted Logical Records as well. [default: False]

-k, --keep-going Keep going as far as sensible. [default: False]

--frame-slice FRAME_SLICE NOTE: Requires -R, –LR. Do not process all frames but sample or slice the frames. SAMPLE: Sample is of the form “N” so a maximum of N frames, roughly regularly spaced, will be processed. N must be +ve, non-zero integer. Example: “64” - process a maximum of 64 frames. SLICE: Slice the frames is of the form start,stop,step as a comma separated list. Values can be absent or “None”. Examples: “,,” - every frame, “,,2” - every other frame, “,10,” - frames 0 to 9, “4,10,2” - frames 4, 6, 8, “40,-1,4” - every fourth frame from 40 to the end. Results will be truncated by frame array length. [default: “,,” i.e. all frames]

--eflr-as-table When with –LR and not –html then dump EFLRs as tables, otherwise every EFLR object. [default: False]

-l LOG_LEVEL, --log-level LOG_LEVEL Log Level as an integer or symbol. (0<->NOTSET, 10<->DEBUG, 20<->INFO, 30<->WARNING, 40<->ERROR, 50<->CRITICAL) [default: 30]

-v, --verbose Increase verbosity, additive [default: 0]

--gnuplot GNUPLOT Directory to write the gnuplot data.

-T, --test-data Dump the file as annotated bytes, useful for creating test data. [default: False]

Examples

Scanning Visible Records with --VR

Example of scanning a RP66V1 file:

```
$ tdrp66v1scan --VR example_data/RP66V1/206_05a-_3_DWL_DWL_WIRE_258276498.DLIS
********************************************** RP66V1 Visible and LRSH Records␣
↪**********************************************
================================================== Summary of Visible Records␣
↪==================================================
Visible records: 66
-------------------------------------------------- RLE Visible Record Position ------
↪----------------------------------------------
Datum:                80 0x00000050 Repeat:       3 Stride:  8,192 0x2000
Datum:            32,844 0x0000804c Repeat:       5 Stride:  8,192 0x2000
Datum:            81,988 0x00014044 Repeat:       4 Stride:  8,192 0x2000
Datum:           122,940 0x0001e03c Repeat:      10 Stride:  8,192 0x2000
Datum:           213,016 0x00034018 Repeat:       3 Stride:  8,192 0x2000
Datum:           245,764 0x0003c004 Repeat:       3 Stride:  8,192 0x2000
Datum:           278,516 0x00043ff4 Repeat:       3 Stride:  8,192 0x2000
Datum:           311,268 0x0004bfe4 Repeat:       3 Stride:  8,192 0x2000
Datum:           344,020 0x00053fd4 Repeat:       2 Stride:  8,192 0x2000
Datum:           368,576 0x00059fc0 Repeat:       4 Stride:  8,192 0x2000
Datum:           409,524 0x00063fb4 Repeat:       2 Stride:  8,192 0x2000
Datum:           434,080 0x00069fa0 Repeat:       3 Stride:  8,192 0x2000
Datum:           466,832 0x00071f90 Repeat:       3 Stride:  8,192 0x2000
Datum:           499,584 0x00079f80 Repeat:       3 Stride:  8,192 0x2000
```

(continues on next page)

(continued from previous page)

```
Datum:          532,336 0x00081f70 Repeat:      0 Stride:      0 0x0000
------------------------------------------------ END RLE Visible Record Position ----
↪------------------------------------------------
================================================= END Summary of Visible Records␣
↪=================================================
***************************************** END RP66V1 Visible and LRSH Records␣
↪*****************************************
```

And with the -v option:

```
$ tdrp66v1scan --VR -v example_data/RP66V1/206_05a-_3_DWL_DWL_WIRE_258276498.DLIS
************************************************* RP66V1 Visible and LRSH Records␣
↪*************************************************
<VisibleRecord: position=0x00000050 length=0x2000 version=0xff01> Stride: 0x00000050 ␣
↪    80
<VisibleRecord: position=0x00002050 length=0x2000 version=0xff01> Stride: 0x00002000 ␣
↪8,192
<VisibleRecord: position=0x00004050 length=0x2000 version=0xff01> Stride: 0x00002000 ␣
↪8,192
<VisibleRecord: position=0x00006050 length=0x1ffc version=0xff01> Stride: 0x00002000 ␣
↪8,192
<VisibleRecord: position=0x0000804c length=0x2000 version=0xff01> Stride: 0x00001ffc ␣
↪8,188
<VisibleRecord: position=0x0000a04c length=0x2000 version=0xff01> Stride: 0x00002000 ␣
↪8,192
<VisibleRecord: position=0x0000c04c length=0x2000 version=0xff01> Stride: 0x00002000 ␣
↪8,192
<VisibleRecord: position=0x0000e04c length=0x2000 version=0xff01> Stride: 0x00002000 ␣
↪8,192
<VisibleRecord: position=0x0001004c length=0x2000 version=0xff01> Stride: 0x00002000 ␣
↪8,192
<VisibleRecord: position=0x0001204c length=0x1ff8 version=0xff01> Stride: 0x00002000 ␣
↪8,192
<VisibleRecord: position=0x00014044 length=0x2000 version=0xff01> Stride: 0x00001ff8 ␣
↪8,184
<VisibleRecord: position=0x00016044 length=0x2000 version=0xff01> Stride: 0x00002000 ␣
↪8,192
<VisibleRecord: position=0x00018044 length=0x2000 version=0xff01> Stride: 0x00002000 ␣
↪8,192
<VisibleRecord: position=0x0001a044 length=0x2000 version=0xff01> Stride: 0x00002000 ␣
↪8,192
<VisibleRecord: position=0x0001c044 length=0x1ff8 version=0xff01> Stride: 0x00002000 ␣
↪8,192
<VisibleRecord: position=0x0001e03c length=0x2000 version=0xff01> Stride: 0x00001ff8 ␣
↪8,184
<VisibleRecord: position=0x0002003c length=0x2000 version=0xff01> Stride: 0x00002000 ␣
↪8,192
...
<VisibleRecord: position=0x0007df80 length=0x2000 version=0xff01> Stride: 0x00002000 ␣
↪8,192
<VisibleRecord: position=0x0007ff80 length=0x1ff0 version=0xff01> Stride: 0x00002000 ␣
↪8,192
<VisibleRecord: position=0x00081f70 length=0x1f64 version=0xff01> Stride: 0x00001ff0 ␣
↪8,176
================================================= Summary of Visible Records␣
↪=================================================
```

Scanning Logical Record Segments with `--LRSH`

Example of scanning a RP66V1 file for Logical Record Segments, this gives just a summary:

```
$ tdrp66v1scan --LRSH example_data/RP66V1/206_05a-_3_DWL_DWL_WIRE_258276498.DLIS
...
============================================================ Summary of LRSH␣
↪============================================================
LRSH: total=3,303 is_first=3252
LRSH: record types and counts (first segments only) [9]:
  0 :    3,223
  1 :        1
  3 :        1
  4 :        1
  5 :       10
128 :        2
129 :        2
132 :       10
133 :        2
LRSH: record lengths and counts (all segments) [62] range: 16...8188
============================================================ END Summary of LRSH␣
↪============================================================
```

And with the `-v` option gives the Visible Records and Logical Record Segments:

```
************************************************ RP66V1 Visible and LRSH Records␣
↪************************************************
<VisibleRecord: position=0x00000050 length=0x2000 version=0xff01> Stride: 0x00000050 ␣
↪   80
   <LogicalRecordSegmentHeader: position=0x00000054 length=0x007c attributes=0x80 LR␣
↪type=  0> Stride: 0x00000054     84
   <LogicalRecordSegmentHeader: position=0x000000d0 length=0x0504 attributes=0x81 LR␣
↪type=  1> Stride: 0x0000007c    124
   <LogicalRecordSegmentHeader: position=0x000005d4 length=0x05e0 attributes=0x81 LR␣
↪type=  5> Stride: 0x00000504  1,284
   <LogicalRecordSegmentHeader: position=0x00000bb4 length=0x03e4 attributes=0x99 LR␣
↪type=132> Stride: 0x000005e0  1,504
   <LogicalRecordSegmentHeader: position=0x00000f98 length=0x0254 attributes=0x99 LR␣
↪type=132> Stride: 0x000003e4    996
   <LogicalRecordSegmentHeader: position=0x000011ec length=0x0588 attributes=0x81 LR␣
↪type=  5> Stride: 0x00000254    596
   <LogicalRecordSegmentHeader: position=0x00001774 length=0x023c attributes=0x98 LR␣
↪type=132> Stride: 0x00000588  1,416
   <LogicalRecordSegmentHeader: position=0x000019b0 length=0x0084 attributes=0x98 LR␣
↪type=132> Stride: 0x0000023c    572
   <LogicalRecordSegmentHeader: position=0x00001a34 length=0x061c attributes=0xa0 LR␣
↪type=132> Stride: 0x00000084    132
<VisibleRecord: position=0x00002050 length=0x2000 version=0xff01> Stride: 0x00002000 ␣
↪8,192
    --<LogicalRecordSegmentHeader: position=0x00002054 length=0x0304 attributes=0xc1␣
↪LR type=132> Stride: 0x00000620  1,568
   <LogicalRecordSegmentHeader: position=0x00002358 length=0x0e3c attributes=0x81 LR␣
↪type=  5> Stride: 0x00000304    772
   <LogicalRecordSegmentHeader: position=0x00003194 length=0x0ebc attributes=0xb9 LR␣
↪type=132> Stride: 0x00000e3c  3,644
<VisibleRecord: position=0x00004050 length=0x2000 version=0xff01> Stride: 0x00002000 ␣
↪8,192
```

(continues on next page)

(continued from previous page)

```
    --<LogicalRecordSegmentHeader: position=0x00004054 length=0x0110 attributes=0xd9␣
↪LR type=132> Stride: 0x00000ec0  3,776
   <LogicalRecordSegmentHeader: position=0x00004164 length=0x1eec attributes=0xa0 LR␣
↪type=  5> Stride: 0x00000110    272
<VisibleRecord: position=0x00006050 length=0x1ffc version=0xff01> Stride: 0x00002000 ␣
↪8,192
    --<LogicalRecordSegmentHeader: position=0x00006054 length=0x1864 attributes=0xc1␣
↪LR type=  5> Stride: 0x00001ef0  7,920
   <LogicalRecordSegmentHeader: position=0x000078b8 length=0x0794 attributes=0xb9 LR␣
↪type=132> Stride: 0x00001864  6,244
<VisibleRecord: position=0x0000804c length=0x2000 version=0xff01> Stride: 0x00001ffc ␣
↪8,188
    --<LogicalRecordSegmentHeader: position=0x00008050 length=0x1080 attributes=0xd9␣
↪LR type=132> Stride: 0x00000798  1,944
   <LogicalRecordSegmentHeader: position=0x000090d0 length=0x01e0 attributes=0x81 LR␣
↪type=  5> Stride: 0x00001080  4,224
   <LogicalRecordSegmentHeader: position=0x000092b0 length=0x023c attributes=0x99 LR␣
↪type=132> Stride: 0x000001e0    480
   <LogicalRecordSegmentHeader: position=0x000094ec length=0x0314 attributes=0x81 LR␣
↪type=  5> Stride: 0x0000023c    572
   <LogicalRecordSegmentHeader: position=0x00009800 length=0x0154 attributes=0x99 LR␣
↪type=128> Stride: 0x00000314    788
   <LogicalRecordSegmentHeader: position=0x00009954 length=0x0238 attributes=0x81 LR␣
↪type=  5> Stride: 0x00000154    340
   <LogicalRecordSegmentHeader: position=0x00009b8c length=0x0270 attributes=0x81 LR␣
↪type=  5> Stride: 0x00000238    568
   <LogicalRecordSegmentHeader: position=0x00009dfc length=0x0250 attributes=0xa0 LR␣
↪type=  5> Stride: 0x00000270    624
...
```

Scanning Logical Data with `--LD`

Example of scanning a RP66V1 file for Logical Record Segments, this gives just a summary:

```
$ tdrp66v1scan --LD example_data/RP66V1/206_05a-_3_DWL_DWL_WIRE_258276498.DLIS
Cmd: tdrp66v1scan --LD example_data/RP66V1/206_05a-_3_DWL_DWL_WIRE_258276498.DLIS
gnuplot version: "b'gnuplot 5.2 patchlevel 6'"
args: Namespace(EFLR=False, IFLR=False, LD=True, LR=False, LRSH=False, VR=False, dump_
↪bytes=0, dump_raw_bytes=False, eflr_as_table=False, eflr_set_type=[],␣
↪encrypted=False, frame_slice=',,', gnuplot=None, iflr_set_type=[], keep_going=False,
↪ log_level=30, path_in='example_data/RP66V1/206_05a-_3_DWL_DWL_WIRE_258276498.DLIS',
↪ path_out='', recurse=False, verbose=0)
Use -v to see individual logical data.
Use -v and --dump-bytes to see actual first n bytes.
************************************************** RP66V1 Logical Data Summary␣
↪**************************************************
================================================== RP66V1 Logical Data EFLR Summary␣
↪==================================================
Total number of EFLR records: 30
Total length of EFLR records: 78,109
EFLR record type 0 lengths and count [1]:
       120:          1
EFLR record type 1 lengths and count [1]:
     1,279:          1
EFLR record type 3 lengths and count [1]:
```

(continues on next page)

(continued from previous page)

```
     7,174:          1
EFLR record type 4 lengths and count [1]:
       572:          1
EFLR record type 5 lengths and count [10]:
       181:          1
       475:          1
       561:          1
       617:          1
       781:          1
$ tdrp66v1scan --LD example_data/RP66V1/206_05a-_3_DWL_DWL_WIRE_258276498.DLIS
Cmd: tdrp66v1scan --LD example_data/RP66V1/206_05a-_3_DWL_DWL_WIRE_258276498.DLIS
gnuplot version: "b'gnuplot 5.2 patchlevel 6'"
args: Namespace(EFLR=False, IFLR=False, LD=True, LR=False, LRSH=False, VR=False, dump_
↪bytes=0, dump_raw_bytes=False, eflr_as_table=False, eflr_set_type=[],␣
↪encrypted=False, frame_slice=',,', gnuplot=None, iflr_set_type=[], keep_going=False,
↪ log_level=30, path_in='example_data/RP66V1/206_05a-_3_DWL_DWL_WIRE_258276498.DLIS',
↪ path_out='', recurse=False, verbose=0)
Use -v to see individual logical data.
Use -v and --dump-bytes to see actual first n bytes.
************************************************** RP66V1 Logical Data Summary␣
↪**************************************************
================================================== RP66V1 Logical Data EFLR Summary␣
↪==================================================
Total number of EFLR records: 30
Total length of EFLR records: 78,109
EFLR record type 0 lengths and count [1]:
       120:          1
EFLR record type 1 lengths and count [1]:
     1,279:          1
EFLR record type 3 lengths and count [1]:
     7,174:          1
EFLR record type 4 lengths and count [1]:
       572:          1
EFLR record type 5 lengths and count [10]:
       181:          1
       475:          1
       561:          1
       617:          1
       781:          1
     1,409:          1
     1,497:          1
     1,620:          1
     3,637:          1
    14,149:          1
EFLR record type 128 lengths and count [2]:
       336:          1
       888:          1
EFLR record type 129 lengths and count [2]:
       111:          1
     1,226:          1
EFLR record type 132 lengths and count [9]:
       128:          1
       288:          1
       512:          1
       568:          2
       592:          1
       992:          1
```

(continues on next page)

(continued from previous page)

```
     2,325:          1
     4,036:          1
     6,156:          1
EFLR record type 133 lengths and count [2]:
       999:          1
    24,312:          1
============================================== END RP66V1 Logical Data EFLR Summary␣
↪==============================================
================================================ RP66V1 Logical Data IFLR Summary␣
↪================================================
Total number of IFLR records: 3,222
Total length of IFLR records: 440,173
IFLR record type 0 lengths and count [4]:
        25:        127
        26:        794
       180:        127
       181:      2,174
============================================== END RP66V1 Logical Data IFLR Summary␣
↪==============================================
Total length EFLR/IFLR: 17.745%
************************************************ END RP66V1 Logical Data Summary␣
↪************************************************
         540,372         -1    0.059  -0.000%    115.4 False "example_data/RP66V1/206_
↪05a-_3_DWL_DWL_WIRE_258276498.DLIS"
Execution time =    0.060 (S)
Processed 1 files and 540,372 bytes, 115.8 ms/Mb
```

And with the −v option gives the Visible Records and Logical Record Segments. The letter 'E' is for EFLRs and 'I' for IFLRs, 'Plain' is for un-encrypted records and 'Crypt' for encrypted records:

```
$ tdrp66v1scan --LD -v example_data/RP66V1/206_05a-_3_DWL_DWL_WIRE_258276498.DLIS
Use -v and --dump-bytes to see actual first n bytes.
************************************************** RP66V1 Logical Data Summary␣
↪**************************************************
Visible R  LRSH       Typ         Length
---------- ---------- --- - ----- --------
0x00000050 0x00000054   0 E Plain      120
           0x000000d0   1 E Plain    1,279
           0x000005d4   5 E Plain    1,497
           0x00000bb4 132 E Crypt      992
           0x00000f98 132 E Crypt      592
           0x000011ec   5 E Plain    1,409
           0x00001774 132 E Crypt      568
           0x000019b0 132 E Crypt      128
           0x00001a34 132 E Plain    2,325
0x00002050 0x00002358   5 E Plain    3,637
           0x00003194 132 E Crypt    4,036
0x00004050 0x00004164   5 E Plain   14,149
0x00006050 0x000078b8 132 E Crypt    6,156
0x0000804c 0x000090d0   5 E Plain      475
           0x000092b0 132 E Crypt      568
           0x000094ec   5 E Plain      781
           0x00009800 128 E Crypt      336
           0x00009954   5 E Plain      561
           0x00009b8c   5 E Plain      617
           0x00009dfc   5 E Plain    1,620
```

(continues on next page)

(continued from previous page)

```
...
0x00081f70 0x00081f74   0 I Plain      181
           0x00082030   0 I Plain       26
           0x00082050   0 I Plain      181
           0x0008210c   0 I Plain      181
           0x000821c8   0 I Plain      181
           ...
           0x00083d3c   0 I Plain       26
           0x00083d5c   0 I Plain      181
           0x00083e18   0 I Plain      181
================================================ RP66V1 Logical Data EFLR Summary␣
↪================================================
Total number of EFLR records: 30
Total length of EFLR records: 78,109
EFLR record type 0 lengths and count [1]:
       120:          1
EFLR record type 1 lengths and count [1]:
     1,279:          1
EFLR record type 3 lengths and count [1]:
     7,174:          1
EFLR record type 4 lengths and count [1]:
       572:          1
EFLR record type 5 lengths and count [10]:
       181:          1
       475:          1
       561:          1
       617:          1
       781:          1
     1,409:          1
     1,497:          1
     1,620:          1
     3,637:          1
    14,149:          1
EFLR record type 128 lengths and count [2]:
       336:          1
       888:          1
EFLR record type 129 lengths and count [2]:
       111:          1
     1,226:          1
EFLR record type 132 lengths and count [9]:
       128:          1
       288:          1
       512:          1
       568:          2
       592:          1
       992:          1
     2,325:          1
     4,036:          1
     6,156:          1
EFLR record type 133 lengths and count [2]:
       999:          1
    24,312:          1
============================================== END RP66V1 Logical Data EFLR Summary␣
↪==============================================
================================================ RP66V1 Logical Data IFLR Summary␣
↪================================================
Total number of IFLR records: 3,222
```

(continues on next page)

(continued from previous page)

```
Total length of IFLR records: 440,173
IFLR record type 0 lengths and count [4]:
        25:        127
        26:        794
       180:        127
       181:      2,174
============================================== END RP66V1 Logical Data IFLR Summary␣
↪==============================================
Total length EFLR/IFLR: 17.745%
************************************************ END RP66V1 Logical Data Summary␣
↪************************************************
```

The `--dump-bytes` combined with `-v` shows the initial bytes of each logical record, here the first 16 bytes are dumped:

```
$ tdrp66v1scan --LD -v --dump-bytes=16 example_data/RP66V1/206_05a-_3_DWL_DWL_WIRE_
↪258276498.DLIS
************************************************** RP66V1 Logical Data Summary␣
↪**************************************************
Visible R  LRSH       Typ         Length
---------- ---------- --- - ----- --------
0x00000050 0x00000054   0 E Plain      120 f00b 4649 4c45 2d48 4541 4445 5234 0f53 ..
↪FILE-HEADER4.S
           0x000000d0   1 E Plain    1,279 f006 4f52 4947 494e 3c07 4649 4c45 2d49 ..
↪ORIGIN<.FILE-I
           0x000005d4   5 E Plain    1,497 f809 4551 5549 504d 454e 5402 3531 3006 ..
↪EQUIPMENT.510.
           0x00000bb4 132 E Crypt      992 0018 01b8 ced6 0000 be18 0000 8467 0000 ...
↪..........g..
           0x00000f98 132 E Crypt      592 0018 01b8 dee9 0000 4916 0000 f16d 0000 ...
↪.....I....m..
           0x000011ec   5 E Plain    1,409 f804 544f 4f4c 0235 3430 0a50 4152 414d ..
↪TOOL.540.PARAM
           0x00001774 132 E Crypt      568 0018 01b8 9a99 0000 3c15 0000 877e 0000 ...
↪.....<....~..
           0x000019b0 132 E Crypt      128 0018 01b8 acb3 0000 064d 0000 b74d 0000 ...
↪......M...M..
           0x00001a34 132 E Plain    2,325 f80b 3434 302d 4348 414e 4e45 4c02 3537 ..
↪440-CHANNEL.57
0x00002050 0x00002358   5 E Plain    3,637 f809 5041 5241 4d45 5445 5202 3538 3006 ..
↪PARAMETER.580.
           0x00003194 132 E Crypt    4,036 0018 01b8 9aa6 0000 c84d 0000 4364 0000 ...
↪......M..Cd..
0x00004050 0x00004164   5 E Plain   14,149 f809 5041 5241 4d45 5445 5202 3630 3006 ..
↪PARAMETER.600.
0x00006050 0x000078b8 132 E Crypt    6,156 0018 01b8 565d 0000 0945 0000 3812 0000 ...
↪.V]...E..8...
0x0000804c 0x000090d0   5 E Plain      475 f809 5041 5241 4d45 5445 5202 3632 3006 ..
↪PARAMETER.620.
           0x000092b0 132 E Crypt      568 0018 01b8 010d 0000 f57f 0000 890a 0000 ...
↪..............
           0x000094ec   5 E Plain      781 f817 4341 4c49 4252 4154 494f 4e2d 4d45 ..
↪CALIBRATION-ME
           0x00009800 128 E Crypt      336 0018 01b8 4550 0000 ae56 0000 3207 0000 ...
↪.EP...V..2...
           0x00009954   5 E Plain      561 f817 4341 4c49 4252 4154 494f 4e2d 434f ..
↪CALIBRATION-CO
```

(continues on next page)

(continued from previous page)

```
           0x00009b8c   5 E Plain      617 f817 4341 4c49 4252 4154 494f 4e2d 434f ..
↪CALIBRATION-CO
           0x00009dfc   5 E Plain    1,620 f80b 4341 4c49 4252 4154 494f 4e02 3734 ..
↪CALIBRATION.74
...
```

The raw bytes object is dumped of the `--dump-raw-bytes` flag is used along with `--dump-bytes` combined with `-v`. This can be useful for creating test cases:

```
$ tdrp66v1scan --LD -v --dump-bytes=16 --dump-raw-bytes example_data/RP66V1/206_05a-_
↪3_DWL_DWL_WIRE_258276498.DLIS | head -n 40
************************************************** RP66V1 Logical Data Summary␣
↪**************************************************
Visible R  LRSH       Typ         Length
---------- ---------- --- - ----- --------
0x00000050 0x00000054   0 E Plain      120 b'\xf0\x0bFILE-HEADER4\x0fS'
           0x000000d0   1 E Plain    1,279 b'\xf0\x06ORIGIN<\x07FILE-I'
           0x000005d4   5 E Plain    1,497 b'\xf8\tEQUIPMENT\x02510\x06'
           0x00000bb4 132 E Crypt      992 b'\x00\x18\x01\xb8\xce\xd6\x00\x00\xbe\x18\
↪x00\x00\x84g\x00\x00'
           0x00000f98 132 E Crypt      592 b'\x00\x18\x01\xb8\xde\xe9\x00\x00I\x16\
↪x00\x00\xf1m\x00\x00'
           0x000011ec   5 E Plain    1,409 b'\xf8\x04TOOL\x02540\nPARAM'
           0x00001774 132 E Crypt      568 b'\x00\x18\x01\xb8\x9a\x99\x00\x00<\x15\
↪x00\x00\x87~\x00\x00'
           0x000019b0 132 E Crypt      128 b'\x00\x18\x01\xb8\xac\xb3\x00\x00\x06M\
↪x00\x00\xb7M\x00\x00'
           0x00001a34 132 E Plain    2,325 b'\xf8\x0b440-CHANNEL\x0257'
0x00002050 0x00002358   5 E Plain    3,637 b'\xf8\tPARAMETER\x02580\x06'
           0x00003194 132 E Crypt    4,036 b'\x00\x18\x01\xb8\x9a\xa6\x00\x00\xc8M\
↪x00\x00Cd\x00\x00'
0x00004050 0x00004164   5 E Plain   14,149 b'\xf8\tPARAMETER\x02600\x06'
0x00006050 0x000078b8 132 E Crypt    6,156 b'\x00\x18\x01\xb8V]\x00\x00\tE\x00\x008\
↪x12\x00\x00'
0x0000804c 0x000090d0   5 E Plain      475 b'\xf8\tPARAMETER\x02620\x06'
           0x000092b0 132 E Crypt      568 b'\x00\x18\x01\xb8\x01\r\x00\x00\xf5\x7f\
↪x00\x00\x89\n\x00\x00'
           0x000094ec   5 E Plain      781 b'\xf8\x17CALIBRATION-ME'
           0x00009800 128 E Crypt      336 b'\x00\x18\x01\xb8EP\x00\x00\xaeV\x00\x002\
↪x07\x00\x00'
           0x00009954   5 E Plain      561 b'\xf8\x17CALIBRATION-CO'
           0x00009b8c   5 E Plain      617 b'\xf8\x17CALIBRATION-CO'
           0x00009dfc   5 E Plain    1,620 b'\xf8\x0bCALIBRATION\x0274'
...
```

Scanning Explicitly Formatted Logical Records with `--EFLR`

Example of scanning a RP66V1 file for Logical Record Segments, this gives just a summary:

```
$ tdrp66v1scan --EFLR example_data/RP66V1/206_05a-_3_DWL_DWL_WIRE_258276498.DLIS
Use -v to see individual logical data.
********************************************** RP66V1 EFLR and IFLR Data Summary␣
↪**********************************************
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'FILE-HEADER' name: b''>
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'ORIGIN' name: b''>
```

(continues on next page)

(continued from previous page)

```
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'EQUIPMENT' name: b'51'>
Encrypted EFLR: VR: 0x00000050 LRSH: 0x00000bb4
Encrypted EFLR: VR: 0x00000050 LRSH: 0x00000f98
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'TOOL' name: b'54'>
Encrypted EFLR: VR: 0x00000050 LRSH: 0x00001774
Encrypted EFLR: VR: 0x00000050 LRSH: 0x000019b0
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'440-CHANNEL' name: b'57'>
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'PARAMETER' name: b'58'>
Encrypted EFLR: VR: 0x00002050 LRSH: 0x00003194
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'PARAMETER' name: b'60'>
Encrypted EFLR: VR: 0x00006050 LRSH: 0x000078b8
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'PARAMETER' name: b'62'>
Encrypted EFLR: VR: 0x0000804c LRSH: 0x000092b0
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'CALIBRATION-MEASUREMENT' name: b'64
↪'>
Encrypted EFLR: VR: 0x0000804c LRSH: 0x00009800
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'CALIBRATION-COEFFICIENT' name: b'72
↪'>
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'CALIBRATION-COEFFICIENT' name: b'73
↪'>
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'CALIBRATION' name: b'74'>
Encrypted EFLR: VR: 0x0000a04c LRSH: 0x0000a45c
Encrypted EFLR: VR: 0x0000a04c LRSH: 0x0000a7d8
Encrypted EFLR: VR: 0x0000a04c LRSH: 0x0000a8fc
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'PROCESS' name: b'78'>
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'440-OP-CORE_TABLES' name: b'79'>
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'440-OP-CORE_REPORT_FORMAT' name: b
↪'330'>
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'CHANNEL' name: b''>
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'440-PRESENTATION-DESCRIPTION'␣
↪name: b'375'>
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'440-OP-CHANNEL' name: b'377'>
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'FRAME' name: b''>
********************************************** END RP66V1 EFLR and IFLR Data Summary␣
↪**********************************************
```

The -v flag can be added to see the initial data:

```
$ tdrp66v1scan --EFLR -v example_data/RP66V1/206_05a-_3_DWL_DWL_WIRE_258276498.DLIS
********************************************** RP66V1 EFLR and IFLR Data Summary␣
↪**********************************************
Visible R  LRSH       Typ         Length
---------- ---------- --- - ----- --------
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'FILE-HEADER' name: b''>
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'ORIGIN' name: b''>
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'EQUIPMENT' name: b'51'>
Encrypted EFLR: <FileLogicalData VR: 0x00000050 LRSH: 0x00000bb4 LR type 132 E y len␣
↪0x03e0 Idx 0x0000  0018 01b8 ced6 0000 be18 0000 8467 0000 .............g..>
Encrypted EFLR: <FileLogicalData VR: 0x00000050 LRSH: 0x00000f98 LR type 132 E y len␣
↪0x0250 Idx 0x0000  0018 01b8 dee9 0000 4916 0000 f16d 0000 ........I....m..>
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'TOOL' name: b'54'>
Encrypted EFLR: <FileLogicalData VR: 0x00000050 LRSH: 0x00001774 LR type 132 E y len␣
↪0x0238 Idx 0x0000  0018 01b8 9a99 0000 3c15 0000 877e 0000 ........<....~..>
Encrypted EFLR: <FileLogicalData VR: 0x00000050 LRSH: 0x000019b0 LR type 132 E y len␣
↪0x0080 Idx 0x0000  0018 01b8 acb3 0000 064d 0000 b74d 0000 .........M...M..>
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'440-CHANNEL' name: b'57'>
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'PARAMETER' name: b'58'>
```

(continues on next page)

(continued from previous page)

```
Encrypted EFLR: <FileLogicalData VR: 0x00002050 LRSH: 0x00003194 LR type 132 E y len␣
↪0x0fc4 Idx 0x0000  0018 01b8 9aa6 0000 c84d 0000 4364 0000 .........M..Cd..>
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'PARAMETER' name: b'60'>
Encrypted EFLR: <FileLogicalData VR: 0x00006050 LRSH: 0x000078b8 LR type 132 E y len␣
↪0x180c Idx 0x0000  0018 01b8 565d 0000 0945 0000 3812 0000 ....V]...E..8...>
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'PARAMETER' name: b'62'>
Encrypted EFLR: <FileLogicalData VR: 0x0000804c LRSH: 0x000092b0 LR type 132 E y len␣
↪0x0238 Idx 0x0000  0018 01b8 010d 0000 f57f 0000 890a 0000 ................>
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'CALIBRATION-MEASUREMENT' name: b'64
↪'>
Encrypted EFLR: <FileLogicalData VR: 0x0000804c LRSH: 0x00009800 LR type 128 E y len␣
↪0x0150 Idx 0x0000  0018 01b8 4550 0000 ae56 0000 3207 0000 ....EP...V..2...>
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'CALIBRATION-COEFFICIENT' name: b'72
↪'>
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'CALIBRATION-COEFFICIENT' name: b'73
↪'>
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'CALIBRATION' name: b'74'>
Encrypted EFLR: <FileLogicalData VR: 0x0000a04c LRSH: 0x0000a45c LR type 128 E y len␣
↪0x0378 Idx 0x0000  0018 01b8 eff6 0000 fd5c 0000 123e 0000 .........\...>..>
Encrypted EFLR: <FileLogicalData VR: 0x0000a04c LRSH: 0x0000a7d8 LR type 132 E y len␣
↪0x0120 Idx 0x0000  0018 01b8 4644 0000 ad4c 0000 4f31 0000 ....FD...L..O1..>
Encrypted EFLR: <FileLogicalData VR: 0x0000a04c LRSH: 0x0000a8fc LR type 132 E y len␣
↪0x0200 Idx 0x0000  0018 01b8 abb7 0000 d01c 0000 6b36 0000 ............k6..>
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'PROCESS' name: b'78'>
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'440-OP-CORE_TABLES' name: b'79'>
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'440-OP-CORE_REPORT_FORMAT' name: b
↪'330'>
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'CHANNEL' name: b''>
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'440-PRESENTATION-DESCRIPTION'␣
↪name: b'375'>
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'440-OP-CHANNEL' name: b'377'>
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'FRAME' name: b''>
********************************************** END RP66V1 EFLR and IFLR Data Summary␣
↪**********************************************
```

The `--eflr-set-type` can be used to select only specific EFLRs:

```
$ tdrp66v1scan --EFLR -v --eflr-set-type=ORIGIN --eflr-as-table example_data/RP66V1/
↪206_05a-_3_DWL_DWL_WIRE_258276498.DLIS
********************************************** RP66V1 EFLR and IFLR Data Summary␣
↪**********************************************
Visible R  LRSH       Typ         Length
---------- ---------- --- - ----- --------
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'ORIGIN' name: b''>
********************************************** END RP66V1 EFLR and IFLR Data Summary␣
↪**********************************************
```

Scanning Implicitly Formatted Logical Records with `--IFLR`

Example of scanning a RP66V1 file for Logical Record Segments, this gives just a summary:

```
$ tdrp66v1scan --IFLR example_data/RP66V1/206_05a-_3_DWL_DWL_WIRE_258276498.DLIS
Use -v to see individual logical data.
Use -v and --dump-bytes to see actual first n bytes.
********************************************** RP66V1 EFLR and IFLR Data Summary␣
↪**********************************************
<IndirectlyFormattedLogicalRecord b'2000T'   frame:         1 free data[  16]>
<IndirectlyFormattedLogicalRecord b'800T'    frame:         1 free data[ 172]>
<IndirectlyFormattedLogicalRecord b'800T'    frame:         2 free data[ 172]>
<IndirectlyFormattedLogicalRecord b'2000T'   frame:         2 free data[  16]>
<IndirectlyFormattedLogicalRecord b'800T'    frame:         3 free data[ 172]>
<IndirectlyFormattedLogicalRecord b'2000T'   frame:         3 free data[  16]>
<IndirectlyFormattedLogicalRecord b'800T'    frame:         4 free data[ 172]>
<IndirectlyFormattedLogicalRecord b'800T'    frame:         5 free data[ 172]>
<IndirectlyFormattedLogicalRecord b'800T'    frame:         6 free data[ 172]>
<IndirectlyFormattedLogicalRecord b'800T'    frame:         7 free data[ 172]>
<IndirectlyFormattedLogicalRecord b'2000T'   frame:         4 free data[  16]>
<IndirectlyFormattedLogicalRecord b'800T'    frame:         8 free data[ 172]>
<IndirectlyFormattedLogicalRecord b'800T'    frame:         9 free data[ 172]>
<IndirectlyFormattedLogicalRecord b'2000T'   frame:         5 free data[  16]>
<IndirectlyFormattedLogicalRecord b'800T'    frame:        10 free data[ 172]>
...
<IndirectlyFormattedLogicalRecord b'2000T'   frame:       920 free data[  16]>
<IndirectlyFormattedLogicalRecord b'800T'    frame:     2,298 free data[ 172]>
<IndirectlyFormattedLogicalRecord b'800T'    frame:     2,299 free data[ 172]>
<IndirectlyFormattedLogicalRecord b'2000T'   frame:       921 free data[  16]>
<IndirectlyFormattedLogicalRecord b'800T'    frame:     2,300 free data[ 172]>
<IndirectlyFormattedLogicalRecord b'800T'    frame:     2,301 free data[ 172]>
********************************************** END RP66V1 EFLR and IFLR Data Summary␣
↪**********************************************
         540,372         -1    0.435  -0.000%    844.6 False "example_data/RP66V1/206_
↪05a-_3_DWL_DWL_WIRE_258276498.DLIS"
Execution time =    0.436 (S)
Processed 1 files and 540,372 bytes, 845.2 ms/Mb
```

Scanning Everything with `--LR`

This reads every byte in the file and writes a very verbose output of each EFLR and a summary of each Log Pass. For example:

```
$ tdrp66v1scan --LR example_data/RP66V1/206_05a-_3_DWL_DWL_WIRE_258276498.DLIS
****************************************************** RP66V1 File Data Summary␣
↪******************************************************
StorageUnitLabel:
  Storage Unit Sequence Number: 1
                   DLIS Version: b'V1.00'
         Storage Unit Structure: b'RECORD'
          Maximum Record Length: 8192
         Storage Set Identifier: b'Default Storage Set
↪          '                                                                    ␣
============================================================ Logical File [0/1]␣
↪============================================================
<TotalDepth.RP66V1.core.LogicalFile.LogicalFile object at 0x104d3f6a0>
```

(continues on next page)

(continued from previous page)

```
---------------------------------------- EFLR [0/19] at VR: 0x00000050 LRSH:␣
↪0x00000054 ----------------------------------------
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'FILE-HEADER' name: b''>
  Template [2]:
    CD: 001 10100 L: b'SEQUENCE-NUMBER' C: 1 R: 20 (ASCII) U: b'' V: None
    CD: 001 10100 L: b'ID' C: 1 R: 20 (ASCII) U: b'' V: None
  Objects [1]:
    OBNAME: O: 2 C: 0 I: b'5'
      CD: 001 00001 L: b'SEQUENCE-NUMBER' C: 1 R: 20 (ASCII) U: b'' V: [b'       197']
      CD: 001 00001 L: b'ID' C: 1 R: 20 (ASCII) U: b'' V: [b'MSCT_197LTP             ␣
↪                                      ']
---------------------------------------- END EFLR [0/19] at VR: 0x00000050 LRSH:␣
↪0x00000054 ----------------------------------------
---------------------------------------- EFLR [1/19] at VR: 0x00000050 LRSH:␣
↪0x000000d0 ----------------------------------------
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'ORIGIN' name: b''>
  Template [20]:
    CD: 001 11100 L: b'FILE-ID' C: 1 R: 20 (ASCII) U: b'' V: None
    CD: 001 11100 L: b'FILE-SET-NAME' C: 1 R: 19 (IDENT) U: b'' V: None
    CD: 001 11100 L: b'FILE-SET-NUMBER' C: 1 R: 18 (UVARI) U: b'' V: None
    CD: 001 11100 L: b'FILE-NUMBER' C: 1 R: 18 (UVARI) U: b'' V: None
    CD: 001 11100 L: b'FILE-TYPE' C: 1 R: 19 (IDENT) U: b'' V: None
    CD: 001 11100 L: b'PRODUCT' C: 1 R: 20 (ASCII) U: b'' V: None
    CD: 001 11100 L: b'VERSION' C: 1 R: 20 (ASCII) U: b'' V: None
    CD: 001 11100 L: b'PROGRAMS' C: 1 R: 20 (ASCII) U: b'' V: None
    CD: 001 11100 L: b'CREATION-TIME' C: 1 R: 21 (DTIME) U: b'' V: None
    CD: 001 11100 L: b'ORDER-NUMBER' C: 1 R: 20 (ASCII) U: b'' V: None
    CD: 001 11000 L: b'DESCENT-NUMBER' C: 1 R: 19 (IDENT) U: b'' V: None
    CD: 001 11000 L: b'RUN-NUMBER' C: 1 R: 19 (IDENT) U: b'' V: None
    CD: 001 11100 L: b'WELL-ID' C: 1 R: 20 (ASCII) U: b'' V: None
    CD: 001 11100 L: b'WELL-NAME' C: 1 R: 20 (ASCII) U: b'' V: None
    CD: 001 11100 L: b'FIELD-NAME' C: 1 R: 20 (ASCII) U: b'' V: None
    CD: 001 11100 L: b'PRODUCER-CODE' C: 1 R: 16 (UNORM) U: b'' V: None
    CD: 001 11100 L: b'PRODUCER-NAME' C: 1 R: 20 (ASCII) U: b'' V: None
    CD: 001 11100 L: b'COMPANY' C: 1 R: 20 (ASCII) U: b'' V: None
    CD: 001 11100 L: b'NAME-SPACE-NAME' C: 1 R: 19 (IDENT) U: b'' V: None
    CD: 001 11100 L: b'NAME-SPACE-VERSION' C: 1 R: 18 (UVARI) U: b'' V: None
  Objects [1]:
    OBNAME: O: 2 C: 0 I: b'DLIS_DEFINING_ORIGIN'
      CD: 001 00001 L: b'FILE-ID' C: 1 R: 20 (ASCII) U: b'' V: [b'MSCT_197LTP         ␣
↪                                        ']
      CD: 001 00001 L: b'FILE-SET-NAME' C: 1 R: 19 (IDENT) U: b'' V: [b'FAROE_
↪PETROLEUM/206_05A-3']
      CD: 001 00001 L: b'FILE-SET-NUMBER' C: 1 R: 18 (UVARI) U: b'' V: [41]
      CD: 001 00001 L: b'FILE-NUMBER' C: 1 R: 18 (UVARI) U: b'' V: [167]
      CD: 001 00001 L: b'FILE-TYPE' C: 1 R: 19 (IDENT) U: b'' V: [b'STATION LOG']
      CD: 001 00001 L: b'PRODUCT' C: 1 R: 20 (ASCII) U: b'' V: [b'OP']
      CD: 001 00001 L: b'VERSION' C: 1 R: 20 (ASCII) U: b'' V: [b'19C0-187']
      CD: 001 01001 L: b'PROGRAMS' C: 4 R: 20 (ASCII) U: b'' V: [b'MSCT: Mechanical␣
↪Sidewall Coring Tool', b'SGTP: Scintillation Gamma-Ray - P', b'LEHQT: Logging␣
↪Equipment Head - QT', b'WELLCAD: WellCAD file generator']
      CD: 001 00001 L: b'CREATION-TIME' C: 1 R: 21 (DTIME) U: b'' V: [<<class
↪'TotalDepth.RP66V1.core.RepCode.DateTime'> 2011-08-20 22:48:50.000 DST>]
      CD: 001 00001 L: b'ORDER-NUMBER' C: 1 R: 20 (ASCII) U: b'' V: [b'BSAX-00003
↪                                                                              ␣
↪                              ']
      CD: 001 00001 L: b'DESCENT-NUMBER' C: 1 R: 19 (IDENT) U: b'' V: [b'-1']
```

(continues on next page)

(continued from previous page)

```
      CD: 001 00001 L: b'RUN-NUMBER' C: 1 R: 19 (IDENT) U: b'' V: [b'1']
      CD: 001 00001 L: b'WELL-ID' C: 1 R: 20 (ASCII) U: b'' V: [b'
↪                                                                              ␣
↪                                                                              ␣
↪                        ']
      CD: 001 00001 L: b'WELL-NAME' C: 1 R: 20 (ASCII) U: b'' V: [b'206/05a-3
↪                                                                              ␣
↪                                                                              ␣
↪                          ']
      CD: 001 00001 L: b'FIELD-NAME' C: 1 R: 20 (ASCII) U: b'' V: [b'Fulla
↪                                                                              ␣
↪                                                                              ␣
↪                           ']
      CD: 001 00001 L: b'PRODUCER-CODE' C: 1 R: 16 (UNORM) U: b'' V: [440]
      CD: 001 00001 L: b'PRODUCER-NAME' C: 1 R: 20 (ASCII) U: b'' V: [b'Schlumberger']
      CD: 001 00001 L: b'COMPANY' C: 1 R: 20 (ASCII) U: b'' V: [b'Faroe Petroleum
↪                                                                              ␣
↪                                                                              ␣
↪                        ']
      CD: 001 00001 L: b'NAME-SPACE-NAME' C: 1 R: 19 (IDENT) U: b'' V: [b'SLB']
      CD: 000 00000 L: b'NAME-SPACE-VERSION' C: 1 R: 18 (UVARI) U: b'' V: None
-------------------------------------- END EFLR [1/19] at VR: 0x00000050 LRSH:␣
↪0x000000d0 ----------------------------------------
... Many EFLRs later ...
-------------------------------------- END EFLR [18/19] at VR: 0x0001204c LRSH:␣
↪0x00013014 ----------------------------------------
------------------------------------------------------------ Log Pass ---------------
↪-----------------------------------------------
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ Frame Array [0/2] ^^^^^^^^^^
↪^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
FrameArray: ID: OBNAME: O: 2 C: 0 I: b'2000T' b''
  FrameChannel: OBNAME: O: 2 C: 4 I: b'TIME'           Rc:   2 Co:    1 Un: b'ms'   ␣
↪    Di: [1] b'1 second River Time'
  FrameChannel: OBNAME: O: 2 C: 4 I: b'TDEP'           Rc:   2 Co:    1 Un: b'0.1 in
↪'    Di: [1] b'1 second River Depth'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'TENS_SL'        Rc:   2 Co:    1 Un: b'lbf'  ␣
↪    Di: [1] b'Cable Tension'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'DEPT_SL'        Rc:   2 Co:    1 Un: b'0.1 in
↪'    Di: [1] b'Station logging depth'
X Axis summary (all IFLRs):
Min: 16677259.0 Max: 17597260.0 [b'ms'] Count: 921
X Axis spacing summary:
Min: 1000.0 Max: 1001.0 Mean: 1000.0010869565217 Median: 1000.0
   Normal: 920
Duplicate: 0
  Skipped: 0
     Back: 0
Spacing histogram
     Value [   N]: Relative Frequency
  1000.000 [ 919]:␣
↪*********************************************************************************
  1000.100 [   0]:
  1000.200 [   0]:
  1000.300 [   0]:
  1000.400 [   0]:
  1000.500 [   0]:
  1000.600 [   0]:
  1000.700 [   0]:
  1000.800 [   0]:
  1000.900 [   1]:
Frames [921] from: 16677259.000 to 17597260.000 Interval: 1000.000 b'ms'
```

(continues on next page)

(continued from previous page)

```
Frame spacing: <Slice on length=921 start=0 stop=921 step=1> number of frames: 921␣
↪numpy size: 14,736 bytes
Channel   Size   Absent            Min            Mean     Std.Dev.            Max     ␣
↪  Units     dtype
-------   ----   ------   ------------   ------------   ----------   ------------   --
↪-------   -------
   TIME    921        0   16677259.000   17137260.404   265869.810   17597260.000     ␣
↪  b'ms'   float32
   TDEP    921        0     852606.000     872468.708    17513.899     893302.000   b
↪'0.1 in'   float32
TENS_SL    921        0       1825.000       2145.789      198.506       2594.000     ␣
↪ b'lbf'   float32
DEPT_SL    921        0     852606.000     872467.735    17513.909     893303.000   b
↪'0.1 in'   float32

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ END Frame Array [0/2] ^^^^^^^^^
↪^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ Frame Array [1/2] ^^^^^^^^^^^
↪^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
FrameArray: ID: OBNAME: O: 2 C: 0 I: b'800T' b''
  FrameChannel: OBNAME: O: 2 C: 5 I: b'TIME'            Rc:   2 Co:    1 Un: b'ms'     ␣
↪    Di: [1] b'400 milli-second time channel'
  FrameChannel: OBNAME: O: 2 C: 5 I: b'TDEP'            Rc:   2 Co:    1 Un: b'0.1 in
↪'    Di: [1] b'MSCT depth channel'
  FrameChannel: OBNAME: O: 2 C: 1 I: b'ETIM'            Rc:   2 Co:    1 Un: b's'      ␣
↪    Di: [1] b'Elapsed Logging Time'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'LMVL'            Rc:   2 Co:    1 Un: b'V'      ␣
↪    Di: [1] b'Lower Motor Voltage Limit'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'UMVL'            Rc:   2 Co:    1 Un: b'V'      ␣
↪    Di: [1] b'Upper Motor Voltage Limit'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'CFLA'            Rc:   2 Co:    1 Un: b' '      ␣
↪    Di: [1] b'Coring Flag'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'OCD'             Rc:   2 Co:    1 Un: b'ft'     ␣
↪    Di: [1] b'Observed Core Depth'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'RCMD'            Rc:   2 Co:    1 Un: b'V'      ␣
↪    Di: [1] b'Raw Coring Motor Downhole Voltage'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'RCPP'            Rc:   2 Co:    1 Un: b'in'     ␣
↪    Di: [1] b'Raw Kinematics Piston Position'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'CMRT'            Rc:   2 Co:    1 Un: b'h'      ␣
↪    Di: [1] b'Coring Motor Run Time'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'RCNU'            Rc:   2 Co:    1 Un: b' '      ␣
↪    Di: [1] b'Raw Core Number'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'DCFL'            Rc:   2 Co:    1 Un: b' '      ␣
↪    Di: [1] b'Down Command Flag'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'DFS'             Rc:   2 Co:    1 Un: b' '      ␣
↪    Di: [1] b'Data Full Scale'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'DZER'            Rc:   2 Co:    1 Un: b' '      ␣
↪    Di: [1] b'Data Zero'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'RHMD'            Rc:   2 Co:    1 Un: b'V'      ␣
↪    Di: [1] b'Raw Hydraulic Motor Downhole Voltage'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'HMRT'            Rc:   2 Co:    1 Un: b'h'      ␣
↪    Di: [1] b'Hydraulic Motor Run Time'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'RHV'             Rc:   2 Co:    1 Un: b'V'      ␣
↪    Di: [1] b'Raw Head Voltage'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'RLSW'            Rc:   2 Co:    1 Un: b' '      ␣
↪    Di: [1] b'Raw Limit Switch'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'MNU'             Rc:   2 Co:    1 Un: b' '      ␣
↪    Di: [1] b'Marker Number'
```

(continues on next page)

(continued from previous page)

```
  FrameChannel: OBNAME: O: 2 C: 0 I: b'S1CY'            Rc:   2 Co:    1 Un: b' '
↪    Di: [1] b'Solenoid 1 Cycles'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'S2CY'            Rc:   2 Co:    1 Un: b' '
↪    Di: [1] b'Solenoid 2 Cycles'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'RSCU'            Rc:   2 Co:    1 Un: b' '
↪    Di: [1] b'Raw Solenoid Current'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'RSTS'            Rc:   2 Co:    1 Un: b' '
↪    Di: [1] b'Raw Solenoid Status'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'UCFL'            Rc:   2 Co:    1 Un: b' '
↪    Di: [1] b'Up Command Flag'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'CARC'            Rc:   2 Co:    1 Un: b'mA'
↪    Di: [1] b'Cartridge Current'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'CMDV'            Rc:   2 Co:    1 Un: b'V'
↪    Di: [1] b'Coring Motor Downhole Voltage'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'CMPP'            Rc:   2 Co:    1 Un: b'in'
↪    Di: [1] b'Kinematics Piston Position'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'CNU'             Rc:   2 Co:    1 Un: b' '
↪    Di: [1] b'Core Number'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'HMDV'            Rc:   2 Co:    1 Un: b'V'
↪    Di: [1] b'Hydraulic Motor Downhole Voltage'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'HV'              Rc:   2 Co:    1 Un: b'V'
↪    Di: [1] b'Head Voltage'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'LSWI'            Rc:   2 Co:    1 Un: b' '
↪    Di: [1] b'Limit Switch'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'SCUR'            Rc:   2 Co:    1 Un: b' '
↪    Di: [1] b'Solenoid Current'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'SSTA'            Rc:   2 Co:    1 Un: b' '
↪    Di: [1] b'Solenoid Status'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'RCMP'            Rc:   2 Co:    1 Un: b'psi'
↪    Di: [1] b'Raw Coring Motor Pressure'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'RHPP'            Rc:   2 Co:    1 Un: b'psi'
↪    Di: [1] b'Raw Hydraulic Pump Pressure'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'RRPP'            Rc:   2 Co:    1 Un: b'psi'
↪    Di: [1] b'Raw Kinematics Pressure'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'CMPR'            Rc:   2 Co:    1 Un: b'psi'
↪    Di: [1] b'Coring Motor Pressure'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'HPPR'            Rc:   2 Co:    1 Un: b'psi'
↪    Di: [1] b'Hydraulic Pump Pressure'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'RPPV'            Rc:   2 Co:    1 Un: b'psi'
↪    Di: [1] b'Kinematics Pressure'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'SMSC'            Rc:  14 Co:    1 Un: b' '
↪    Di: [1] b'MSCT Status Word'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'CMCU'            Rc:   2 Co:    1 Un: b'mA'
↪    Di: [1] b'Coring Motor Current'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'HMCU'            Rc:   2 Co:    1 Un: b'mA'
↪    Di: [1] b'Hydrailic Motor Current'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'CMLP'            Rc:   2 Co:    1 Un: b'in'
↪    Di: [1] b'Coring Motor Linear Position'
X Axis summary (all IFLRs):
Min: 16677259.0 Max: 17597260.0 [b'ms'] Count: 2301
X Axis spacing summary:
Min: 400.0 Max: 401.0 Mean: 400.0004347826087 Median: 400.0
   Normal: 2300
Duplicate: 0
  Skipped: 0
     Back: 0
Spacing histogram
```

(continues on next page)

(continued from previous page)

```
     Value [    N]: Relative Frequency
   400.000 [ 2299]:␣
↪*********************************************************************************
   400.100 [    0]:
   400.200 [    0]:
   400.300 [    0]:
   400.400 [    0]:
   400.500 [    0]:
   400.600 [    0]:
   400.700 [    0]:
   400.800 [    0]:
   400.900 [    1]:
Frames [2301] from: 16677259.000 to 17597260.000 Interval: 400.000 b'ms'
Frame spacing: <Slice on length=2301 start=0 stop=2301 step=1> number of frames: 2301␣
↪numpy size: 395,772 bytes
Channel   Size   Absent            Min            Mean     Std.Dev.            Max    ␣
↪  Units     dtype
-------   ----   ------   ------------   ------------   ----------   ------------   --
↪-------   -------
   TIME   2301        0   16677259.000   17137261.698   265696.737   17597260.000    ␣
↪  b'ms'   float32
   TDEP   2301        0     852606.000     872468.805    17512.407     893304.000   b
↪'0.1 in'   float32
   ETIM   2301        0          0.000        460.001      265.697        920.001    ␣
↪   b's'   float32
   LMVL   2301        0        585.000        585.000        0.000        585.000    ␣
↪   b'V'   float32
   UMVL   2301        0        635.000        635.000        0.000        635.000    ␣
↪   b'V'   float32
   CFLA   2301        0          0.000         13.361        5.757         18.000    ␣
↪   b' '   float32
    OCD   2301        0       6789.050       7153.751      165.517       7433.008    ␣
↪  b'ft'   float32
   RCMD   2301        0          0.000        191.060      305.260        704.275    ␣
↪   b'V'   float32
   RCPP   2301        0          0.443          0.853        0.649          2.598    ␣
↪  b'in'   float32
   CMRT   2301        0          0.637          0.676        0.018          0.708    ␣
↪   b'h'   float32
   RCNU   2301        0         20.000         20.756        0.532         22.000    ␣
↪   b' '   float32
   DCFL   2301        0          0.000          1.229       12.818        143.000    ␣
↪   b' '   float32
    DFS   2301        0        209.000        209.464        0.499        210.000    ␣
↪   b' '   float32
   DZER   2301        0          0.000          0.002        0.042          1.000    ␣
↪   b' '   float32
   RHMD   2301        0          0.000        345.934      320.677        674.725    ␣
↪   b'V'   float32
   HMRT   2301        0          1.490          1.563        0.032          1.628    ␣
↪   b'h'   float32
    RHV   2301        0        142.319        151.464        1.880        159.428    ␣
↪   b'V'   float32
   RLSW   2301        0          0.000          0.377        0.485          1.000    ␣
↪   b' '   float32
    MNU   2301        0         24.000         24.757        0.533         26.000    ␣
↪   b' '   float32
```

(continues on next page)

(continued from previous page)

```
   S1CY   2301        0         24.000        25.240        0.479        26.000    ␣
↪   b' '   float32
   S2CY   2301        0         27.000        28.939        0.788        30.000    ␣
↪   b' '   float32
   RSCU   2301        0         21.000        74.272       62.645       174.000    ␣
↪   b' '   float32
   RSTS   2301        0          0.000         0.707        0.882         2.000    ␣
↪   b' '   float32
   UCFL   2301        0        128.000       132.961        6.559       143.000    ␣
↪   b' '   float32
   CARC   2301        0        178.238       201.121       12.822       211.238    ␣
↪  b'mA'   float32
   CMDV   2301        0          0.000       191.060      305.260       704.275    ␣
↪   b'V'   float32
   CMPP   2301        0         -0.004         0.407        0.651         2.158    ␣
↪  b'in'   float32
    CNU   2301        0         20.000        20.756        0.532        22.000    ␣
↪   b' '   float32
   HMDV   2301        0          0.000       345.934      320.677       674.725    ␣
↪   b'V'   float32
     HV   2301        0        142.319       151.464        1.880       159.428    ␣
↪   b'V'   float32
   LSWI   2301        0          0.000         0.377        0.485         1.000    ␣
↪   b' '   float32
   SCUR   2301        0         21.000        74.272       62.645       174.000    ␣
↪   b' '   float32
   SSTA   2301        0          0.000         0.707        0.882         2.000    ␣
↪   b' '   float32
   RCMP   2301        0         14.696       149.036      215.905       574.505    ␣
↪ b'psi'   float32
   RHPP   2301        0         14.696      1427.014     1451.455      4201.299    ␣
↪ b'psi'   float32
   RRPP   2301        0         14.696      1434.264     1145.652      4009.911    ␣
↪ b'psi'   float32
   CMPR   2301        0         14.696       149.036      215.905       574.505    ␣
↪ b'psi'   float32
   HPPR   2301        0         14.696      1427.014     1451.455      4201.299    ␣
↪ b'psi'   float32
   RPPV   2301        0         14.696      1434.264     1145.652      4009.911    ␣
↪ b'psi'   float32
   SMSC   2301        0            192       212.597       27.112           254    ␣
↪   b' '     int32
   CMCU   2301        0        -53.000      1059.832     1610.049      8295.000    ␣
↪  b'mA'   float32
   HMCU   2301        0         10.000       339.616      302.494       747.125    ␣
↪  b'mA'   float32
   CMLP   2301        0         -0.927        -0.296        1.043         2.891    ␣
↪  b'in'   float32

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ END Frame Array [1/2] ^^^^^^^^^
↪^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
----------------------------------------------------------- END Log Pass -------------
↪----------------------------------------------
====================================================== END Logical File [0/1]␣
↪======================================================
*************************************************** END RP66V1 File Data Summary␣
↪***************************************************
```

(continues on next page)

(continued from previous page)

```
        540,372         -1    0.750  -0.000%   1456.0 False "example_data/RP66V1/206_
↪05a-_3_DWL_DWL_WIRE_258276498.DLIS"
Execution time =    0.751 (S)
Processed 1 files and 540,372 bytes, 1456.5 ms/Mb
```

1.6 Importing TotalDepth into your Python Project

1.6.1 Processing Western Atlas BIT Files

Reading a BIT File Log Data

Suppose that we have a file in `~/tmp/BIT.bit` and you want to read the frame data from a particular log pass, first import what we need:

```
>>> from TotalDepth.BIT import ReadBIT
>>> import os
```

Then read the BIT file:

```
>>> fpath = os.path.expanduser('~/tmp/BIT.bit')
>>> frame_arrays = ReadBIT.create_bit_frame_array_from_path(fpath)
>>> frame_arrays
[<TotalDepth.BIT.ReadBIT.BITFrameArray object at 0x10f7041c0>, <TotalDepth.BIT.
↪ReadBIT.BITFrameArray object at 0x10f6d6880>]
```

In this file there is are two frame arrays, we can get the description of the first one using `long_str()`:

```
>>> print(frame_arrays[0].long_str())
BITFrameArray: ident="0"
   Unknown head: b'\x00\x02\x00\x00'
    Description: b'SHELL EXPRO U.K.      24 OCT 84      MANSFIELD/DODDS
↪      '
      Unknown A: b'\x00\n\x00\x18\x00'
      Unknown B: b'T  2 9 / 1 0 - 3
↪         '
      Unknown C: b'\x00\x12\x00\x0b\x00\x06  '
  Channels [10]: ['COND', 'SN  ', 'SP  ', 'GR  ', 'CAL ', 'TEN ', 'SPD ', 'ACQ ', 'AC
↪ ', 'RT  ']
   BIT Log Pass: LogPassRange(depth_from=14950.000891089492, depth_to=14590.
↪000869631818, spacing=0.2500000149011621, unknown_a=0.0, unknown_b=16.
↪000000953674373)
   Unknown tail: b'MN239J 1'
    Frame count: 1472
    Frame array:       FrameArray: ID: 0 b'SHELL EXPRO U.K.      24 OCT 84
↪MANSFIELD/DODDS                    '
        <FrameChannel: 'X    ' "Computed X-axis" units: 'b''' count: 1 dimensions: (1,
↪) frames: 1472>
        <FrameChannel: 'COND' "COND" units: 'b''' count: 1 dimensions: (1,) frames:
↪1472>
        <FrameChannel: 'SN   ' "SN   " units: 'b''' count: 1 dimensions: (1,) frames:
↪1472>
        <FrameChannel: 'SP   ' "SP   " units: 'b''' count: 1 dimensions: (1,) frames:
↪1472>
```

(continues on next page)

(continued from previous page)

```
        <FrameChannel: 'GR   ' "GR   " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1472>
        <FrameChannel: 'CAL  ' "CAL  " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1472>
        <FrameChannel: 'TEN  ' "TEN  " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1472>
        <FrameChannel: 'SPD  ' "SPD  " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1472>
        <FrameChannel: 'ACQ  ' "ACQ  " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1472>
        <FrameChannel: 'AC   ' "AC   " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1472>
        <FrameChannel: 'RT   ' "RT   " units: 'b''' count: 1 dimensions: (1,) frames:␣
↪1472>
```

To get the actual vales in the frame we can access the numpy array directly, either by channel ordinal or by name (spaces are significant):

```
>>> sp = frame_arrays[0].frame_array['SP   '].array
array([[-2.49709030e+02],
       [-2.49709030e+02],
       [-2.49709030e+02],
       ...,
       [ 9.99999962e-05],
       [ 9.99999962e-05],
       [ 9.99999962e-05]])
```

Now, if you are familiar with numpy then all normal operations are possible, for example get the min:

```
>>> sp.min()
-2.49709030e+02
```

Note: The `LogPass.FrameArray` is universal, but BIT can only represent one value per frame per channel. To access that value you need to use `array[frame_index][0]`.

References:

BITFrameArray: *TotalDepth.BIT.ReadBIT.BITFrameArray*

FrameArray: *TotalDepth.common.LogPass.FrameArray*

1.6.2 Processing LAS Files

Reading a LAS File Log Data

There is some example files in `example_data/LAS/data`, lets read one:

```
from TotalDepth.LAS.core import LASRead

las_file_path = os.path.join('example_data', 'LAS', 'data', 'BASIC_FILE_0_50.las')
las_file = LASRead.LASRead(las_file_path, las_file_path, raise_on_error=False)
# If there is an array section it will be initialised as a LogPass.FrameArray
if las_file.frame_array is not None:
```

(continues on next page)

(continued from previous page)

```
    # Can iterate through the channels...
    for channel in las_file.frame_array.channels:
        # channel.array is a numpy masked array.
        array = channel.array
        print(f'{channel.ident:4} [{channel.units:4}] Shape: {array.shape} Minimum:
↪{array.min():8g}')
```

The output will typically be:

```
DEPT [m   ] Shape: (649, 1) Minimum:   2889.4
TENS [lbs ] Shape: (649, 1) Minimum:  5292.04
ETIM [min ] Shape: (649, 1) Minimum:    0.019
DHTN [lbs ] Shape: (649, 1) Minimum:  2562.92
GR   [api ] Shape: (649, 1) Minimum:   43.201
```

Note: The `LogPass.FrameArray` is universal, but LAS can only represent one value per frame per channel. To access that value you need to use `array[frame_index][0]`.

References:

FrameArray: *TotalDepth.common.LogPass.FrameArray*

Todo: Complete this.

1.6.3 Processing LIS Files

To use TotalDepth:

```
import TotalDepth
```

Important Concepts

Due to the size of data TotalDepth makes extensive use of lazy evaluation and generators, this means that compute power is only used where necessary.

Log Pass

A *Log Pass* represents a typical, independent, logging recording consisting of a format specification plus binary data. The format specification defines how to interpret the binary data (number of channels, name and units for each channel). In the LIS format the formatting record is known as the *Data Format Specification Record* (*DFSR*). A "Repeat Section" plus a "Main Log" are two, distinct, independent Log Passes.

Frame Set

A *Frame Set* is the logging data converted to the platforms internal types (usually C doubles) and stored in memory as a table where each row represents a particular depth (or time) and each column is the output of a specific channel. Each row is often referred to as a *Frame*

Important LIS Concepts

LIS File

A LIS file is a wrapper round the platforms native I/O system.

Reference: *TotalDepth.LIS.core.File.FileRead*

LIS Index

The LIS file format is designed for recording rather than processing. As such it is a sequential self anouncing format where to understand any part of it you have to have processed all of the preceeding data. This becomes very expensive with large files.

To avoid this an index is first created for the file that then allows random access to any part of that file including an individual frame. All access to the file is using this index. This index is very fast to create and its size is typically 1% of the original file size.

Reference: *TotalDepth.LIS.core.FileIndexer.FileIndex*

Reading a LIS File Table Data

Suppose that we have a file in `~/tmp/LIS.lis` and you want to read the well site data. First import what we need:

```
>>> import TotalDepth
>>> from TotalDepth.LIS.core import File
>>> from TotalDepth.LIS.core import FileIndexer
>>> import os
```

Then open the LIS file and create an index from it:

```
>>> fpath = os.path.expanduser('~/tmp/LIS.lis')
>>> lis_file = TotalDepth.LIS.core.File.FileRead(fpath)
>>> lis_index = TotalDepth.LIS.core.FileIndexer.FileIndex(lis_file)
```

There is a method on the index `genAll()` that iterates through all the records, filter this for only table data:

```
>>> from TotalDepth.LIS.core import LogiRec
>>> cons_records = [lr for lr in lis_index.genAll() if lr.lrType in LogiRec.LR_TYPE_
↪TABLE_DATA]
>>> cons_records
[<TotalDepth.LIS.core.FileIndexer.IndexTable object at 0x103ac3dd8>]
```

In this file there is only one table record. Now read it:

```
>>> lis_file.seekLr(cons_records[0].tell)
>>> table = LogiRec.LrTableRead(lis_file)
```

Now we can explore the table:

```
>>> table.desc
'Well site data'
>>> table.value
b'CONS'
>>> table.colLabels()
odict_keys([b'MNEM', b'ALLO', b'PUNI', b'TUNI', b'VALU'])
>>> table.rowLabels()
dict_keys([b'HIDE', b'HID1', b'HID2', b'CN  ', b'WN  ', ..., b'C30 '])
```

Notice all the entries are represented as Python bytes objects (`b'...'`), this is because LIS does not support Unicode. LIS is also a bit shouty.

To get a specific value, say the well name:

```
>>> print(table[b'WN  '][b'VALU'])
CB: type=69 rc=65 size=16 mnem=b'VALU' EngValRc: b'GUSHER'
>>> table[b'WN  '][b'VALU'].value
b'GUSHER'
```

You can index by integer:

```
>>> table[4][0].value
b'WN  '
>>> table[4][4].value
b'GUSHER'
>>> [v.value for v in table[4]]
[b'WN  ', b'ALLO', b'    ', b'    ', b'GROSSENKNETEN Z2']
```

You can index by slice:

```
>>> [v.value for v in table[4][:2]]
[b'WN  ', b'ALLO']
```

To print the whole table there are some generators for this:

```
>>> for row in table.genRows():
...     for col in row.genCells():
...         print(col.value, ' ', end='')
...     print()
...
b'HIDE'  b'ALLO'  b'    '  b'    '  b'MAIN LOG'
b'HID1'  b'ALLO'  b'    '  b'    '  b'RAW DATA'
b'HID2'  b'ALLO'  b'    '  b'    '  b''
b'CN  '  b'ALLO'  b'    '  b'    '  b'BIG COMPANY'
b'WN  '  b'ALLO'  b'    '  b'    '  b'GUSHER'
...
```

Reference: *TotalDepth.LIS.core.LogiRec.LrTable*

Reading a LIS File Log Data

Suppose that we have a file in `~/tmp/LIS.lis` and you want to read the frame data from a particular log pass, first import what we need:

```
>>> import TotalDepth
>>> from TotalDepth.LIS.core import File
>>> from TotalDepth.LIS.core import FileIndexer
>>> import os
```

Then open the LIS file and create an index from it:

```
>>> fpath = os.path.expanduser('~/tmp/LIS.lis')
>>> lis_file = TotalDepth.LIS.core.File.FileRead(fpath)
>>> lis_index = TotalDepth.LIS.core.FileIndexer.FileIndex(lis_file)
```

There is a method on the index `genLogPasses()` that iterates through the log passes, lets get them all:

```
>>> log_passes = list(lis_index.genLogPasses())
>>> print(log_passes)
[<TotalDepth.LIS.core.FileIndexer.IndexLogPass object at 0x103ac3e80>]
```

In this file there is only one log pass, we can get the description of it using `longstr()`:

```
>>> print(log_passes[0].logPass.longStr())
<TotalDepth.LIS.core.LogPass.LogPass object at 0x103ae10b8>:
       DFSR: <TotalDepth.LIS.core.LogiRec.LrDFSRRead object at 0x103ac3eb8>: "Data␣
↪format specification record"
 Frame plan: <TotalDepth.LIS.core.Type01Plan.FrameSetPlan object at 0x103ae10f0>:␣
↪indr=0 frame length=24 channels=6
   Channels: [b'DEPT', b'SP  ', b'SN  ', b'ILD ', b'CILD', b'DT  ']
        RLE: <TotalDepth.LIS.core.Rle.RLEType01 object at 0x103ae1128>: func=None:␣
↪[RLEItemType01: datum=8592 stride=1014 repeat=7 frames=42, RLEItemType01:␣
↪datum=16704 stride=None repeat=0 frames=39]
     X axis: first=2052.983 last=1995.986 frames=375 overall spacing=-0.1524 in␣
↪optical units=b'M   ' (actual units=b'M   ')
  Frame set: None
```

Note the last line `Frame set: None`, this is because the log pass is a lightweight object which does not (yet) contain all the frame data. To read all the frame data from the file we call `setFrameData(LisFile)` on the log pass:

```
>>> log_passes[0].logPass.setFrameSet(lis_file)
```

Note: The call to `setFrameSet()` will raise an Exception if there is no frame data in the Log Pass. The call should be wrapped in a check for frame data:

```
if log_passes[0].logPass.totalFrames > 0:
    log_passes[0].logPass.setFrameSet(lis_file)
    # Your code here...
```

Now the frame set is fully populated:

```
>>> print(list(log_passes[0].logPass.genFrameSetScNameUnit()))
[('DEPT', 'M   '), ('SP  ', 'MV  '), ('SN  ', 'OHMM'), ('ILD ', 'OHMM'), ('CILD',
↪'MMHO'), ('DT  ', 'US/M')]
```

To get the actual vales in the frame we can access the numpy array directly:

```
>>> data = log_passes[0].logPass.frameSet.frames
>>> data
array([[  2.05298340e+03,  -4.54907703e+00,   1.34538269e+00,
          1.26347518e+00,   3.86598633e+02,  -9.99250000e+02],
       [  2.05283105e+03,  -5.13720322e+00,   1.36061692e+00,
          1.29521227e+00,   5.00510803e+02,  -9.99250000e+02],
       [  2.05267871e+03,  -6.66747475e+00,   1.38543439e+00,
          1.45785594e+00,   5.95623291e+02,  -9.99250000e+02],
       ...,
       [  1.99629077e+03,  -9.99250000e+02,  -9.99250000e+02,
         -9.99250000e+02,  -9.99250000e+02,  -9.99250000e+02],
       [  1.99613843e+03,  -9.99250000e+02,  -9.99250000e+02,
         -9.99250000e+02,  -9.99250000e+02,  -9.99250000e+02],
       [  1.99598608e+03,  -9.99250000e+02,  -9.99250000e+02,
         -9.99250000e+02,  -9.99250000e+02,  -9.99250000e+02]])
```

Now, if you are familiar with numpy then all normal operations are possible, for example get the X axis:

```
>>> data[:,0]
array([ 2052.98339844,  2052.83105469,  2052.67871094,  2052.52636719,
        2052.37402344,  2052.22167969,  2052.06933594,  2051.91650391,
        2051.76416016,  2051.61181641,  2051.45947266,  2051.30712891,
        ...
        1996.29077148,  1996.13842773,  1995.98608398])
```

Find the min, mean, max:

```
>>> data.min(axis=0)
array([ 1995.98608398,  -999.25      ,  -999.25      ,  -999.25      ,
        -999.25      ,  -999.25      ])
>>> data.mean(axis=0)
array([ 2024.48480339,  -305.07223682,  -326.84234802,  -324.20620109,
         206.05499658,    28.2555695 ])
>>> data.max(axis=0)
array([  2.05298340e+03,  -7.69491196e-01,   1.98412299e+00,
         2.34852839e+00,   1.75242944e+03,   4.59522583e+02])
```

References:

LogPass: *TotalDepth.LIS.core.LogPass.LogPass*

FrameSet: *TotalDepth.LIS.core.FrameSet.FrameSet*

1.6.4 Processing RP66V1 Files

This describes how you can programatically access the contents of an RP66V1 file with TotalDepth with worked examples. First, here is how TotalDepth represents the structure of a RP66V1 file.

RP66V1 File Structure

A physical RP66V1 file (typically a file on disk) consists of a *RP66V1.Storage Unit Label* followed of any number of *RP66V1.Explicitly Formatted Logical Record* (EFLRs) and any number of *RP66V1.Indirectly Formatted Logical Record* (IFLRs) in any order.

At this level the file conceptually looks like this:

```
Physical File
|-> Storage Unit Label
|-> EFLR
|-> EFLR
|-> IFLR
|-> EFLR
|-> IFLR
|-> EFLR
    ...
```

The EFLRs and IFLRs are organised into one or more any number of *Logical Files*. Specifically a Logical File starts with an `FILE-HEADER` EFLR followed by an `ORIGIN` or `WELL-REFERENCE` EFLR.

A Logical FIle might represent all the data recorded in, say, a "Repeat Section" and Logical Files are independent from each other[1].

This conceptual model for the Logical File delimited RP66V1 file is something like this:

```
Physical File
|-> Storage Unit Label
|-> Logical File[0] "Repeat Section"
    |-> EFLR type FILE-HEADER
    |-> EFLR type ORIGIN
    |-> EFLR
    |-> EFLR
    |-> IFLR
    |-> EFLR
    |-> IFLR
    ...
|-> Logical File[1] "Main Log"
    |-> EFLR type FILE-HEADER
    |-> EFLR type ORIGIN
    |-> EFLR
    ...
```

The collection of IFLRs for any Logical File can be further organised into a single *Log Pass*.

[1] RP66V1 provides a method of collecting together physical files by using the *RP66V1.Storage Unit Label* where the fields *RP66V1.Storage Set Identifier* and *RP66V1.Storage Unit Sequence Number* provide a means of linking physical files. In practice this has not been seen but if this is your use case then the class `TotalDepth.RP66V1.core.pFile.FileRead` can minimally read the Storage Unit Label and its fields.

Log Pass

A *Log Pass* represents a typical, independent, logging recording. A "Repeat Section" and a "Main Log" are two, distinct, independent Log Passes. The *Log Pass* is defined by two EFLRs, a `CHANNEL` that describes all the available measurements for that Logical File and a `FRAME` that describes the sets of these channels that go into a recording, the index and the spacing. The *Log Pass* represents a single data collection run (example: "Repeat Section") and contains one or more *Frame Arrays*.

Frame Array

A *Frame Array* contains a number of frames of channel data converted to the platforms internal types (usually C doubles) and stored in memory as a table. In this table each row represents a particular value on the X axis (typically depth or time) and each column is the output of a specific channel. Each row is often referred to as a *Frame* There can be multiple values for any channel in a frame except for the X axis which has a single value per frame. A value can be an *Absent Value* indicating that no data was recorded for this channel and frame.

Conceptual Model Presented by TotalDepth

By example here is a RP66V1 file that contains two Logical Files representing, say, the "Repeat Section" and the "Main Log". Each of these has a number of EFLRs and a Log Pass that contains two Frame Arrays, one is sampled every inch in depth and the other every six inches. The two Frame Arrays might have different channels.

This model of the physical RP66V1 file will be something like:

```
Physical File
|-> Storage Unit Label
|-> Logical File[0] "Repeat Section"
|   |-> EFLRs
|   |-> ...
|   |-> Log Pass
|       |-> Frame Array at 1 inch spacing with channels A, B, C
|       |-> Frame Array at 6 inch spacing with channels B, D, E
|-> Logical File[1] "Main Log"
    |-> EFLRs
|   |-> ...
    |-> Log Pass
|       |-> Frame Array at 1 inch spacing with channels A, B, C, X
|       |-> Frame Array at 6 inch spacing with channels B, D, E, F
```

This model is exposed with the class *`TotalDepth.RP66V1.core.LogicalFile.LogicalIndex`*. The LogicalIndex contains a sequence of *`TotalDepth.RP66V1.core.LogicalFile.LogicalFile`* objects that allows random access to all parts of the file.

The following examples show how to iterate through a RP66V1 file with the `LogicalIndex` and access the tables and frame data. The code snippets here are all in `example_data/RP66V1/demo_read.py`

Warning: At this version, 0.4.0rc0, these APIs are provisional, not final.

Basic Pattern for Reading RP66V1 Files with TotalDepth

All these examples take the following pattern where a `LogicalIndex` is created as a context manager.

This can be done with a file path as a string:

```
from TotalDepth.RP66V1.core import LogicalFile

with LogicalFile.LogicalIndex(path) as logical_index:
    # Do something
```

Or an open, binary, file:

```
from TotalDepth.RP66V1.core import LogicalFile

with open(path, 'rb') as fobj:
    with LogicalFile.LogicalIndex(fobj) as logical_index:
        # Do something
```

Example Data

There are some example RP66V1 files distributed in `example_data/RP66V1/data`, for example:

```
from TotalDepth.RP66V1.core import LogicalFile

path = os.path.join('example_data', 'RP66V1', 'data', '206_05a-_3_DWL_DWL_WIRE_
↪258276498.DLIS')
with LogicalFile.LogicalIndex(path) as logical_index:
    # Do something
```

There is also some example RP66V1 binary data in the module `tests.unit.RP66V1.core.test_data`, for example:

```
from TotalDepth.RP66V1.core import LogicalFile
from tests.unit.RP66V1.core import test_data

file_object = io.BytesIO(test_data.BASIC_FILE)
with LogicalFile.LogicalIndex(file_object) as logical_index:
    # Do something
```

Inspecting the Logical File

Once a *TotalDepth.RP66V1.core.LogicalFile.LogicalIndex* has been created the *TotalDepth.RP66V1.core.LogicalFile.LogicalFile* object can be accessed, for example:

```
from TotalDepth.RP66V1.core import LogicalFile
from tests.unit.RP66V1.core import test_data

file_object = io.BytesIO(test_data.BASIC_FILE)
with LogicalFile.LogicalIndex(file_object) as logical_index:
    for l, logical_file in enumerate(logical_index.logical_files):
        print(f'LogicalFile [{l}]: {logical_file}')
```

Produces the single Logical File in `BASIC_FILE`:

```
LogicalFile [0]: <TotalDepth.RP66V1.core.LogicalFile.LogicalFile object at
↪0x11ca5ec50>
```

The Logical File object has at least these properties:

Property	Description
`file_header`	The `FILE-HEADER` EFLR that defines the Logical File. See: [RP66V1 Section 5.1 File Header Logical Record (FHLR)].
`origin_logical_record`	The `ORIGIN` EFLR that defines the origin of the Logical Record. See [RP66V1 Section 5.2 Origin Logical Record (OLR)].
`defining_origin`	Returns the Defining Origin of the Logical File. This is the first row of the `ORIGIN` Logical Record. From [RP66V1 Section 5.2.1 Origin Objects]: *"The first Object in the first ORIGIN Set is the Defining Origin for the Logical File in which it is contained, and the corresponding Logical File is called the Origin's Parent File. It is intended that no two Logical Files will ever have Defining Origins with all Attribute Values identical."*
`channel`	The CHANNEL EFLR or None.
`frame`	The FRAME EFLR or None.
`has_log_pass`	True if this Logical File has both a CHANNEL and a FRAME EFLR.

Notes:

- **TotalDepth implements an EFLR as the class:** *`TotalDepth.RP66V1.core.LogicalRecord.EFLR.ExplicitlyFormattedLogicalRecord`*
- **That class has a method, used below, that provides verbose information about the table:** *`TotalDepth.RP66V1.core.LogicalRecord.EFLR.ExplicitlyFormattedLogicalRecord.str_long()`*

Here is an example of accessing all of the above properties for the `BASIC_FILE`:

```
from TotalDepth.RP66V1.core import LogicalFile
from tests.unit.RP66V1.core import test_data

file_object = io.BytesIO(test_data.BASIC_FILE)
with LogicalFile.LogicalIndex(file_object) as logical_index:
    for l, logical_file in enumerate(logical_index.logical_files):
        print(f'***** logical_file.file_header_logical_record.str_long():')
        print(logical_file.file_header_logical_record.str_long())
        print()
        print(f'***** logical_file.origin_logical_record.str_long():')
        print(logical_file.origin_logical_record.str_long())
        print()
        print(f'***** logical_file.defining_origin:')
        print(logical_file.defining_origin)
        print()
        if logical_file.channel is not None:
            print(f'***** logical_file.channel.str_long():')
            print(logical_file.channel.str_long())
            print()
        if logical_file.frame is not None:
            print(f'***** logical_file.frame.str_long():')
            print(logical_file.frame.str_long())
            print()
        print(f'***** logical_file.has_log_pass:')
```

(continues on next page)

(continued from previous page)

```
        print(logical_file.has_log_pass)
        print()
```

Gives:

```
***** logical_file.file_header_logical_record.str_long():
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'FILE-HEADER' name: b''>
  Template [2]:
    CD: 001 10100 L: b'SEQUENCE-NUMBER' C: 1 R: 20 (ASCII) U: b'' V: None
    CD: 001 10100 L: b'ID' C: 1 R: 20 (ASCII) U: b'' V: None
  Objects [1]:
    OBNAME: O: 2 C: 0 I: b'1'
      CD: 001 00001 L: b'SEQUENCE-NUMBER' C: 1 R: 20 (ASCII) U: b'' V: [b'0000000001']
      CD: 001 00001 L: b'ID' C: 1 R: 20 (ASCII) U: b'' V: [b'HES INSITE.1
↪                                        ']

***** logical_file.origin_logical_record.str_long():
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'ORIGIN' name: b''>
  Template [20]:
    CD: 001 11000 L: b'FILE-ID' C: 0 R: 19 (IDENT) U: b'' V: None
    CD: 001 11000 L: b'FILE-SET-NAME' C: 0 R: 19 (IDENT) U: b'' V: None
    CD: 001 11000 L: b'FILE-SET-NUMBER' C: 0 R: 19 (IDENT) U: b'' V: None
    CD: 001 11000 L: b'FILE-NUMBER' C: 0 R: 19 (IDENT) U: b'' V: None
    CD: 001 11000 L: b'FILE-TYPE' C: 0 R: 19 (IDENT) U: b'' V: None
    CD: 001 11000 L: b'PRODUCT' C: 0 R: 19 (IDENT) U: b'' V: None
    CD: 001 11000 L: b'VERSION' C: 0 R: 19 (IDENT) U: b'' V: None
    CD: 001 11000 L: b'PROGRAMS' C: 0 R: 19 (IDENT) U: b'' V: None
    CD: 001 11000 L: b'CREATION-TIME' C: 0 R: 19 (IDENT) U: b'' V: None
    CD: 001 11000 L: b'ORDER-NUMBER' C: 0 R: 19 (IDENT) U: b'' V: None
    CD: 001 11000 L: b'DESCENT-NUMBER' C: 0 R: 19 (IDENT) U: b'' V: None
    CD: 001 11000 L: b'RUN-NUMBER' C: 0 R: 19 (IDENT) U: b'' V: None
    CD: 001 11000 L: b'WELL-ID' C: 0 R: 19 (IDENT) U: b'' V: None
    CD: 001 11000 L: b'WELL-NAME' C: 0 R: 19 (IDENT) U: b'' V: None
    CD: 001 11000 L: b'FIELD-NAME' C: 0 R: 19 (IDENT) U: b'' V: None
    CD: 001 11000 L: b'PRODUCER-CODE' C: 0 R: 19 (IDENT) U: b'' V: None
    CD: 001 11000 L: b'PRODUCER-NAME' C: 0 R: 19 (IDENT) U: b'' V: None
    CD: 001 11000 L: b'COMPANY' C: 0 R: 19 (IDENT) U: b'' V: None
    CD: 001 11000 L: b'NAME-SPACE-NAME' C: 0 R: 19 (IDENT) U: b'' V: None
    CD: 001 11000 L: b'NAME-SPACE-VERSION' C: 0 R: 19 (IDENT) U: b'' V: None
  Objects [1]:
    OBNAME: O: 2 C: 0 I: b'0'
      CD: 001 01101 L: b'FILE-ID' C: 1 R: 20 (ASCII) U: b'' V: [b'HES INSITE.1']
      CD: 001 01101 L: b'FILE-SET-NAME' C: 1 R: 19 (IDENT) U: b'' V: [b'BURU ENERGY␣
↪LIMITED/VALHALLA NORTH 1']
      CD: 001 01101 L: b'FILE-SET-NUMBER' C: 1 R: 18 (UVARI) U: b'' V: [257346645]
      CD: 001 01101 L: b'FILE-NUMBER' C: 1 R: 18 (UVARI) U: b'' V: [1]
      CD: 001 01101 L: b'FILE-TYPE' C: 1 R: 19 (IDENT) U: b'' V: [b'PLAYBACK']
      CD: 001 01101 L: b'PRODUCT' C: 1 R: 20 (ASCII) U: b'' V: [b'HES INSITE']
      CD: 001 01101 L: b'VERSION' C: 1 R: 20 (ASCII) U: b'' V: [b'R5.1.4']
      CD: 000 00000 L: b'PROGRAMS' C: 0 R: 19 (IDENT) U: b'' V: None
      CD: 001 01101 L: b'CREATION-TIME' C: 1 R: 21 (DTIME) U: b'' V: [<<class
↪'TotalDepth.RP66V1.core.RepCode.DateTime'> 2012-03-07 10:00:49.000 STD>]
      CD: 001 01101 L: b'ORDER-NUMBER' C: 1 R: 20 (ASCII) U: b'' V: [b'9262611']
      CD: 000 00000 L: b'DESCENT-NUMBER' C: 0 R: 19 (IDENT) U: b'' V: None
      CD: 000 00000 L: b'RUN-NUMBER' C: 0 R: 19 (IDENT) U: b'' V: None
      CD: 001 01101 L: b'WELL-ID' C: 1 R: 20 (ASCII) U: b'' V: [b'N/A']
      CD: 001 01101 L: b'WELL-NAME' C: 1 R: 20 (ASCII) U: b'' V: [b'VALHALLA NORTH 1']
```

(continues on next page)

(continued from previous page)

```
      CD: 001 01101 L: b'FIELD-NAME' C: 1 R: 20 (ASCII) U: b'' V: [b'VALHALLA']
      CD: 001 01101 L: b'PRODUCER-CODE' C: 1 R: 16 (UNORM) U: b'' V: [280]
      CD: 001 01101 L: b'PRODUCER-NAME' C: 1 R: 20 (ASCII) U: b'' V: [b'Halliburton']
      CD: 001 01101 L: b'COMPANY' C: 1 R: 20 (ASCII) U: b'' V: [b'BURU ENERGY LIMITED
↪']
      CD: 000 00000 L: b'NAME-SPACE-NAME' C: 0 R: 19 (IDENT) U: b'' V: None
      CD: 000 00000 L: b'NAME-SPACE-VERSION' C: 0 R: 19 (IDENT) U: b'' V: None

***** logical_file.defining_origin:
OBNAME: O: 2 C: 0 I: b'0'
  CD: 001 01101 L: b'FILE-ID' C: 1 R: 20 (ASCII) U: b'' V: [b'HES INSITE.1']
  CD: 001 01101 L: b'FILE-SET-NAME' C: 1 R: 19 (IDENT) U: b'' V: [b'BURU ENERGY␣
↪LIMITED/VALHALLA NORTH 1']
  CD: 001 01101 L: b'FILE-SET-NUMBER' C: 1 R: 18 (UVARI) U: b'' V: [257346645]
  CD: 001 01101 L: b'FILE-NUMBER' C: 1 R: 18 (UVARI) U: b'' V: [1]
  CD: 001 01101 L: b'FILE-TYPE' C: 1 R: 19 (IDENT) U: b'' V: [b'PLAYBACK']
  CD: 001 01101 L: b'PRODUCT' C: 1 R: 20 (ASCII) U: b'' V: [b'HES INSITE']
  CD: 001 01101 L: b'VERSION' C: 1 R: 20 (ASCII) U: b'' V: [b'R5.1.4']
  CD: 000 00000 L: b'PROGRAMS' C: 0 R: 19 (IDENT) U: b'' V: None
  CD: 001 01101 L: b'CREATION-TIME' C: 1 R: 21 (DTIME) U: b'' V: [<<class 'TotalDepth.
↪RP66V1.core.RepCode.DateTime'> 2012-03-07 10:00:49.000 STD>]
  CD: 001 01101 L: b'ORDER-NUMBER' C: 1 R: 20 (ASCII) U: b'' V: [b'9262611']
  CD: 000 00000 L: b'DESCENT-NUMBER' C: 0 R: 19 (IDENT) U: b'' V: None
  CD: 000 00000 L: b'RUN-NUMBER' C: 0 R: 19 (IDENT) U: b'' V: None
  CD: 001 01101 L: b'WELL-ID' C: 1 R: 20 (ASCII) U: b'' V: [b'N/A']
  CD: 001 01101 L: b'WELL-NAME' C: 1 R: 20 (ASCII) U: b'' V: [b'VALHALLA NORTH 1']
  CD: 001 01101 L: b'FIELD-NAME' C: 1 R: 20 (ASCII) U: b'' V: [b'VALHALLA']
  CD: 001 01101 L: b'PRODUCER-CODE' C: 1 R: 16 (UNORM) U: b'' V: [280]
  CD: 001 01101 L: b'PRODUCER-NAME' C: 1 R: 20 (ASCII) U: b'' V: [b'Halliburton']
  CD: 001 01101 L: b'COMPANY' C: 1 R: 20 (ASCII) U: b'' V: [b'BURU ENERGY LIMITED']
  CD: 000 00000 L: b'NAME-SPACE-NAME' C: 0 R: 19 (IDENT) U: b'' V: None
  CD: 000 00000 L: b'NAME-SPACE-VERSION' C: 0 R: 19 (IDENT) U: b'' V: None

***** logical_file.channel.str_long():
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'CHANNEL' name: b''>
  Template [8]:
    CD: 001 11100 L: b'LONG-NAME' C: 0 R: 20 (ASCII) U: b'' V: None
    CD: 001 11100 L: b'PROPERTIES' C: 0 R: 19 (IDENT) U: b'' V: None
    CD: 001 11100 L: b'REPRESENTATION-CODE' C: 0 R: 15 (USHORT) U: b'' V: None
    CD: 001 11100 L: b'DIMENSION' C: 0 R: 18 (UVARI) U: b'' V: None
    CD: 001 11100 L: b'ELEMENT-LIMIT' C: 0 R: 18 (UVARI) U: b'' V: None
    CD: 001 11100 L: b'UNITS' C: 0 R: 27 (UNITS) U: b'' V: None
    CD: 001 11100 L: b'AXIS' C: 0 R: 23 (OBNAME) U: b'' V: None
    CD: 001 11100 L: b'SOURCE' C: 0 R: 24 (OBJREF) U: b'' V: None
  Objects [5]:
    OBNAME: O: 2 C: 0 I: b'DEPT'
      CD: 001 01101 L: b'LONG-NAME' C: 1 R: 20 (ASCII) U: b'' V: [b'DEPT/Depth']
      CD: 000 00000 L: b'PROPERTIES' C: 0 R: 19 (IDENT) U: b'' V: None
      CD: 001 01001 L: b'REPRESENTATION-CODE' C: 1 R: 15 (USHORT) U: b'' V: [7]
      CD: 001 01001 L: b'DIMENSION' C: 1 R: 18 (UVARI) U: b'' V: [1]
      CD: 001 01001 L: b'ELEMENT-LIMIT' C: 1 R: 18 (UVARI) U: b'' V: [1]
      CD: 001 01001 L: b'UNITS' C: 1 R: 27 (UNITS) U: b'' V: [b'm']
      CD: 000 00000 L: b'AXIS' C: 0 R: 23 (OBNAME) U: b'' V: None
      CD: 000 00000 L: b'SOURCE' C: 0 R: 24 (OBJREF) U: b'' V: None
    OBNAME: O: 2 C: 0 I: b'TENS'
      CD: 001 01101 L: b'LONG-NAME' C: 1 R: 20 (ASCII) U: b'' V: [b'TENS/Tension']
      CD: 000 00000 L: b'PROPERTIES' C: 0 R: 19 (IDENT) U: b'' V: None
```

(continues on next page)

(continued from previous page)

```
      CD: 001 01001 L: b'REPRESENTATION-CODE' C: 1 R: 15 (USHORT) U: b'' V: [2]
      CD: 001 01001 L: b'DIMENSION' C: 1 R: 18 (UVARI) U: b'' V: [1]
      CD: 001 01001 L: b'ELEMENT-LIMIT' C: 1 R: 18 (UVARI) U: b'' V: [1]
      CD: 001 01001 L: b'UNITS' C: 1 R: 27 (UNITS) U: b'' V: [b'lbs']
      CD: 000 00000 L: b'AXIS' C: 0 R: 23 (OBNAME) U: b'' V: None
      CD: 001 01001 L: b'SOURCE' C: 1 R: 24 (OBJREF) U: b'' V: [ObjectReference(T=b
↪'TOOL', N=ObjectName(O=2, C=0, I=b'DEP'))]
    OBNAME: O: 2 C: 0 I: b'ETIM'
      CD: 001 01101 L: b'LONG-NAME' C: 1 R: 20 (ASCII) U: b'' V: [b'ETIM/Elapsed Time
↪']
      CD: 000 00000 L: b'PROPERTIES' C: 0 R: 19 (IDENT) U: b'' V: None
      CD: 001 01001 L: b'REPRESENTATION-CODE' C: 1 R: 15 (USHORT) U: b'' V: [7]
      CD: 001 01001 L: b'DIMENSION' C: 1 R: 18 (UVARI) U: b'' V: [1]
      CD: 001 01001 L: b'ELEMENT-LIMIT' C: 1 R: 18 (UVARI) U: b'' V: [1]
      CD: 001 01001 L: b'UNITS' C: 1 R: 27 (UNITS) U: b'' V: [b'min']
      CD: 000 00000 L: b'AXIS' C: 0 R: 23 (OBNAME) U: b'' V: None
      CD: 001 01001 L: b'SOURCE' C: 1 R: 24 (OBJREF) U: b'' V: [ObjectReference(T=b
↪'TOOL', N=ObjectName(O=2, C=0, I=b'DEP'))]
    OBNAME: O: 2 C: 0 I: b'DHTN'
      CD: 001 01101 L: b'LONG-NAME' C: 1 R: 20 (ASCII) U: b'' V: [b'DHTN/CH Tension']
      CD: 000 00000 L: b'PROPERTIES' C: 0 R: 19 (IDENT) U: b'' V: None
      CD: 001 01001 L: b'REPRESENTATION-CODE' C: 1 R: 15 (USHORT) U: b'' V: [2]
      CD: 001 01001 L: b'DIMENSION' C: 1 R: 18 (UVARI) U: b'' V: [1]
      CD: 001 01001 L: b'ELEMENT-LIMIT' C: 1 R: 18 (UVARI) U: b'' V: [1]
      CD: 001 01001 L: b'UNITS' C: 1 R: 27 (UNITS) U: b'' V: [b'lbs']
      CD: 000 00000 L: b'AXIS' C: 0 R: 23 (OBNAME) U: b'' V: None
      CD: 001 01001 L: b'SOURCE' C: 1 R: 24 (OBJREF) U: b'' V: [ObjectReference(T=b
↪'TOOL', N=ObjectName(O=2, C=0, I=b'RWCH'))]
    OBNAME: O: 2 C: 0 I: b'GR'
      CD: 001 01101 L: b'LONG-NAME' C: 1 R: 20 (ASCII) U: b'' V: [b'GR/Gamma API']
      CD: 000 00000 L: b'PROPERTIES' C: 0 R: 19 (IDENT) U: b'' V: None
      CD: 001 01001 L: b'REPRESENTATION-CODE' C: 1 R: 15 (USHORT) U: b'' V: [2]
      CD: 001 01001 L: b'DIMENSION' C: 1 R: 18 (UVARI) U: b'' V: [1]
      CD: 001 01001 L: b'ELEMENT-LIMIT' C: 1 R: 18 (UVARI) U: b'' V: [1]
      CD: 001 01001 L: b'UNITS' C: 1 R: 27 (UNITS) U: b'' V: [b'api']
      CD: 000 00000 L: b'AXIS' C: 0 R: 23 (OBNAME) U: b'' V: None
      CD: 001 01001 L: b'SOURCE' C: 1 R: 24 (OBJREF) U: b'' V: [ObjectReference(T=b
↪'TOOL', N=ObjectName(O=2, C=0, I=b'D4TG'))]

***** logical_file.frame.str_long():
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'FRAME' name: b''>
  Template [8]:
    CD: 001 11100 L: b'DESCRIPTION' C: 0 R: 20 (ASCII) U: b'' V: None
    CD: 001 11100 L: b'CHANNELS' C: 0 R: 23 (OBNAME) U: b'' V: None
    CD: 001 11100 L: b'INDEX-TYPE' C: 0 R: 19 (IDENT) U: b'' V: None
    CD: 001 11100 L: b'DIRECTION' C: 0 R: 19 (IDENT) U: b'' V: None
    CD: 001 11100 L: b'SPACING' C: 0 R: 7 (FDOUBL) U: b'' V: None
    CD: 001 11100 L: b'ENCRYPTED' C: 0 R: 15 (USHORT) U: b'' V: None
    CD: 001 11100 L: b'INDEX-MIN' C: 0 R: 7 (FDOUBL) U: b'' V: None
    CD: 001 11100 L: b'INDEX-MAX' C: 0 R: 7 (FDOUBL) U: b'' V: None
  Objects [1]:
    OBNAME: O: 2 C: 0 I: b'50'
      CD: 000 00000 L: b'DESCRIPTION' C: 0 R: 20 (ASCII) U: b'' V: None
      CD: 001 01001 L: b'CHANNELS' C: 5 R: 23 (OBNAME) U: b'' V: [ObjectName(O=2, C=0,
↪ I=b'DEPT'), ObjectName(O=2, C=0, I=b'TENS'), ObjectName(O=2, C=0, I=b'ETIM'),␣
↪ObjectName(O=2, C=0, I=b'DHTN'), ObjectName(O=2, C=0, I=b'GR')]
      CD: 001 01001 L: b'INDEX-TYPE' C: 1 R: 19 (IDENT) U: b'' V: [b'BOREHOLE-DEPTH']
```

(continues on next page)

(continued from previous page)

```
      CD: 001 01001 L: b'DIRECTION' C: 1 R: 19 (IDENT) U: b'' V: [b'INCREASING']
      CD: 001 01111 L: b'SPACING' C: 1 R: 7 (FDOUBL) U: b'm' V: [0.1]
      CD: 000 00000 L: b'ENCRYPTED' C: 0 R: 15 (USHORT) U: b'' V: None
      CD: 000 00000 L: b'INDEX-MIN' C: 0 R: 7 (FDOUBL) U: b'' V: None
      CD: 000 00000 L: b'INDEX-MAX' C: 0 R: 7 (FDOUBL) U: b'' V: None

***** logical_file.has_log_pass:
True
```

More about RP66V1.EFLR Tables

An *RP66V1.Explicitly Formatted Logical Record* is a table of data organised in rows and columns.

```
Table
    Row
        Value
        Value
        ...
    Row
        Value
        Value
        ...
    ...
```

This is implemented by TotalDepth as:

- Table: *TotalDepth.RP66V1.core.LogicalRecord.EFLR.ExplicitlyFormattedLogicalRecord*
- Row is an `Object`: *TotalDepth.RP66V1.core.LogicalRecord.EFLR.Object*
- Value is an `Attribute`: *TotalDepth.RP66V1.core.LogicalRecord.EFLR.Attribute*

Reading EFLR Contents

Each value in a row/column is known as an *RP66V1.Attribute*

This is implemented by *TotalDepth.RP66V1.core.LogicalRecord.EFLR.Attribute* which has the following properties:

Property	Type	Description
`label`	`bytes`	The label identifying the Attribute.
`count`	`int`	The number of the values the Attribute has.
`rep_code`	`int`	The Representation Code of the values of the Attribute.
`units`	`bytes`	The units of the value.
`value`	`list`	The value itself as a list of instances of the Representation Code.

These Attributes are iterable, for example the following code accesses the contents of every `PARAMETER` EFLR:

```
from TotalDepth.RP66V1.core import LogicalFile
from tests.unit.RP66V1.core import test_data

file_object = io.BytesIO(test_data.BASIC_FILE)
with LogicalFile.LogicalIndex(file_object) as logical_index:
```

(continues on next page)

(continued from previous page)

```
    for logical_file in logical_index.logical_files:
        for position, eflr in logical_file.eflrs:
            # eflr is a TotalDepth.RP66V1.core.LogicalRecord.EFLR.
↪ExplicitlyFormattedLogicalRecord
            if eflr.set.type == b'PARAMETER':
                print(eflr)
                for row in eflr.objects:
                    # row is a TotalDepth.RP66V1.core.LogicalRecord.EFLR.Object
                    print(f'    Row: {row.name.I}')
                    for attr in row.attrs:
                        # attr is a TotalDepth.RP66V1.core.LogicalRecord.EFLR.
↪Attribute
                        print(f'        Attr: {attr.label} = {attr.value} ({attr.
↪units})')
```

Will produce something like this (output truncated):

```
<ExplicitlyFormattedLogicalRecord EFLR Set type: b'PARAMETER' name: b''>
    Row: b'LOC'
        Attr: b'LONG-NAME' = [b'LOCATION'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [b"LATITUDE: 18DEG 01' 32.8'' S"] (b'')
    Row: b'SVCO'
        Attr: b'LONG-NAME' = [b'SERVICECONAME'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [b'Halliburton'] (b'')
    Row: b'IQVR'
        Attr: b'LONG-NAME' = [b'WLIQ VERSION'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [b'R3.2.0'] (b'')
    Row: b'STAT'
        Attr: b'LONG-NAME' = [b'STATE NAME'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [b'WA'] (b'')
    Row: b'COUN'
        Attr: b'LONG-NAME' = [b'COUNTRY NAME'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [b'AUSTRALIA'] (b'')
    Row: b'SON'
        Attr: b'LONG-NAME' = [b'JOB NUMBER'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [b'9262611'] (b'')
    Row: b'SECT'
        Attr: b'LONG-NAME' = [b'SECTION'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
```

(continues on next page)

(continued from previous page)

```
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [b'N/A'] (b'')
    Row: b'TOWN'
        Attr: b'LONG-NAME' = [b'TOWNSHIP'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [b'N/A'] (b'')
    Row: b'RANG'
        Attr: b'LONG-NAME' = [b'RANGE'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [b'N/A'] (b'')
    Row: b'APIN'
        Attr: b'LONG-NAME' = [b'API S/N'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [b'N/A'] (b'')
    Row: b'CN'
        Attr: b'LONG-NAME' = [b'CUSTOMER NAME'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [b'BURU ENERGY LIMITED'] (b'')
    Row: b'WN'
        Attr: b'LONG-NAME' = [b'WELL NAME'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [b'VALHALLA NORTH 1'] (b'')
    Row: b'FN'
        Attr: b'LONG-NAME' = [b'FIELD NAME'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [b'VALHALLA'] (b'')
    Row: b'RIG'
        Attr: b'LONG-NAME' = [b'RIG NAME'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [b'ENSIGN RIG #32'] (b'')
    Row: b'PDAT'
        Attr: b'LONG-NAME' = [b'PERMANENT DATUM'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [b'MSL'] (b'')
    Row: b'LMF'
        Attr: b'LONG-NAME' = [b'LOG MEAS FROM'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [b'RT'] (b'')
```

(continues on next page)

(continued from previous page)

```
    Row: b'DMF'
        Attr: b'LONG-NAME' = [b'DRILL MEAS FROM'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [b'RT'] (b'')
    Row: b'FL1'
        Attr: b'LONG-NAME' = [b'LOCATIONLINE1'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [b"LATITUDE: 18DEG 01' 32.8'' S"] (b'')
    Row: b'FL2'
        Attr: b'LONG-NAME' = [b'LOCATIONLINE2'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [b"LONGITUDE: 124DEG 43' 47.1'' E"] (b'')
    Row: b'FL3'
        Attr: b'LONG-NAME' = [b'LOCATIONLINE3'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [b'EASTING: 683112'] (b'')
    Row: b'FL4'
        Attr: b'LONG-NAME' = [b'LOCATIONLINE4'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [b'NORTHING: 8006107'] (b'')
    Row: b'FL5'
        Attr: b'LONG-NAME' = [b'LOCATIONLINE5'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [b'GDA ZONE 51'] (b'')
    Row: b'DATE'
        Attr: b'LONG-NAME' = [b'DATE'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [b'06-Mar-2012'] (b'')
    Row: b'LCC'
        Attr: b'LONG-NAME' = [b'PRODUCER-CODE'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [b'280'] (b'')
    Row: b'EDF'
        Attr: b'LONG-NAME' = [b'DF ELEV'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [114.9000015258789] (b'm')
    Row: b'EPD'
        Attr: b'LONG-NAME' = [b'ELEVATION'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
```

(continues on next page)

(continued from previous page)

```
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [0.0] (b'm')
    Row: b'EGL'
        Attr: b'LONG-NAME' = [b'GL ELEV'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [109.0] (b'm')
    Row: b'GVFD'
        Attr: b'LONG-NAME' = [b'GRAVITY FIELD'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [1.0] (b'g')
    Row: b'EKB'
        Attr: b'LONG-NAME' = [b'KB ELEV'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [114.9000015258789] (b'm')
    Row: b'TVDS'
        Attr: b'LONG-NAME' = [b'TVDSS CORRECTN'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [5.90000057220459] (b'm')
    Row: b'APD'
        Attr: b'LONG-NAME' = [b'DEPTH ABOVE PD'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [5.90000057220459] (b'm')
    Row: b'DDEV'
        Attr: b'LONG-NAME' = [b'MAX INC'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [1.8200000524520874] (b'deg')
    Row: b'DDEG'
        Attr: b'LONG-NAME' = [b'MAX INC DEPTH'] (b'')
        Attr: b'DIMENSION' = [1] (b'')
        Attr: b'AXIS' = None (b'')
        Attr: b'ZONES' = None (b'')
        Attr: b'VALUES' = [2225.169921875] (b'm')
    ...
```

Or the Attributes can be extracted by identity or integer index, for example:

```
file_object = io.BytesIO(test_data.BASIC_FILE)
with LogicalFile.LogicalIndex(file_object) as logical_index:
    for logical_file in logical_index.logical_files:
        for position, eflr in logical_file.eflrs:
            if eflr.set.type == b'PARAMETER':
                print(eflr[0])
                print()
                print(eflr[0][0])
```

Gives:

```
OBNAME: O: 2 C: 0 I: b'LOC'
  CD: 001 01101 L: b'LONG-NAME' C: 1 R: 20 (ASCII) U: b'' V: [b'LOCATION']
  CD: 001 01001 L: b'DIMENSION' C: 1 R: 18 (UVARI) U: b'' V: [1]
  CD: 000 00000 L: b'AXIS' C: 0 R: 23 (OBNAME) U: b'' V: None
  CD: 000 00000 L: b'ZONES' C: 0 R: 23 (OBNAME) U: b'' V: None
  CD: 001 01101 L: b'VALUES' C: 1 R: 20 (ASCII) U: b'' V: [b"LATITUDE: 18DEG 01' 32.8'
↪' S"]

CD: 001 01101 L: b'LONG-NAME' C: 1 R: 20 (ASCII) U: b'' V: [b'LOCATION']
```

Reading the Frame Data and Accessing the `numpy` Arrays

Here is an example of accessing the numpy arrays and using `np.describe()` to describe each array:

```
import numpy as np

from TotalDepth.RP66V1.core import LogicalFile

with LogicalFile.LogicalIndex(path_in) as logical_index:
    for logical_file in logical_index.logical_files:
        if logical_file.has_log_pass:
            for frame_array in logical_file.log_pass:
                print(frame_array)
                frame_count = logical_file.populate_frame_array(frame_array)
                print(
                    f'Loaded {frame_count} frames and {len(frame_array)} channels'
                    f' from {frame_array.ident} using {frame_array.sizeof_array}␣
↪bytes.'
                )
                for channel in frame_array.channels:
                    print(channel)
                    # channel.array is a numpy array
                    np.info(channel.array)
                    print()
```

The output will be:

```
Loaded 921 frames and 4 channels from OBNAME: O: 2 C: 0 I: b'2000T' using 14736 bytes.

FrameChannel: OBNAME: O: 2 C: 4 I: b'TIME'            Rc:   2 Co:    1 Un: b'ms'      ␣
↪  Di: [1] b'1 second River Time'
class:  ndarray
shape:  (921, 1)
strides:  (4, 4)
itemsize:  4
aligned:  True
contiguous:  True
fortran:  True
data pointer: 0x7faa710b6000
byteorder:  little
byteswap:  False
type: float32

FrameChannel: OBNAME: O: 2 C: 4 I: b'TDEP'            Rc:   2 Co:    1 Un: b'0.1 in' ␣
↪  Di: [1] b'1 second River Depth'
```

(continues on next page)

(continued from previous page)

```
class:  ndarray
shape:  (921, 1)
strides:  (4, 4)
itemsize:  4
aligned:  True
contiguous:  True
fortran:  True
data pointer: 0x7faa710b7000
byteorder:  little
byteswap:  False
type: float32

FrameChannel: OBNAME: O: 2 C: 0 I: b'TENS_SL'         Rc:   2 Co:    1 Un: b'lbf'
↪  Di: [1] b'Cable Tension'
class:  ndarray
shape:  (921, 1)
strides:  (4, 4)
itemsize:  4
aligned:  True
contiguous:  True
fortran:  True
data pointer: 0x7fae6c8c2600
byteorder:  little
byteswap:  False
type: float32

FrameChannel: OBNAME: O: 2 C: 0 I: b'DEPT_SL'         Rc:   2 Co:    1 Un: b'0.1 in'
↪  Di: [1] b'Station logging depth'
class:  ndarray
shape:  (921, 1)
strides:  (4, 4)
itemsize:  4
aligned:  True
contiguous:  True
fortran:  True
data pointer: 0x7fae6c8c3600
byteorder:  little
byteswap:  False
type: float32
...
```

Making Calculations on the `numpy` Data

Very similar to the above we can make some calculations using standard `numpy` calls:

```
import numpy as np

from TotalDepth.RP66V1.core import LogicalFile

with LogicalFile.LogicalIndex(path_in) as logical_index:
    for logical_file in logical_index.logical_files:
        if logical_file.has_log_pass:
            for frame_array in logical_file.log_pass:
                print(frame_array)
                frame_count = logical_file.populate_frame_array(frame_array)
```

(continues on next page)

(continued from previous page)

```
                print(
                    f'Loaded {frame_count} frames and {len(frame_array)} channels'
                    f' from {frame_array.ident} using {frame_array.sizeof_array}␣
↪bytes.'
                )
                for channel in frame_array.channels:
                    print(channel.ident, channel.long_name, channel.units)
                    # channel.array is a numpy array
                    print(f'Min: {channel.array.min():12.3f} Max: {channel.array.
↪max():12.3f}')
```

Would give this output:

```
FrameArray: ID: OBNAME: O: 2 C: 0 I: b'2000T' b''
  FrameChannel: OBNAME: O: 2 C: 4 I: b'TIME'           Rc:   2 Co:    1 Un: b'ms'  ␣
↪    Di: [1] b'1 second River Time'
  FrameChannel: OBNAME: O: 2 C: 4 I: b'TDEP'           Rc:   2 Co:    1 Un: b'0.1 in
↪'    Di: [1] b'1 second River Depth'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'TENS_SL'        Rc:   2 Co:    1 Un: b'lbf'  ␣
↪    Di: [1] b'Cable Tension'
  FrameChannel: OBNAME: O: 2 C: 0 I: b'DEPT_SL'        Rc:   2 Co:    1 Un: b'0.1 in
↪'    Di: [1] b'Station logging depth'
Loaded 921 frames and 4 channels from OBNAME: O: 2 C: 0 I: b'2000T' using 14736 bytes.
OBNAME: O: 2 C: 4 I: b'TIME' b'1 second River Time' b'ms'
Min: 16677259.000 Max: 17597260.000
OBNAME: O: 2 C: 4 I: b'TDEP' b'1 second River Depth' b'0.1 in'
Min:   852606.000 Max:   893302.000
OBNAME: O: 2 C: 0 I: b'TENS_SL' b'Cable Tension' b'lbf'
Min:     1825.000 Max:     2594.000
OBNAME: O: 2 C: 0 I: b'DEPT_SL' b'Station logging depth' b'0.1 in'
Min:   852606.000 Max:   893303.000

FrameArray: ID: OBNAME: O: 2 C: 0 I: b'800T' b''
  FrameChannel: OBNAME: O: 2 C: 5 I: b'TIME'           Rc:   2 Co:    1 Un: b'ms'  ␣
↪    Di: [1] b'400 milli-second time channel'
  FrameChannel: OBNAME: O: 2 C: 5 I: b'TDEP'           Rc:   2 Co:    1 Un: b'0.1 in
↪'    Di: [1] b'MSCT depth channel'
  FrameChannel: OBNAME: O: 2 C: 1 I: b'ETIM'           Rc:   2 Co:    1 Un: b's'   ␣
↪    Di: [1] b'Elapsed Logging Time'
  ... Lots more omitted
  FrameChannel: OBNAME: O: 2 C: 0 I: b'CMLP'           Rc:   2 Co:    1 Un: b'in'  ␣
↪    Di: [1] b'Coring Motor Linear Position'
Loaded 2301 frames and 43 channels from OBNAME: O: 2 C: 0 I: b'800T' using 395772␣
↪bytes.
OBNAME: O: 2 C: 5 I: b'TIME' b'400 milli-second time channel' b'ms'
Min: 16677259.000 Max: 17597260.000
OBNAME: O: 2 C: 5 I: b'TDEP' b'MSCT depth channel' b'0.1 in'
Min:   852606.000 Max:   893304.000
OBNAME: O: 2 C: 1 I: b'ETIM' b'Elapsed Logging Time' b's'
Min:        0.000 Max:      920.001
... Lots more omitted
OBNAME: O: 2 C: 0 I: b'CMLP' b'Coring Motor Linear Position' b'in'
Min:       -0.927 Max:        2.891
```

Limiting the Amount of Data Read

The RP66V1 Frame Array can be very large so to make it more manageable the `TotalDepth.RP66V1.core.LogicalFile.LogicalIndex.populate_frame_array()` can take the following, optional, arguments:

- *channels*: A sequence of channel identifiers. Only these channels will be populated into the numpy arrays in the Frame Array. The other channels will have a zero length numpy array. Channel 0, the X axis, will always be populated.
- *frame_slice* to reduce the number of frames that are populated. You can use either of these classes:
  - *`TotalDepth.common.Slice.Slice`* which takes optional start, stop, step values that default to `(0, len(data), 1)`. For example if there are 128 frames available then `Slice(64, None, 2)` would populate every other frame from frame 64 to the end.
  - `TotalDepth.common.Slice.Split` which takes single integer, this is maximum number of frames to be populated and they will be evenly spaced throughout the Frame Array. For example if there are 128 available frames that `Split(8)` would populate each numpy array with every 16th frame producing 8 frames.

For example, adding the two highlighted lines which populates every 64th frame and channels 1 and 2:

```
from TotalDepth.RP66V1.core import LogicalFile
from TotalDepth.common import Slice

with LogicalFile.LogicalIndex(path_in) as logical_index:
    for logical_file in logical_index.logical_files:
        if logical_file.has_log_pass:
            for frame_array in logical_file.log_pass:
                frame_count = logical_file.populate_frame_array(
                    frame_array,
                    frame_slice=Slice.Slice(0, None, 64),
                    channels={frame_array.channels[1].ident, frame_array.channels[2].
↪ident}
                )
                print(
                    f'Loaded {frame_count} frames'
                    f' from {frame_array.ident} using {frame_array.sizeof_array}␣
↪bytes.'
                )
                for channel in frame_array.channels:
                    if len(channel.array):
                        print(channel.ident, channel.long_name, channel.units)
                        print(f'Min: {channel.array.min():12.3f} Max: {channel.array.
↪max():12.3f}')
                print()
```

Gives:

```
Loaded 15 frames from OBNAME: O: 2 C: 0 I: b'2000T' using 180 bytes.
OBNAME: O: 2 C: 4 I: b'TIME' b'1 second River Time' b'ms'
Min: 16677259.000 Max: 17573260.000
OBNAME: O: 2 C: 4 I: b'TDEP' b'1 second River Depth' b'0.1 in'
Min:   852606.000 Max:   892658.062
OBNAME: O: 2 C: 0 I: b'TENS_SL' b'Cable Tension' b'lbf'
Min:     1877.000 Max:     2561.000

Loaded 36 frames from OBNAME: O: 2 C: 0 I: b'800T' using 432 bytes.
```

(continues on next page)

(continued from previous page)

```
OBNAME: O: 2 C: 5 I: b'TIME' b'400 milli-second time channel' b'ms'
Min: 16677259.000 Max: 17573260.000
OBNAME: O: 2 C: 5 I: b'TDEP' b'MSCT depth channel' b'0.1 in'
Min:   852606.000 Max:   893135.188
OBNAME: O: 2 C: 1 I: b'ETIM' b'Elapsed Logging Time' b's'
Min:        0.000 Max:      896.001
```

1.7 Technical Notes

This collect together various technical descriptions of how TotalDepth solves certain problems.

Contents:

1.7.1 Performance of TotalDepth

This technical note presents the results of some actual performance tests on TotalDepth.

Measuring Performance

This describes some principles used in establishing TotalDepth's performance and setting performance targets.

User's Perception of Performance

Users only want to pay for what they get and experienced users have a rough idea of the cost of what they ask the software to do. For example most users would regard these a cheap operations, and would not expect them to take much time. If they did the user is likely to regard the application as 'slow':

- Load file
- Show log header
- Plot small section

Most users would regard these more expensive operations and is more likely to accept that they would take more time.:

- Cross correlate multiple curves
- Dipmeter processing
- Deconvolution

Most users are aware of the size of the data set with which the are operating and appreciate that operations on larger data sets take longer time. Users do not appreciate O(N^2) or worse behaviour. We don't like it either[1].

TotalDepth measures the cost of operations, in:

```
Execution time (ms)
-------------------
Size of input (Mb)
```

[1] It is very easy for software developers to fail to see this kind of behaviour. For example if the time for an operation is: a + b * N + c * N^2 and c << b. If the software test suit tests an insufficiently small size of N then it appears that the operation is O(N). Along comes a user with a large data set and they see O(N^2) behaviour. This (or worse) is quite commonly observed in many software products.

Example of LIS cost

Our measure is ms/Mb of input. 1Mb of LIS data is typically 250,000 values, or, to put this in context 200 feet of 10 curves (6" sampling) is 0.015 Mb. So, just as an example, the cost of plotting such data from a 20Mb file *might* work out as:

Operation	Cost (ms/Mb)	Data Size (Mb)	User's Time (ms)	Notes
Index a 20Mb file	12	20	240	One-off exercise
Reading 200 feet, 10 curves	1500	0.015	22	
Plotting what has been read	4000	0.015	60	
Total	•	•	**322**	

The rest of this tech note describes the performance of reading LIS files in these ms/Mb terms.

Performance Improvements

The low level performance of TotalDepth is pretty good. FrameSet performance is satisfactory. Further improvement is certain for *Indexing Performance Improvements* once the existing C code (in another project) is integrated into this one.

Populating the frame is a costly exercise and the current solution takes this path:

```
File bytes -> Cython convert to C double -> convert to Python float -> insert into a␣
↪numpy array.
```

All this boxing and unboxing is expensive and a faster (but with more code complexity) is to populate the numpy array directly so this all happens in C code:

```
File bytes -> Convert to C double -> copy directly into numpy memory space
```

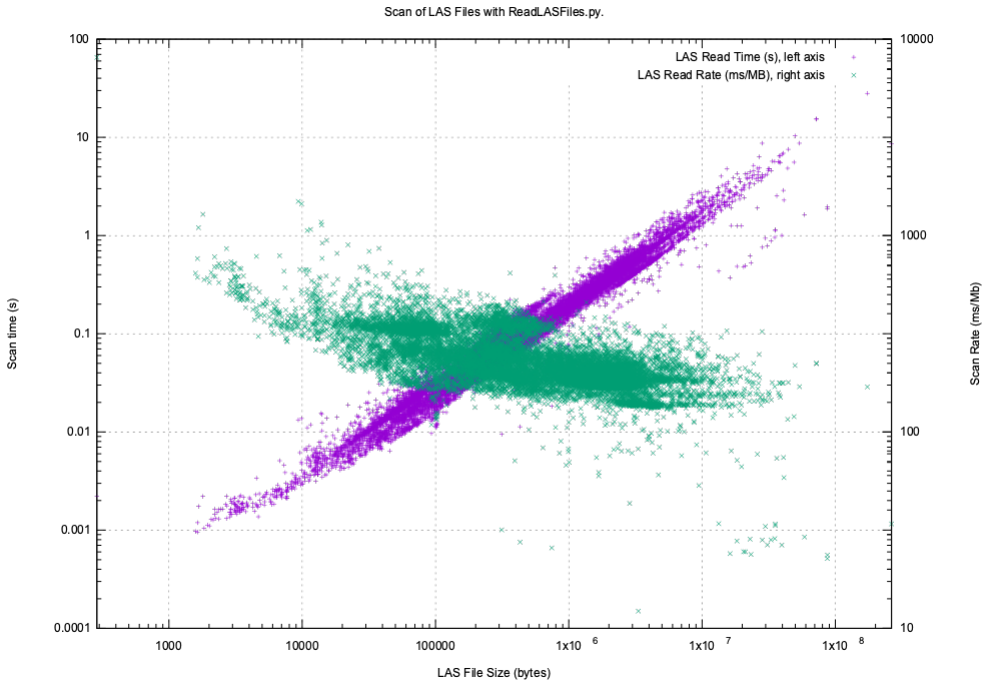
This should provide a great speedup.

The SVG creation is also worth looking at.

1.7.2 LAS Performance

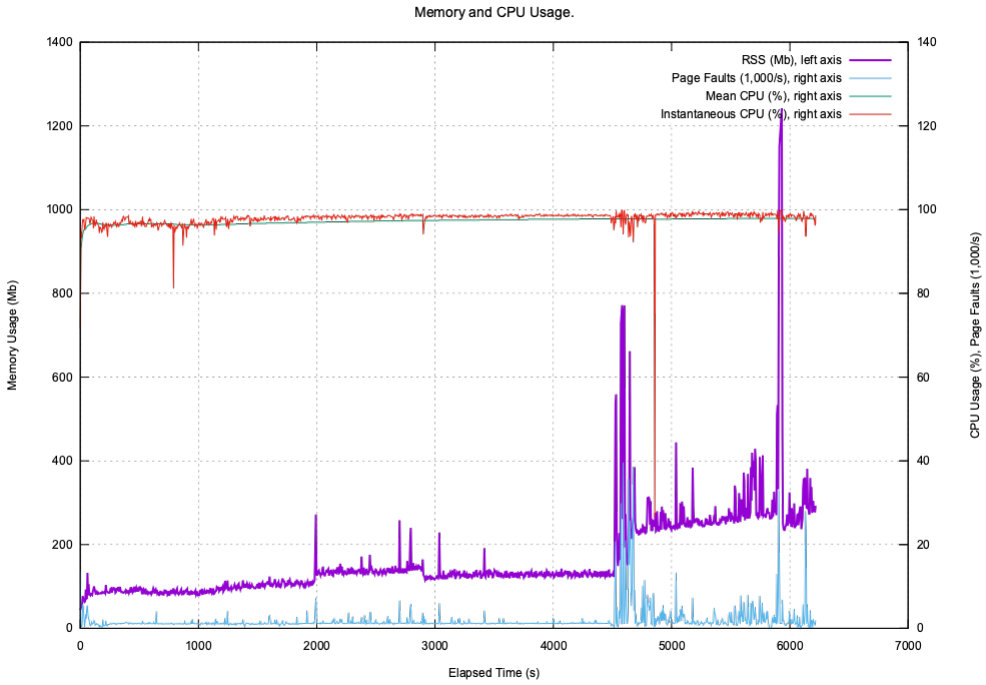
Reading LAS Files

Here is the execution time of reading LAS files with `TotalDepth.LAS.ReadLASFiles` as a single process on 23,000+ real world LAS files totalling 35+Gb, 170+m frames and 2.8+ billion data points. Success rate was >98%. Largest file was 260 Mb:



The overall performance is asymptotic to 180 ms/MB.

The memory usage of doing this follows this pattern:



1.7.3 Indexing LIS Files

Most petrophysical files are recorded in real time and the recording format is sequential, thus everything depends on what has gone before. This is not very satisfactory for the user who might well wish to access the data in a arbitrary manner - "give me these curves over this interval". The solution is to create an index to the original file so that it can be accessed *as if* it is a random access file.

Here we describe how that indexing works and the performance it achieves for the user.

Introduction

The LIS file format is a binary, self describing, sequential format with multiple layers of encoding and, in practice, no forward references. LIS files can be large and, generally speaking, the greater part of the file consists of frame data whose format is invariant within any particular *Log Pass*.

In a dynamic situation, such as a user reading the file, the *instantaneous* amount of data needed from a LIS file is small compared with the file size. For example plotting 200 feet of a 50Mb file might need only 1/3000 of the data in the file. As the LIS file format is geared to sequential recording, not random access, accessing such a small amount efficiently needs additional software cunning.

TotalDepth's approach to this is to use *indexing*. The essential requirements for an indexer are:

- Fast to create an index.
- The index is small.
- The index has sufficiently useful granularity.
- The index can be serialised in a number of ways (as a minimum; XML, binary (e.g. 'pickle'), JSON?).
- The indexer design is flexible and extensible.

Apart from the cost of design and coding a solution the cost/benefit of indexing can be measured thus:

- The time to creating an index.
- The space required by the index, in-memory or serialised in some form.
- The time, in O(N), terms of accessing N bytes of data.

A design that has low time/space requirements is regarded as a 'good' design.

Indexing Design

TotalDepth's LIS indexer works on several levels:

Level	Description
File Level	Indexing of the postion of all Logical Records
Logical Records with frame data	Runs of these can be efficiently indexed with Run Length Encoding
Within a Logical Record containing frame data	Accessing a particular frame and channel can be by computation with the help of a frame index. In particular this can identify seek/read sequences of any value from any channel in O(1) time.

Indexing a LIS File

The intentions is to find the start position of each Logical Record and a minimal amount of information of that Logical Record. The start position is the `size_t` value of the file index of, either, the start of the TIF marker for the first Physical Record in the Logical Record (if TIF encoded), or, the start of the first Physical Record in the Logical Record (if *not* TIF encoded).

As well as recording the file position of the Logical Record the indexer retrieves part or all of the Logical Record contents. This is specialised by Logical Record type thus:

Logical Record	Index Contents
Reel/Tape/File Header/Trailer	The complete contents of the Logical Record. These are small (58 or 128 bytes) but possibly important.
Table records	The Logical Record header and the first Component Block. These are variable, large, but possibly important.
DFSR	The complete contents of the Logical Record. These are variable size, not particularly large and very important.
Logical Records with frame data	Indexed as part of a Log Pass with a RLE object. This records the Logical Record header and the first data word.
All other Logical Records	The Logical Record header only.

Module

The Python module that performs file indexing is `TotalDepth.LIS.core.FileIndexer`

For reference documentation see: *TotalDepth.LIS.core.FileIndexer*.

Example

The LIS package has an Index.py module that will index any LIS file. Here is some (selected) output of a single file:

```
============================== All records ==============================
tell: 0x00000000 type=128 <TotalDepth.LIS.core.FileIndexer.IndexFileHead object at␣
↪0x101b09050>
tell: 0x0000004a type= 34 name=b'CONS' <TotalDepth.LIS.core.FileIndexer.IndexTable␣
↪object at 0x101b090d0>
<TotalDepth.LIS.core.LogPass.LogPass object at 0x101b09450>
tell: 0x0006141c type=129 <TotalDepth.LIS.core.FileIndexer.IndexFileTail object at␣
↪0x101b0c110>
tell: 0x00061466 type=128 <TotalDepth.LIS.core.FileIndexer.IndexFileHead object at␣
↪0x101b0c190>
tell: 0x000614b0 type= 34 name=b'CONS' <TotalDepth.LIS.core.FileIndexer.IndexTable␣
↪object at 0x101b0c210>
<TotalDepth.LIS.core.LogPass.LogPass object at 0x101b0c510>
tell: 0x00065a44 type=129 <TotalDepth.LIS.core.FileIndexer.IndexFileTail object at␣
↪0x101b0cad0>
tell: 0x00065a8e type=128 <TotalDepth.LIS.core.FileIndexer.IndexFileHead object at␣
↪0x101b0cb50>
tell: 0x00065ad8 type= 34 name=b'CONS' <TotalDepth.LIS.core.FileIndexer.IndexTable␣
↪object at 0x101b0cbd0>
<TotalDepth.LIS.core.LogPass.LogPass object at 0x101b0cfd0>
```

(continues on next page)

(continued from previous page)

```
tell: 0x000d2c44 type=129 <TotalDepth.LIS.core.FileIndexer.IndexFileTail object at␣
↪0x101b10450>
============================== All records DONE ==============================
```

Note that the indexing of a Log Pass is separate since it covers an EFLR and zero or more IFLRs. This is the subject of the next section.

Indexing a Log Pass

A Log Pass is described by an EFLR and zero or more IFLRs. In LIS terms this means a DFSR and the associated binary IFLRs. This is a candidate for fairly agressive optimisation since:

- In practice the Logical Records containing the IFLRs are normally adjacent and regular (same size).
- The frame structure is invariant.

Supposing a Log Pass has 18 data channels using representation code 68 (frame length 72 bytes) and each IFLR record contains 24 frames (24*72=1728 bytes). If there are 72 Logical Records then it is fairly easy to to calculate where, in logical space, logical record x is in the file; start + x * 1728[1].

NOTE: The following describes some of the internals of Log Pass indexing, it is for information only as the LogPass and IndexLogPass objects do this automatically.

Run Length Encoding of Logical Records

Run Length Encoding (RLE) is an encoding system that is highly efficient at describing regularly spaced intervals. Since LIS is primarily a recording format the recording software will create and suitably sized buffer for an integer number of frames and other environmental parameters. During recording the buffer will be filled frame by frame. When the buffer is full the buffer will be flushed to file. So a continuos series of IFLRs tends to consist of a series of adjacent Logical Records of the same size followed, possibly, by a single terminating Logical Record that is shorter (by an integer number of frames) than the others.

Module

The Python module that performs RLE of Logical Records is `TotalDepth.LIS.core.Rle`

For reference documentation see: *TotalDepth.LIS.core.Rle (Run Length Encoding)*.

Indexing Frame Data

This describes how an index is created to find arbitrary values in a Logical Record containing Frame Data.

Supposing a Log Pass has 18 data channels using representation code 68 (frame length 72 bytes) and each IFLR record contains 24 frames. If there are 72 Logical Records then it is a simple[2] task to calculate where, in logical space, the bytes for channel x and frame y are.

[1] Illustrative only, it is slightly more complicated than this.
[2] Well not that simple, there is indirect X-axis to take into account and several other things.

Module

The Python module that performs indexing within a Logical Record containing frame data is `TotalDepth.LIS.core.Type01Plan`.

For reference documentation see: *TotalDepth.LIS.core.Type01Plan (Binary Frame Data Random Access)*.

Indexing Performance

As mentioned above the cost of indexing can be measured with these independent measures:

- The time to creating an index.
- The space required by the index, in-memory or serialised in some form.
- The time per byte to access N bytes of data.
- Any deviation from O(N) performance, N being total file size or size of data read.

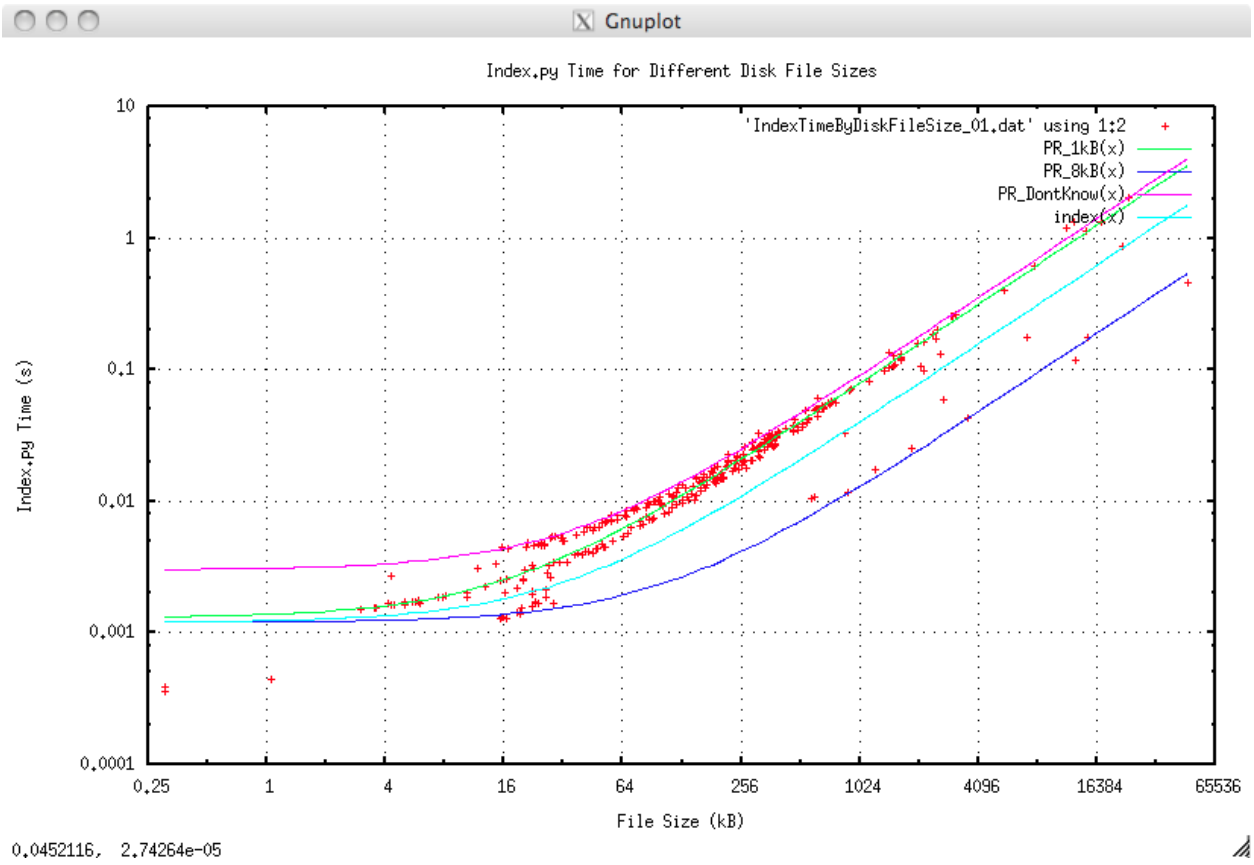
The following data was measured using 300+ LIS files totalling of around 300+Mb, the largest being around 50Mb. The LIS module `TotalDepth.LIS.RandomFrameSetRead.py` was used to conduct the tests.

The original targets for the cost of TotalDepth LIS indexing were:

Cost ms/Mb	Result
<15	Excellent
15-50	Good
50-100	Satisfactory
>100	Unsatisfactory

Index Time

This is simply the actual time taken to create a file level index:



The best fit of all points is the cyan line (index(x) on the plot legend). The best fit for the lines is:

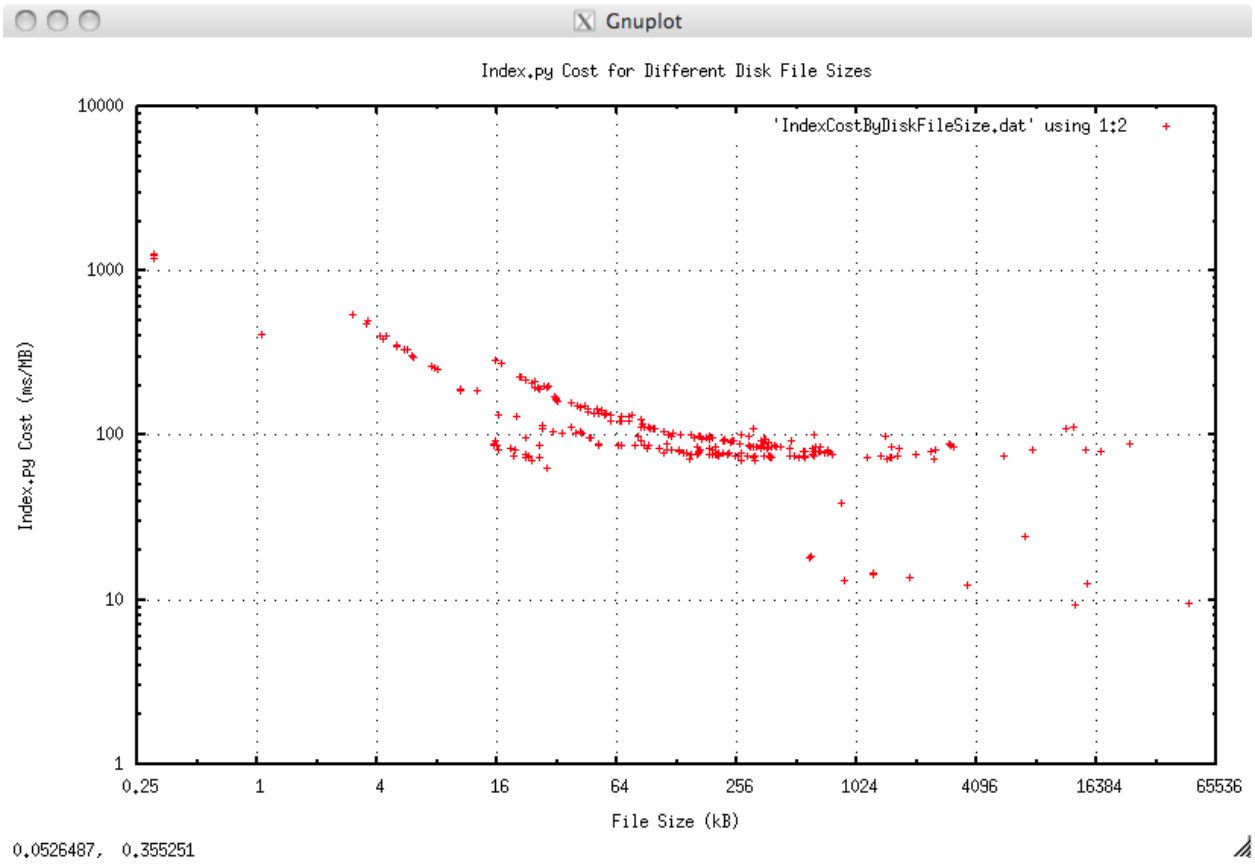
Line	Description	Colour	Latency (ms)	Cost (ms/Mb)	Result
PR_1kB(x)	Physical Record length = 1kb	Green	1.3	80	Satisfactory
PR_8kB(x)	Physical Record length = 8kb	Blue	1.2	12	Excellent
PR_DontKnow	Upper bound, worst case	Magenta	3.0	90	Satisfactory
index(x)	Regression fit on all points	Cyan	1.2	39.9	Good

However there are some other trends that can be teased out when separating files that have Physical Records of 1kb and those in the data set that have 8kb Physical Records (there were no larger Physical Record sizes in the data set). The green fit (PR_1kb(x) on the plot legend) is the best fit for those files that have Physical Records of 1kb. The blue line (PR_8kb(x) on the plot legend) is the best fit for files that have Physical Records of 8kb, it is almost exactly 8 times faster.

In fact in other tests, not presented here, it was shown that there was a linear speedup with Physical Record size up to their maximum size of 64kb. In fact the dominant factor in indexing time was the *number of physical records*. This suggests a couple of things:

- Before archiving or processing LIS files rewrite them to their maximum Physical Record size (64kb). This will pay off later when indexing prior to any read operation.
- Rewrite the Physical Record handler (and below) in C or Cython as it seems to be the bottleneck for indexing.

This is actual time taken to create a file level index divided by the file size. This gives some measure of 'cost' which is defined here as ms per Mb of data processed, here the size in Mb is the total file size:

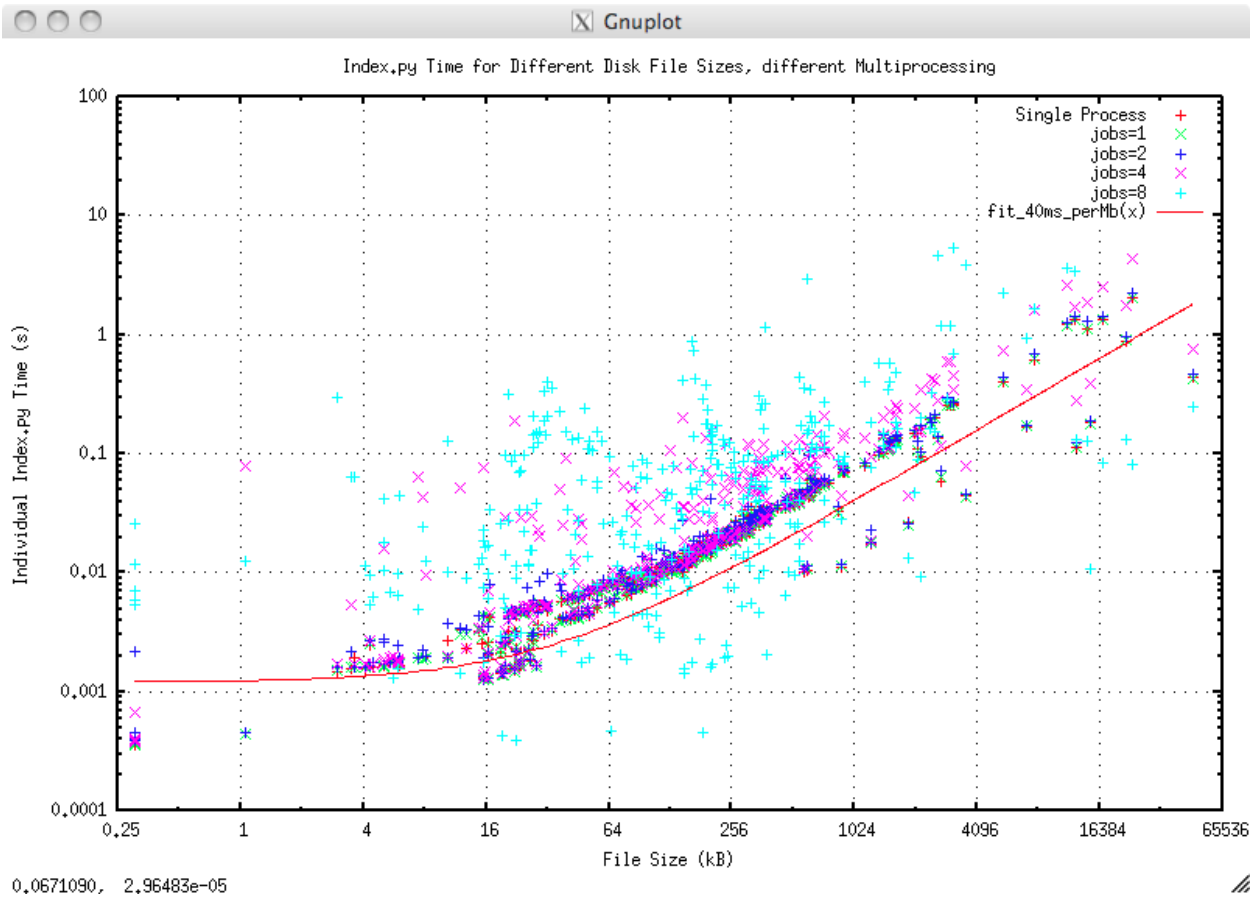


The rising costs for smaller files is no real cause for concern, this represents the startup cost of indexing and reading of around 1 to 2 ms. This graph clearly demonstrates O(N), or better, behaviour.

Again the advantage in having 8kb Physical Records is evident in the lower right hand corner. The cost would be further reduced to around 1 to 1.5 ms/Mb by having 64kb Physical Records.

Multi-Processing

Most of TotalDepth software supports parallel processing. The LIS command line tool Index.py can create indexes in parallel. The following graph shows the file size plotted against total time to index when indexing around 300 LIS files with different numbers of simultaneous processes. The red line is the best fit for single process indexing that costs about 40 ms/Mb:



The index time rises slightly with increasing number of processes and the scatter rises dramatically with 8 processes presumably because of I/O contention.

Actual wall clock times for indexing all those LIS files for different numbers of simultaneous processes on a four core machine with hyper-threading are:

Processes	Total time to index (s)	Improvement over No multiprocessing
No multiprocessing	18.4	Datum
1	18.3	0.995
2	9.9	0.538
4	8.9	0.484
8	7.6	0.413

Moving to two processes gives an almost linear speedup, moving from two to four or eight processes gives only slight improvement presumably because the execution time becomes I/O bound.

Index Size

It is envisaged that the index will be persisted in some form. Once persisted then the LIS file would only be accessed via the index, any file write operation requires a suitable adjustment to the index.

Persistence techniques could be, for example:

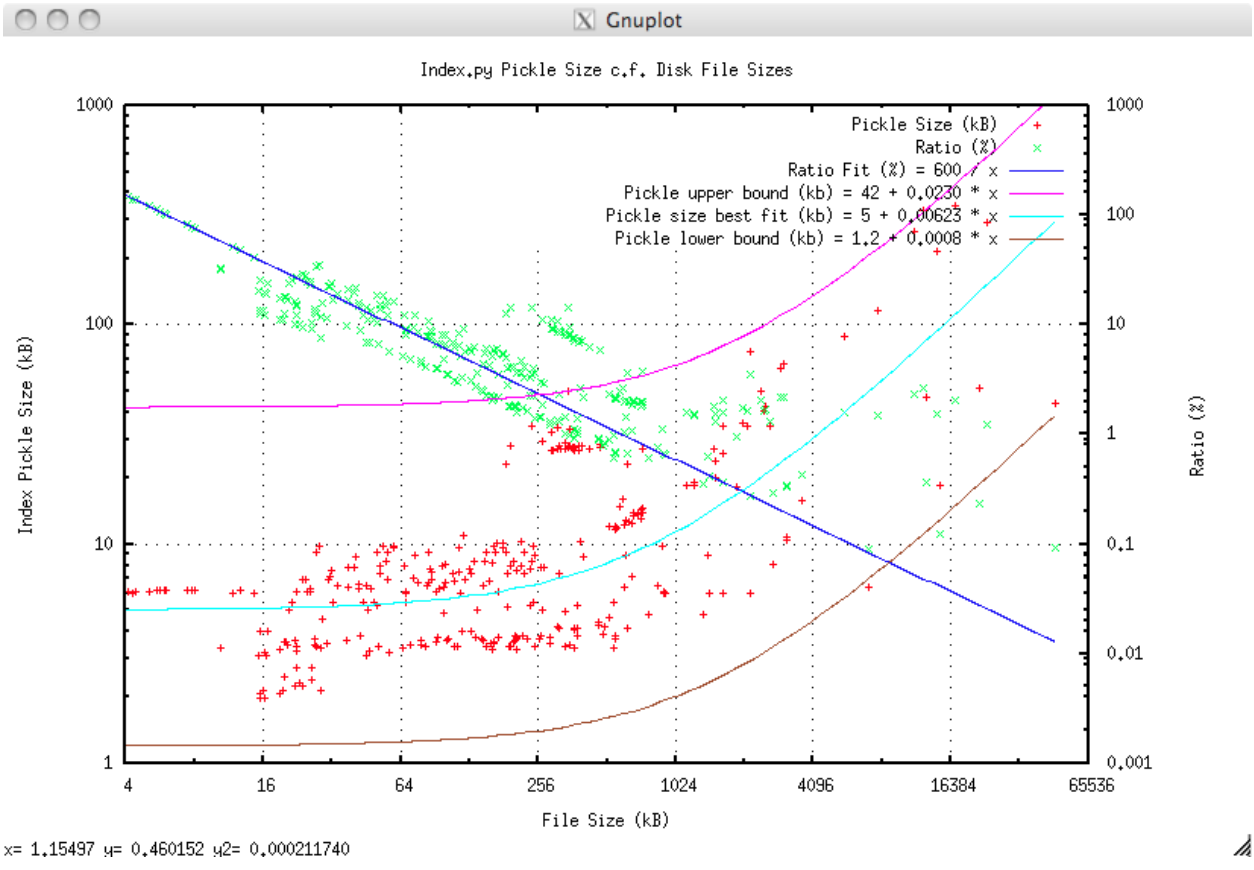
- Stored in a database.
- Serialised in binary form onto the file system.
- Serialised in human readable form, such as XML, onto the file system or database[3].
- Serialised in binary form and attached as a Logical Record to the LIS file it refers to.

TotalDepth and SaaS

There is a further imperative to understanding index size; if TotalDepth were to be used, as it was always intended, as Software as a Service (SaaS) where the bulk of the processing is with the data file on the client machine and the processing done on the server then part of the bootstrap process of any transaction is for the client to index the file and send the index to the server. In that case it is important to keep the index size small.

Thus the size of the index content is a significant consideration.

The following graph measures the size of the index when serialised with Python's `cPickle` module, Pickled size is in kb (red markers, left scale) and compared as a percentage of the original file size (green markers, right scale):



[3] JSON can not handle the index in its current form.

The pickled size shows a wide range that is representative of the wide range of LIS inputs. The best fit for size (albeit with a large scatter) is 5kb + 0.6% of the file size which is entirely satisfactory [the upper bound being 42kb + 2.3% and the lower bound being just 1.2kb + 0.08% with larger files tending towards the lower bound].

The relative size of the index shows a strong downward trend (blue line) for files below 4Mb, before levelling off at the 0.1 to 3% mark. This is quite satisfactory for the use cases described above including SaaS. This graph clearly demonstrates O(N), or better, behaviour.

Indexing Performance Improvements

As noted above there is a substantial improvement in indexing when large Physical Record sizes are used.

It is also likely that significant improvement could be made if the RawStream, TifMarker and PhysRec were to be rewritten in C/C++ or Cython. The PhysRec module has a dependency on the RepCode module (easily removed). All three modules have a dependency on the struct module so the limited functionality that they use from there would have to be reproduced, thats pretty easy since it only involves integer manipulation.

So given an 'average' cost of indexing of 40 ms/Mb (i.e. a 'good' rating) the performance improvements could be:

- Moving to larger Physical Record sizes: x8?
- Integrate the existing (in another project) code in C that handles Raw Stream/TIF/Physical Record handling into this project. This is known to be about x100 faster (and the index has a lower memory footprint).

The performance improvements would not necessarily combine as they are mutually dependent but the combination might reduce the cost to around 2 to 8 ms/MB, an exceptionally good performance indeed.

See *Performance Improvements* for other performance improvements

LIS Read Performance via an Index

This is described here *LIS Performance*

Summary

Indexing is not free, it incurs an overhead, but this overhead is acceptable. The overhead is worst for small data sizes where the performance is high in any case. The overhead is low, and the benefit is very great for large or complex data sizes where the performance, without indexing, could be very poor indeed.

1.7.4 LIS Performance

Actual performance results of some LIS file operations.

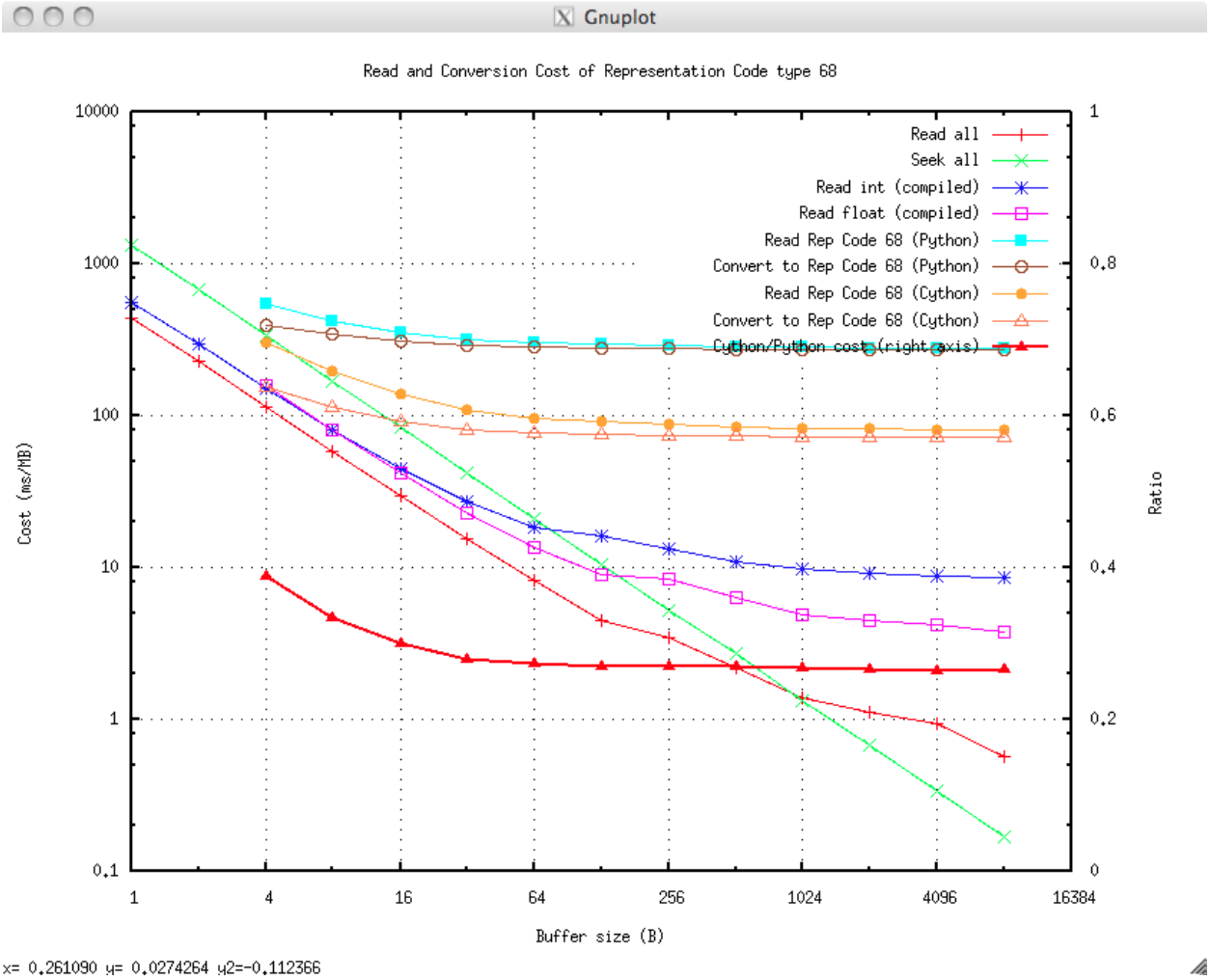
File Read Cost

The measure here is low level reading and conversion from binary words to internal number formats. Three operations are being measured:

- Raw read speed of bytes.
- Above, and converting to internal integers and floats.
- Above and converting from Rep Code 68 to an internal float.

The input data was about 100 binary files with random data of around 100 Mb in total.

The x-axis of the graph below is the buffer length, in other words how much data is swallowed in a single atomic read.



Raw File Read

The 'Read all' graph shows the cost of reading bytes at different buffer sizes from a file and discarding it. This is the baseline and represents the crude storage access time. Large buffer sizes achieve <1 ms/Mb (1 Gb/s).

Raw File seek()

The 'Seek all' graph shows the cost of traversing a file using seek() at different seek lengths. No data is read. Combined with the read information this can be used to estimate the minimum time to make a series of read/seek operations.

Using the `struct` module

The "Read int (compiled)" and "Read float (compiled)" data shows the cost of reading bytes into a buffer and then converting them to a list of integers and floats using the `struct` module (the type declaration is compiled). They show an asymptotic cost, in addition to the read cost, of 8 and 3 ms/Mb respectively. This shows very good performance for reading builtin types with the struct module. As most RP66 types are builtins (or types supported by the struct module) then the struct module should be used for RP66.

It is slightly surprising that reading an integer is slower than reading a floating point number.

Reading Rep Code 68

This establishes the cost of converting a bytes object in Representation Code 68 to a internal floating point value.

The cost of using Python code and Cython code is examined and in each case the the total cost or read+convert is presented and then the integer `struct` read cost (above) is subtracted to get the conversion cost.

Python

The data sets "Read Rep Code 68 (Python)" and "Convert to Rep Code 68 (Python)" plots to the cost of reading bytes and converting them to Representation code 68, the latter curve has the read cost subtracted (from above) so it represents merely the conversion cost of in-memory from bytes to Rep Code 68. This is asymptotic to around 270 ms/Mb.

Cython

The data sets "Read Rep Code 68 (Cython)" and "Convert to Rep Code 68 (Cython)" is as immediately above but using Cython code rather than Python code. This is asymptotic to around 72 ms/Mb. This is significantly longer than using the struct module to interpret a IEEE float so it could be that some improvement could be made with a pure C implementation.

Cython vs Python

The "Cython/Python cost (right axis)" plots the ratio of converting Rep Code 68 with Python and Cython. The Cython code takes 0.27 of the cost of the Python code. Well worth it.

Frame Read Cost

The operation being timed here is, given a LIS file index, read a certain set of frame data then convert each value to an internal floating point number and then populate the internal FrameSet object. This is a basic operation before doing any plotting, channel editing, recalibration etc. The amount of data read is the cumulative size in LIS bytes of all the values read from the file. 250,000 values would constitute typically 1Mb of LIS data.

A number of different methods were use to read the LIS data:

- Method A: All frames all channels. Here indexing provides no optimisation as it is a simple sequential read operation of everything.
- Method C: A sequential subset of the frames and a sequential subset of the channels. This requires a logical seek operation to the start of the frames then, within each frame a seek then read then seek operation.

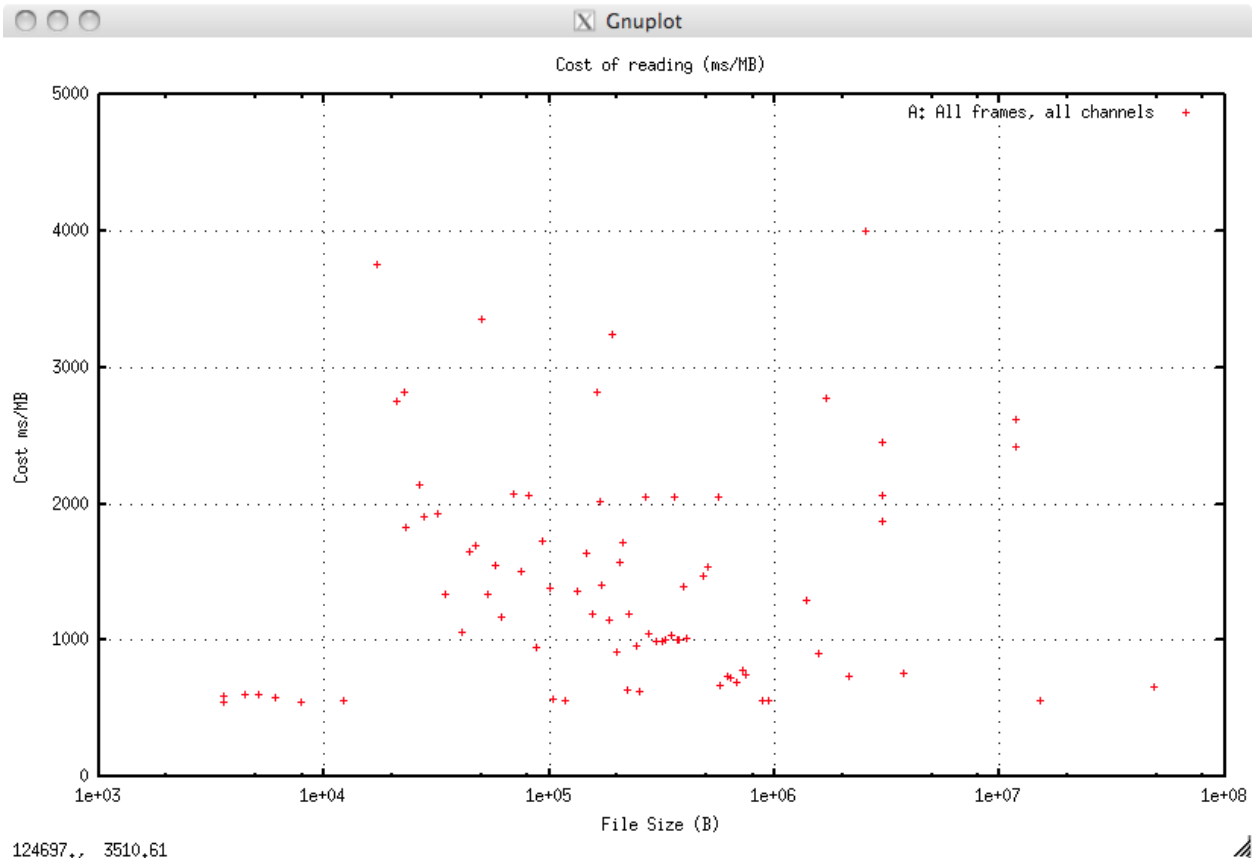
- Method D: A sequential subset of the frames and a non-sequential subset of the channels. This requires a logical seek operation to the start of the frames then, within each frame multiple seek/read operations.
- Method E: A non-sequential subset of the frames and a non-sequential subset of the channels. This requires a set of logical seek/read operations to each frame then, within each frame multiple seek/read operations.

Indexing can provide an optimisation for methods C-E as it hugely reduces the amount of read operations to that just sufficient for the required frames and channels. The important measure is that the cost of indexing, computing and executing seek/read operations does not outweigh the reduction in data read.

For example if a frame has 40 channels and only 5 are required then, if the cost per byte stays the same then the user sees a 8 fold speed improvement. If the cost per byte doubles then the user still sees a 4 fold speed improvement. If the cost per byte rises 8 fold there is no advantage over doing a sequential read of all channels. Beyond an eight fold rise in indexing cost the user sees a performance *reduction*.

A: All Frames, all Channels

This is the baseline, indexing plays no part as this is a straightforward sequential read operation of every channel in every frame.



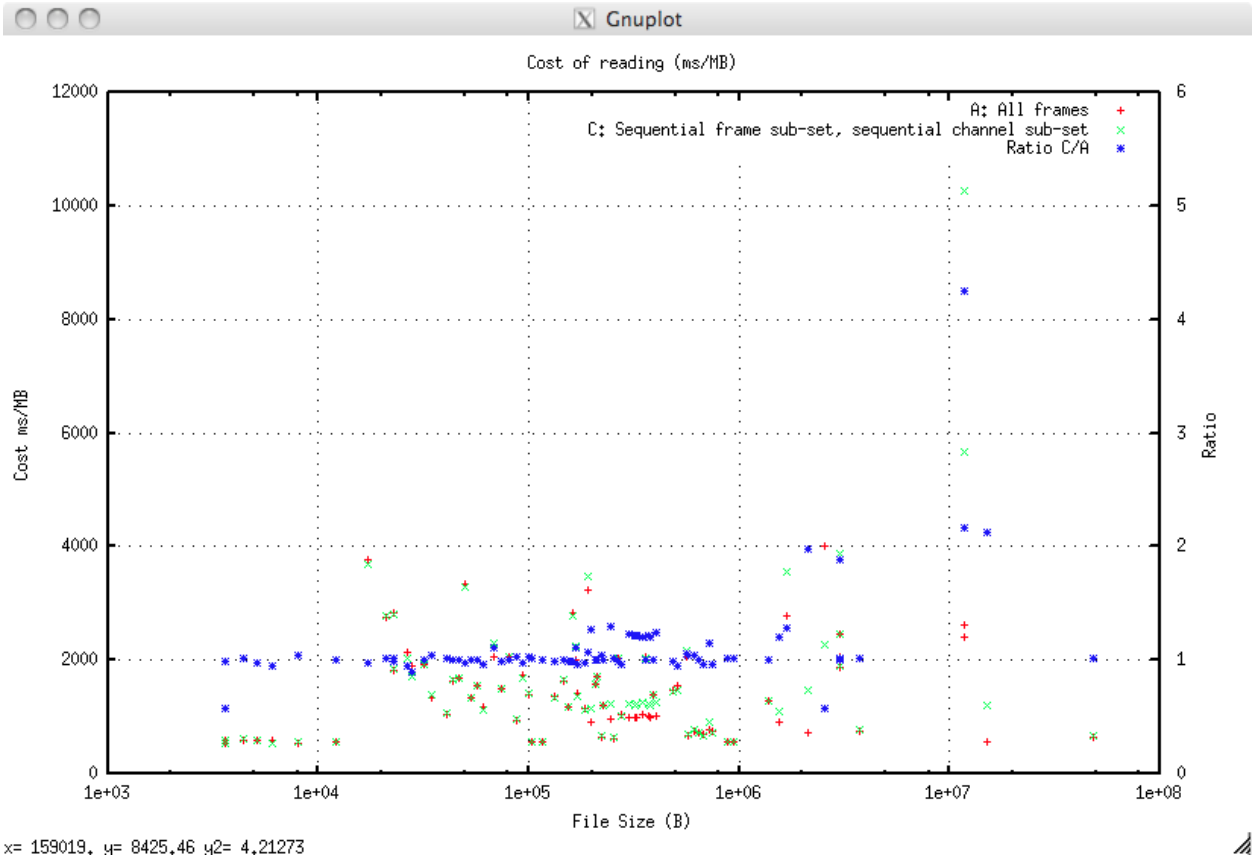
Although there is quite a spread there is, gratifyingly, no obvious trend for the cost to increase as the file size does which would indicate worse than O(N) behaviour. The behaviour is clearly O(N). The cost in ms/Mb is:

A:	Cost ms/Mb
Minimum	541.8
Mean	1446.9
Maximum	4003.3

C: Adjacent Frames and Channels

This is taking a random, but sequential, subset of the frames and a random, but sequential, subset of the channels. This requires the indexer to do a logical seek operation to the start of the frames then, within each frame a seek then read then seek operation.

The results are plotted compared to the baseline and the ratio of the cost of C relative to the cost of A for each file is plotted on the right hand axis.



The cost in ms/Mb is:

Value	A: Cost ms/Mb	C: Cost ms/Mb	Ratio C/A
Minimum	541.8	545.7	0.57
Mean	**1446.9**	**1561.3**	**1.13**
Maximum	4003.3	4485.8	4.25

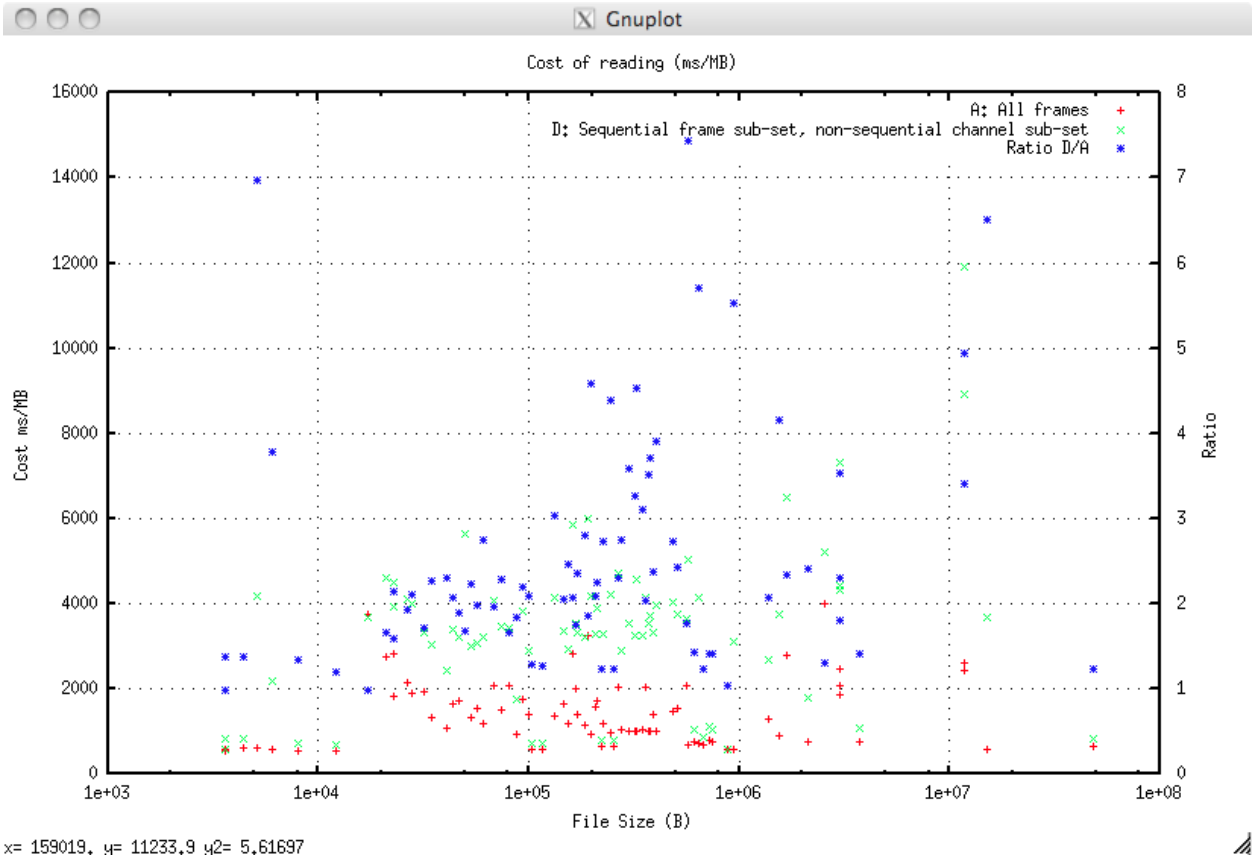
This means that a small increase in cost (13%) is the price of a hugely reduced data set.

D: Adjacent Frames, any Channels

This is taking a random, but sequential, subset of the frames and a random, but non-sequential, subset of the channels. This requires the indexer to do a logical seek operation to the start of the frames then, within each frame multiple seek/read operations.

This is highly representative of the indexing operation performed when plotting, say, any 200 foot section of a number of selected channels.

The results are plotted compared to the baseline and the ratio of the cost of D relative to the cost of A for each file is plotted on the right hand axis.



The cost in ms/Mb is:

Value	A: Cost ms/Mb	D: Cost ms/Mb	Ratio D/A
Minimum	541.8	581.8	0.98
Mean	**1446.9**	**3410.5**	**2.55**
Maximum	4003.3	11928.6	7.44

It should be noted that this test is deliberately harsh in that non-sequential channels are always chosen. If some of the channels are adjacent then the cost reduces, for those channels, more towards C (typically half the cost of D).

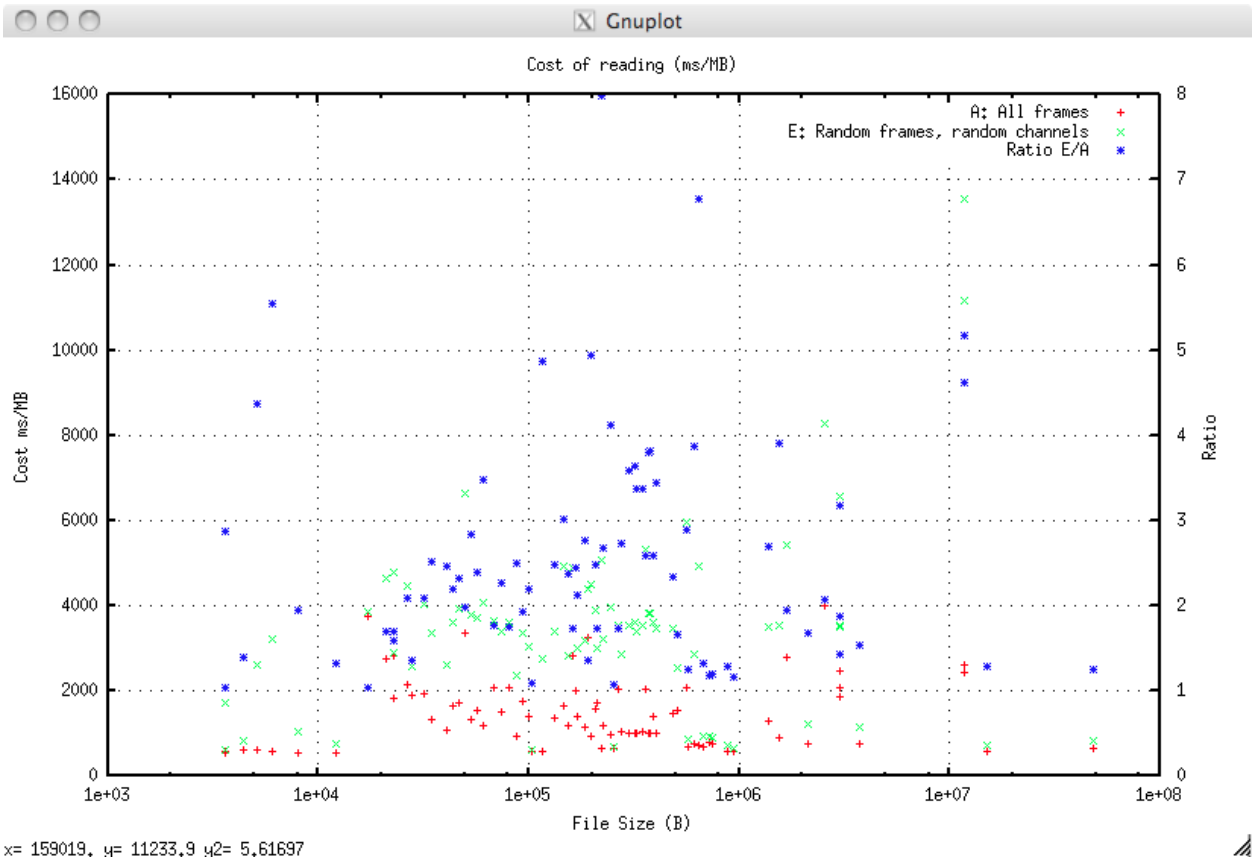
Evens so provided the number of channels of interest is less than 40% of the total indexing this way provides a real speed improvement. The important point being this may not produce an improvement (and may actually be slower) for small frame sizes; they are not the problem precisely because they are small. For the seriously problematic large frame sizes then indexing will always be faster, often dramatically so.

E: Any Frames, any Channels

This is taking a random, non-sequential, subset of the frames and a random, but non-sequential, subset of the channels. This requires the indexer to perform a set of logical seek/read operations to each frame then, within each frame multiple seek/read operations.

Essentially this is the same as test D but skipping intermediate frames.

The results are plotted compared to the baseline and the ratio of the cost of E relative to the cost of A for each file is plotted on the right hand axis.



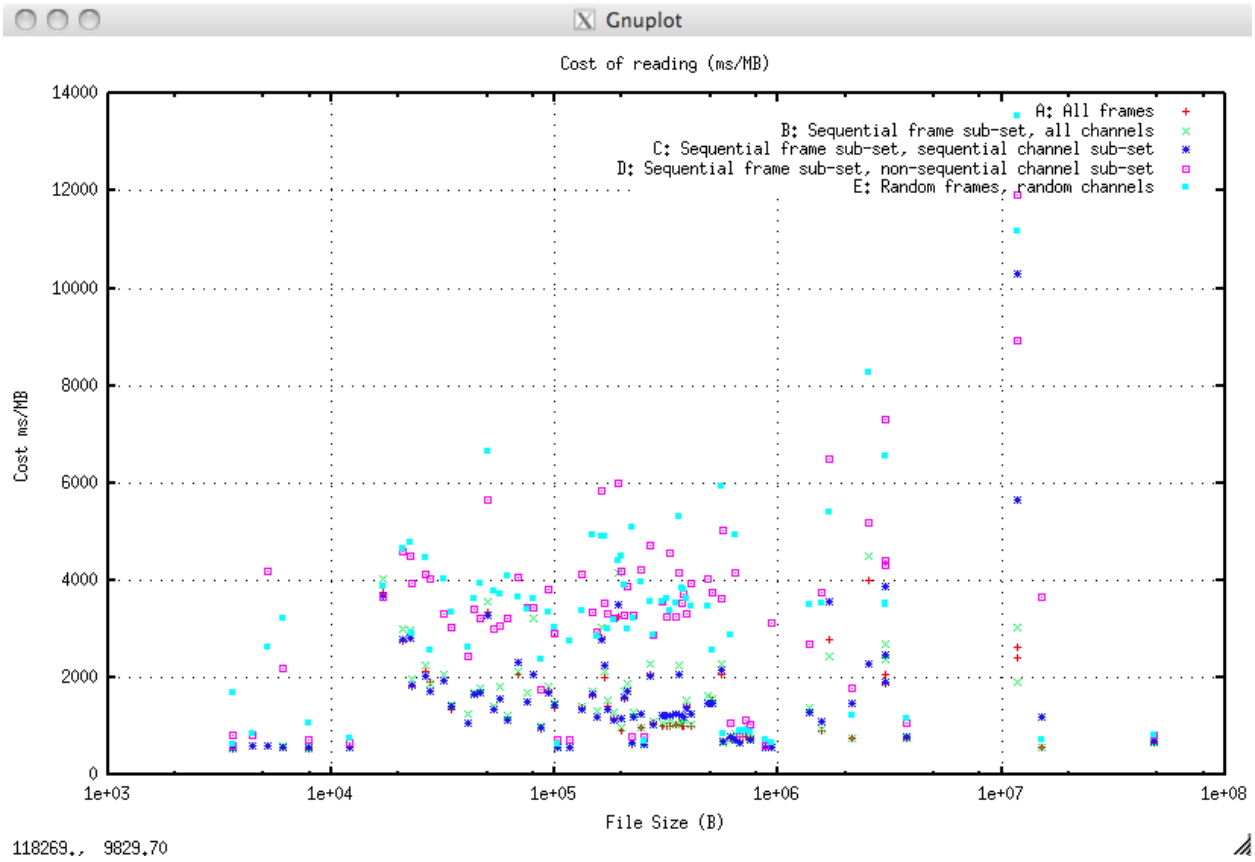
The cost in ms/Mb is:

Value	A: Cost ms/Mb	E: Cost ms/Mb	Ratio E/A
Minimum	541.8	619.9	1.03
Mean	**1446.9**	**3491.7**	**2.57**
Maximum	4003.3	13552.5	7.98

There is a very slight (around 1-5%) additional cost over D but the huge benefit is being able to skip intermediate frames. This means that a low resolution plot (say 1:500) of a high resolution log (say sampling every 1.2 inches) could read every fifth frame (six inch sampling) and that would be plotted every 0.012" (say 1 pixel) as a speedup of almost 500%.

All Measurements

For the sake of completeness, here are all the results:



Performance Improvements

todo:: Complete this

1.7.5 Indexing RP66V1 Files

This describes how RP66V1 binary files are indexed to provide random access to any part of their data structure. This also describes the performance of TotalDepth indexing a test set of around 100+ files that ranged in size from 80kb to 4GB totalling 17GB. The average file size was about 150Mb. These tests were run on a 2.7 GHz Intel Core i7 machine with 4 cores and hyper-threading. This data refers to version 0.3.0 and may not be relavent to the current version, 0.4.0rc0.

Low-Level Index

There are multiple levels of a RP66V1 index in TotalDepth, the lowest being a simple index of the start of every Logical Record. The index is implemented by *TotalDepth.RP66V1.core.pIndex.LogicalRecordIndex*

There is one entry on the index for each Logical Record and the entry consists of:

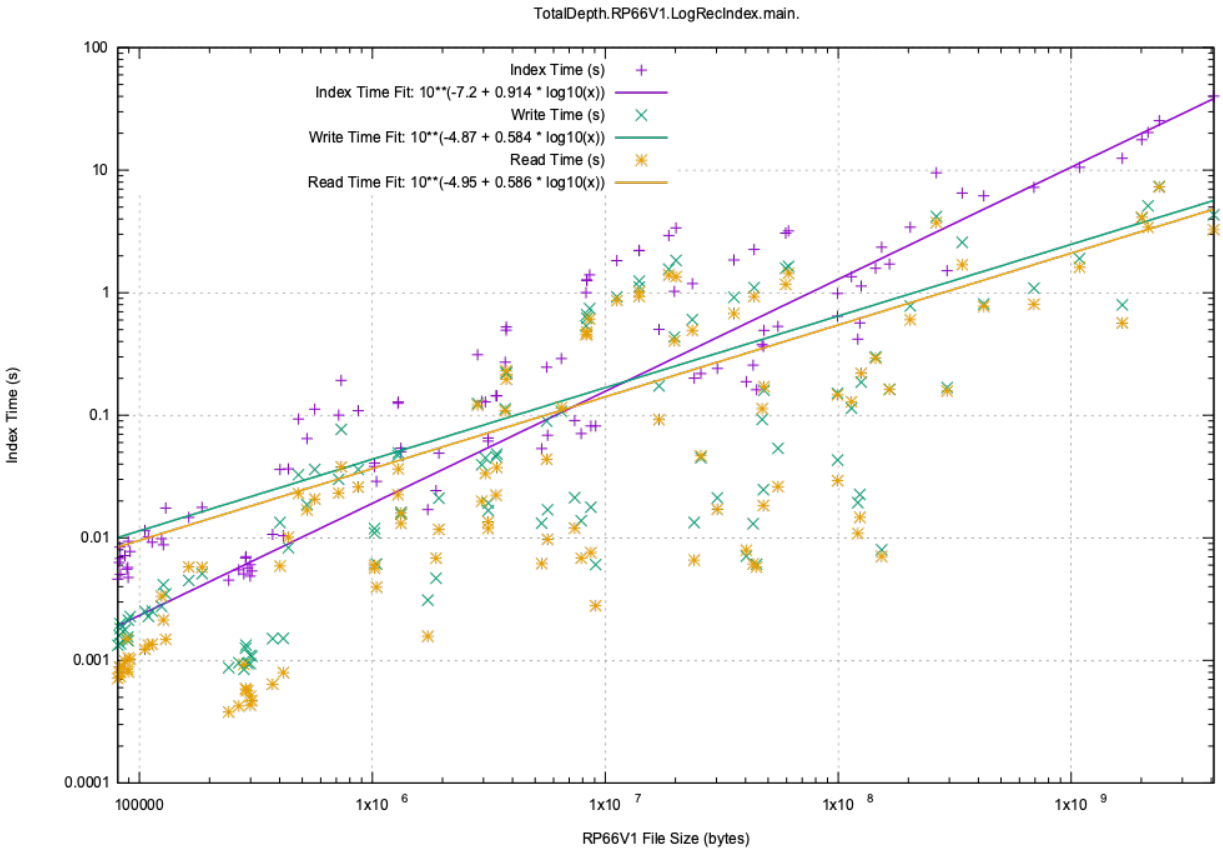
- Integer absolute file position of the immediately prior Visible Record.
- Integer absolute file position of the first Logical Record Segment Header.
- One byte value of the first Logical Record Segment Header attributes.
- One byte value of the first Logical Record Segment Header type.
- Integer length of the Logical Data.

The following performance data was gathered by *TotalDepth.RP66V1.LogRecIndex*. The performance is assessed by:

- The time it takes to index the file, persist it and read it back.
- The size of the persistent index.
- Memory usage.
- The value of multi-processing.

Execution Time

Here is the execution time(s) for creating, persisting and reading back the persisted index compared to RP66V1 files size:

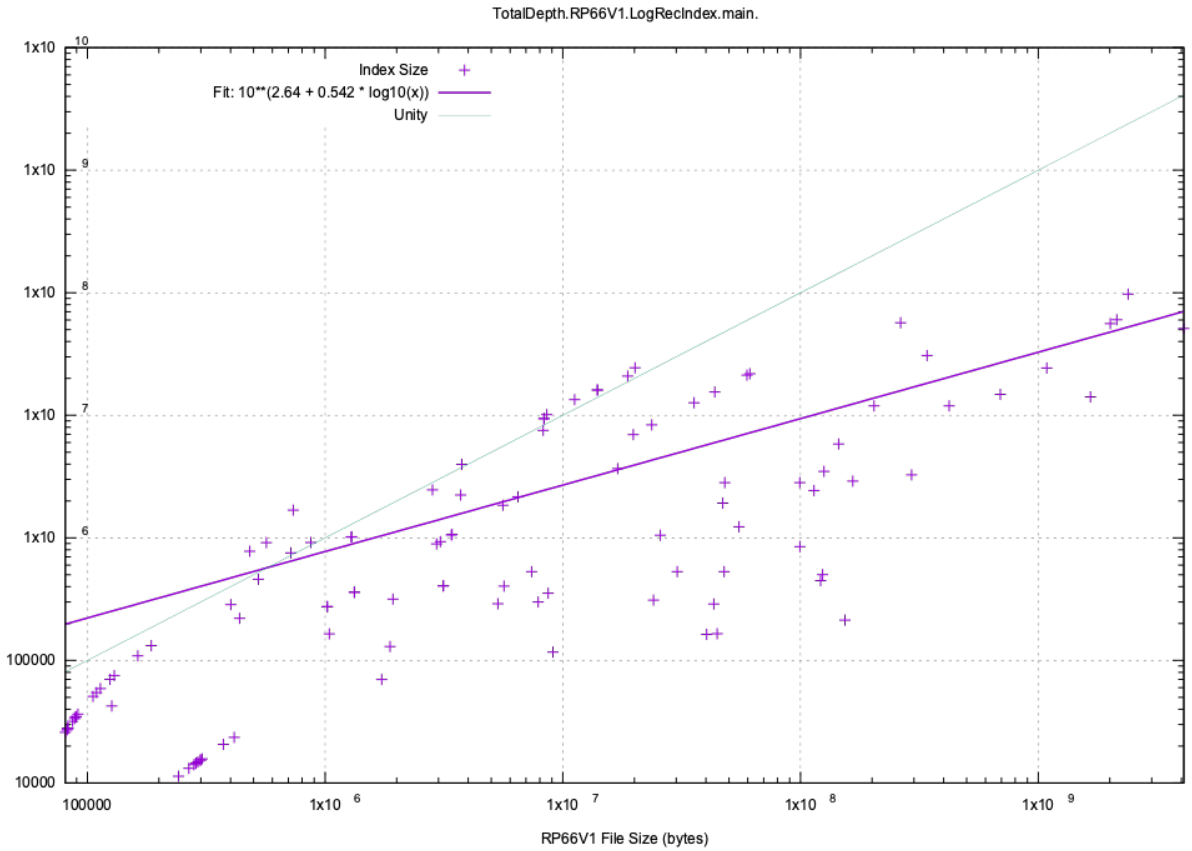


While there is quite a variation by a factor of 10 the *average asymptotic* execution time trends to:

- Create the index: 10.6 ms/Mb
- Persist the index: 2.4 ms/Mb
- Read the persistent index: 2.1 ms/Mb

Index Size

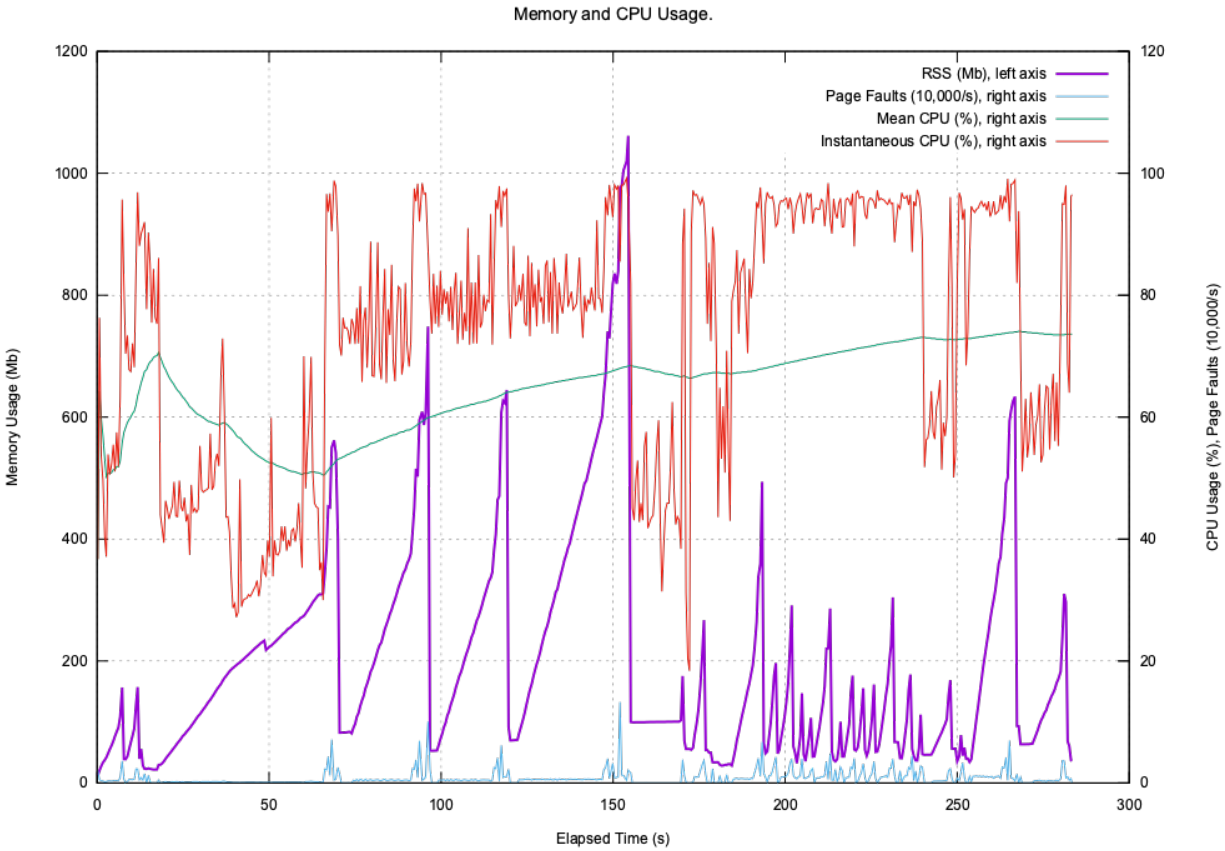
The pickled index size compared to the original file size is shown here. The green line shows where the index size would be equal to the input file size:



This index size trends to around 10% of the file size. The large size of the index reflects to cost of Python's general purpose pickle protocol. Each entry takes about 160 bytes whereas the C/C++ implementation takes only around 24 bytes.

Memory and CPU Usage

Indexing the test set makes this memory and CPU demands:

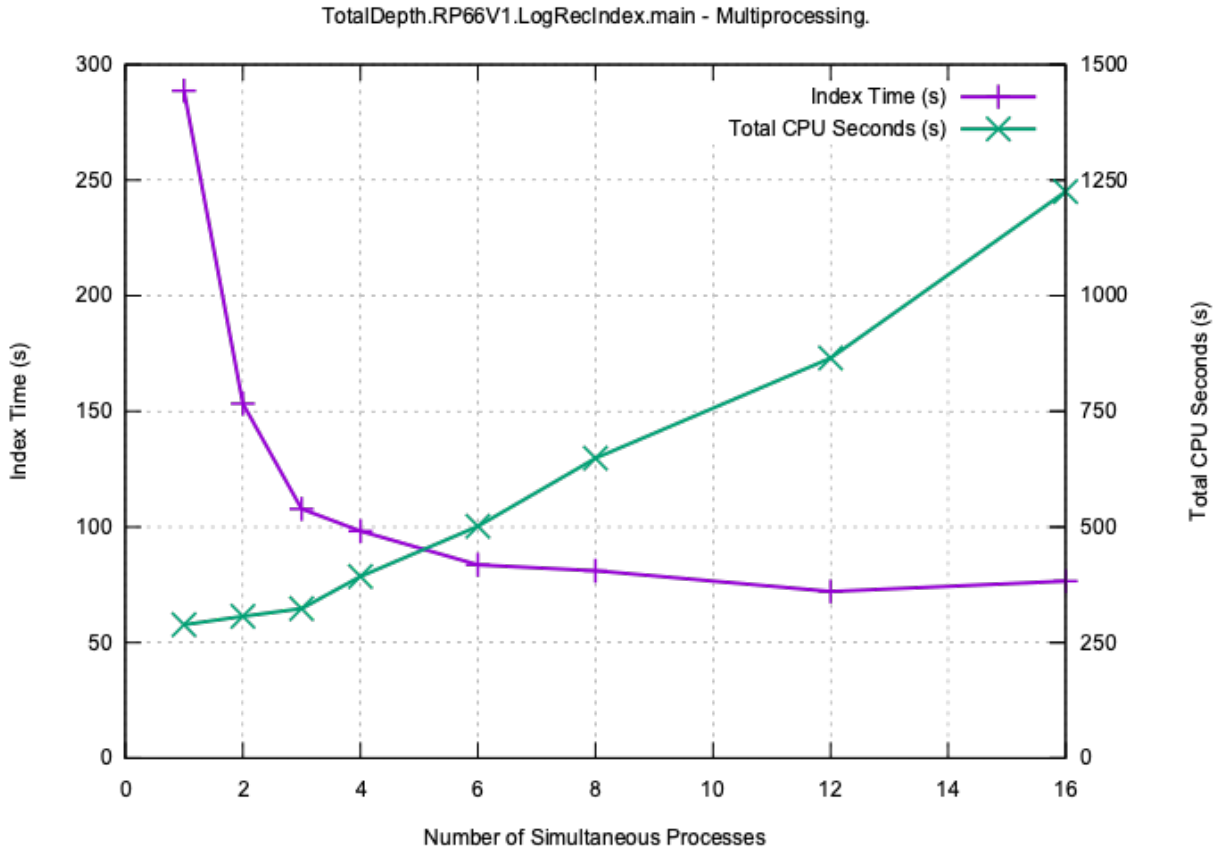


The extra spikes at the end of indexing are due to Python's pickle module that pickles in-memory (even if pickling to a file).

The current version, 0.4.0rc0, this index is implemented in pure Python but in a future release the C/C++ indexer will be used. This is far faster and has a much smaller persistent index.

Multiprocessing

Each index is independent so multi-processing can speed up index creation enormously. The number of simultaneous processes can be specified with the `--jobs=` option. Here is the same test data set being indexed including pickling the index to disk with a different number of simultaneous processes.



As expected beyond four processes the improvement is marginal.

Mid-Level Index

The mid-level index wraps up the low-level index with richer information and is one that the user normally interacts with.

The additional information it contains is:

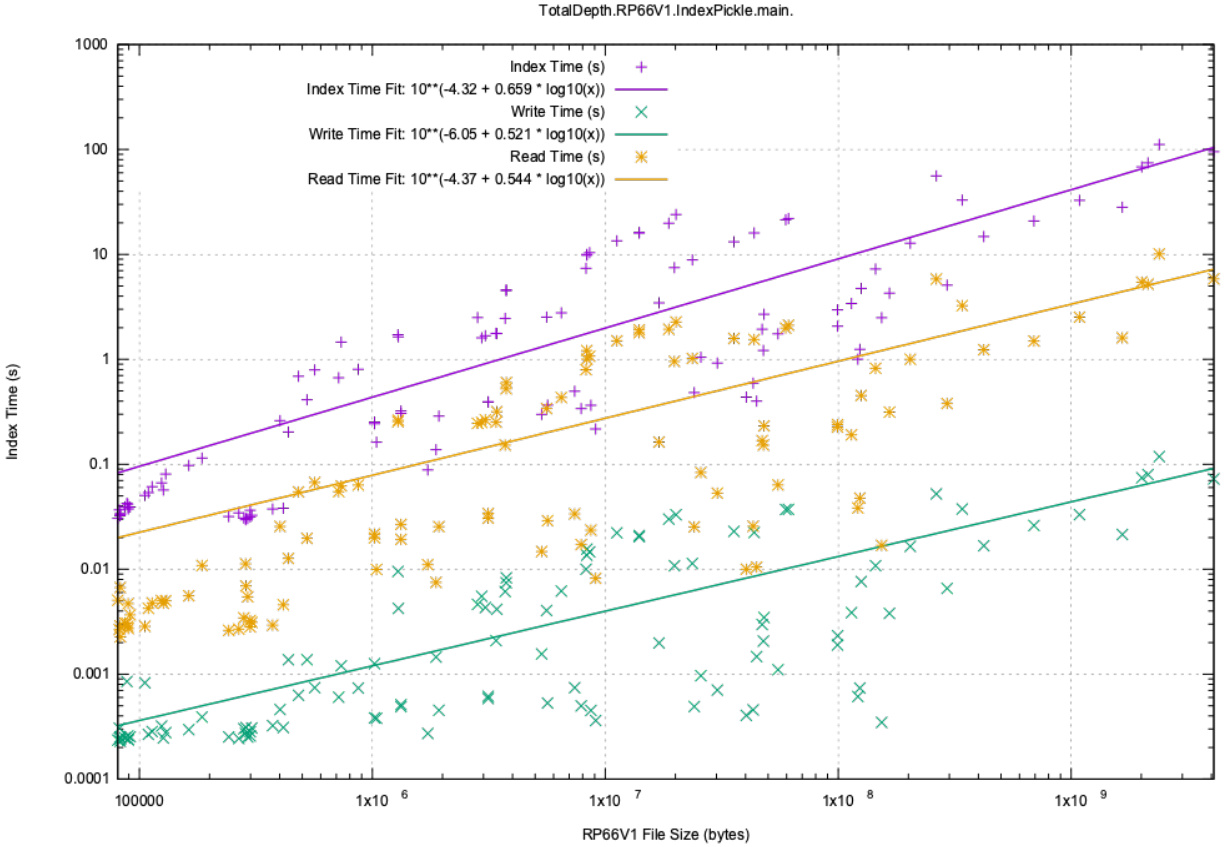
- Every *RP66V1.Explicitly Formatted Logical Record* (EFLR) as a full internal representation.
- A reference to every *RP66V1.Indirectly Formatted Logical Record* (IFLR) as a partial internal representation. This does not contain the entire IFLR but does have each X axis value.

Naturally enough this index takes longer to build and uses more memory (and more disc space when persisted).

The index is implemented by `TotalDepth.RP66V1.core.LogicalFile.LogicalIndex` The following performance data was gathered by `TotalDepth.RP66V1.IndexPickle`.

Execution Time

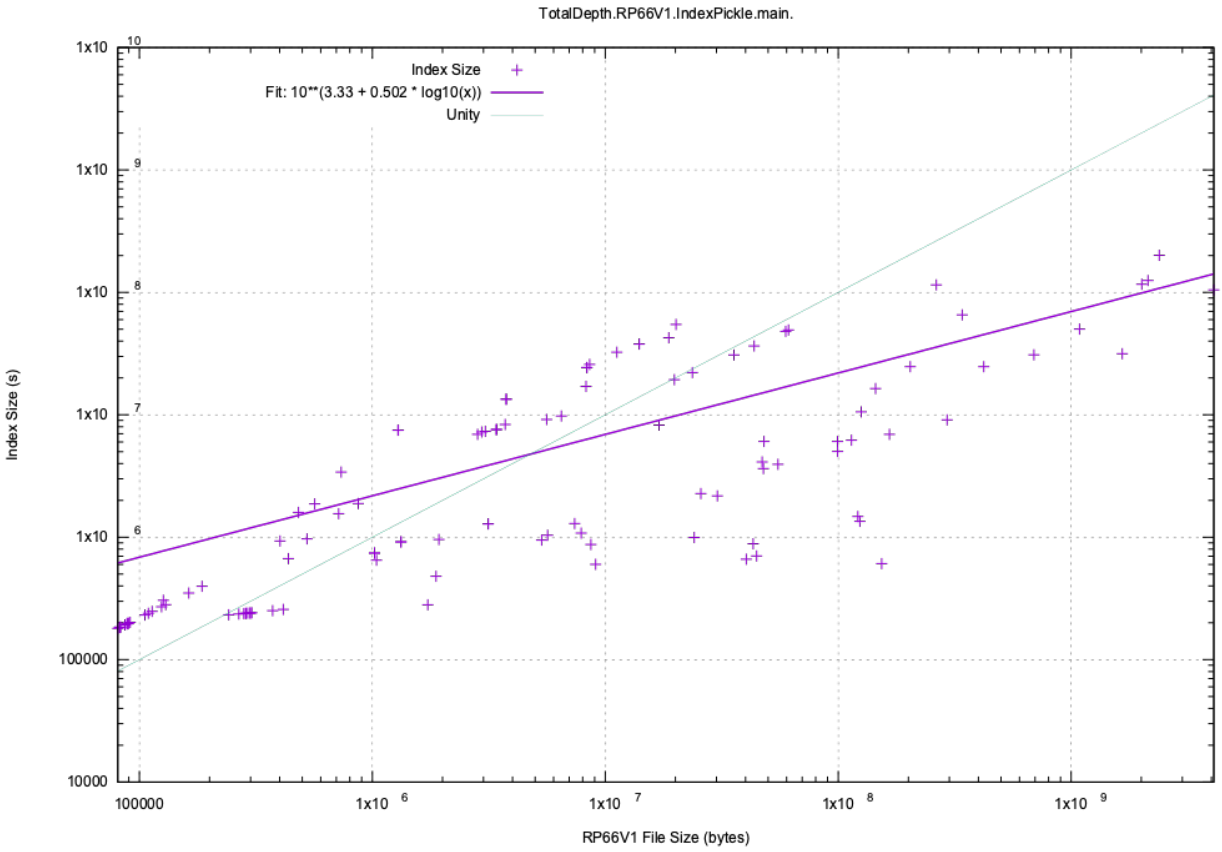
Firstly time to create, persist and read back the persisted index:



While there is quite a variation by a factor of 10 the *average asymptotic* execution time trends to:

- Create the index: 42 ms/Mb (this includes the low level index of 10.6 ms/Mb).
- Persist the index: 0.07 ms/Mb (this seems remarkably small).
- Read the persistent index: 4 ms/Mb

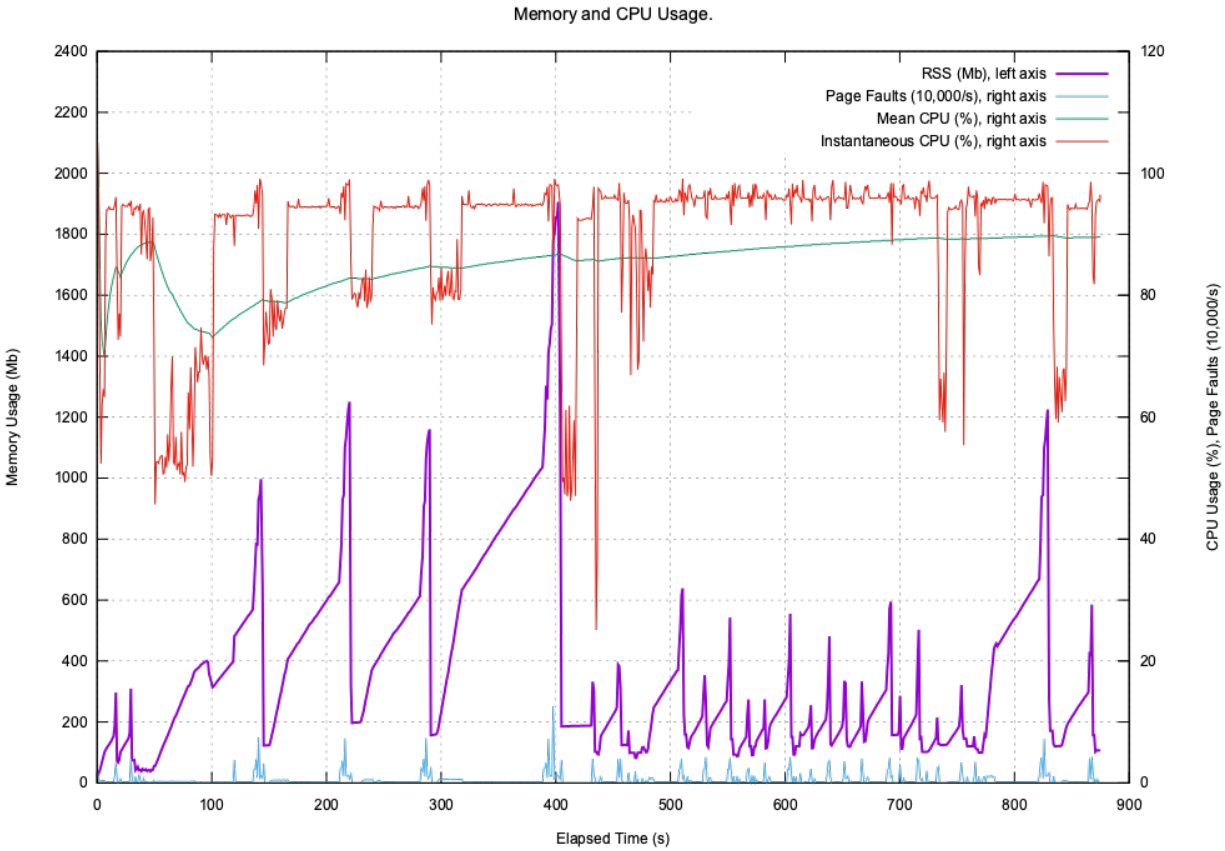
Index Size



The index is roughly twice the size of the low-level index (although that ratio will increase dramatically when the C/C++ low-level index code is merged).

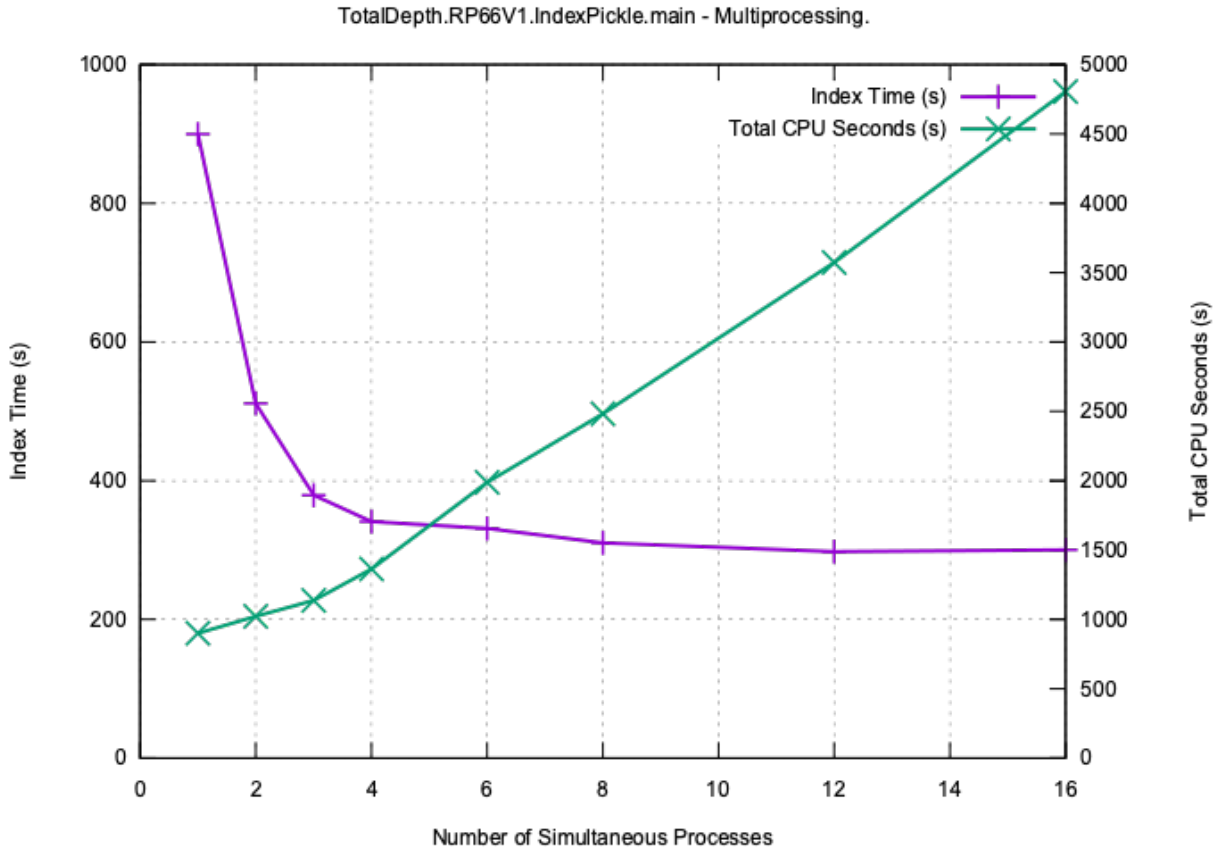
Memory and CPU Usage

Indexing the test set makes this memory and CPU demands:



Multiprocessing

Here is the same test data set being indexed including pickling the index to disk with a different number of simultaneous processes.



Because the I/O is so much higher the improvement is more limited, at best x3 faster with 4 cores but still useful.

1.7.6 RP66V1 Performance

This describes the performance of processing RP66V1 binary files by TotalDepth.

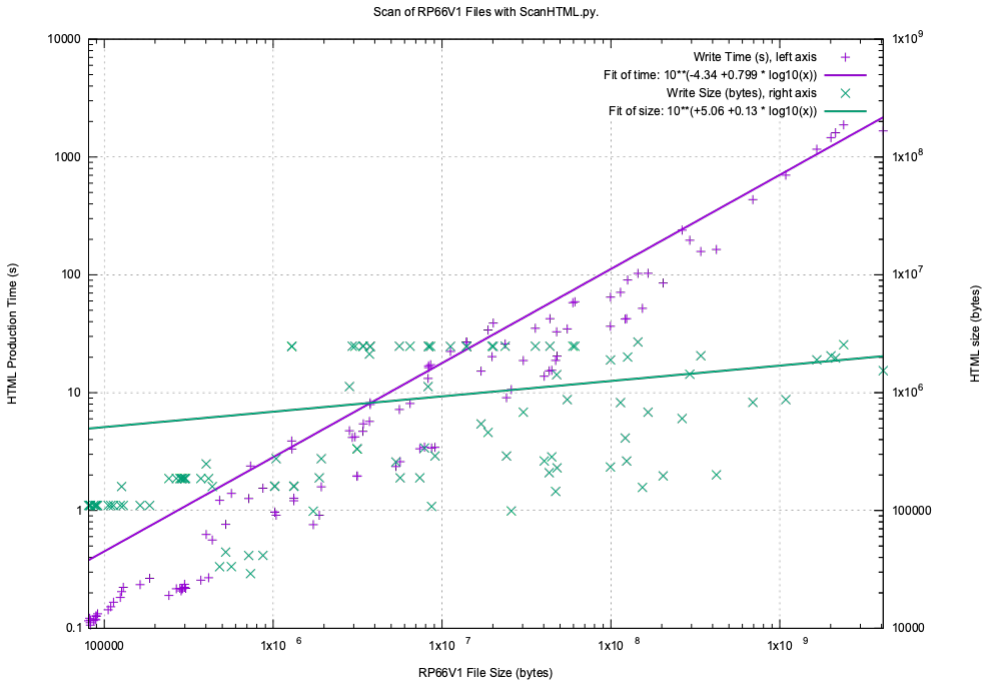
Note: This data refers to version 0.3.0 and may or may not be relavent to the current version, 0.4.0rc0.

Scanning RP66V1 to produce an HTML Summary

The tast set was around 100+ files that ranged in size from 80kb to 4GB totalling 17GB. The average file size was about 150Mb. These tests were run on a 2.7 GHz Intel Core i7 machine with 4 cores and hyper-threading.

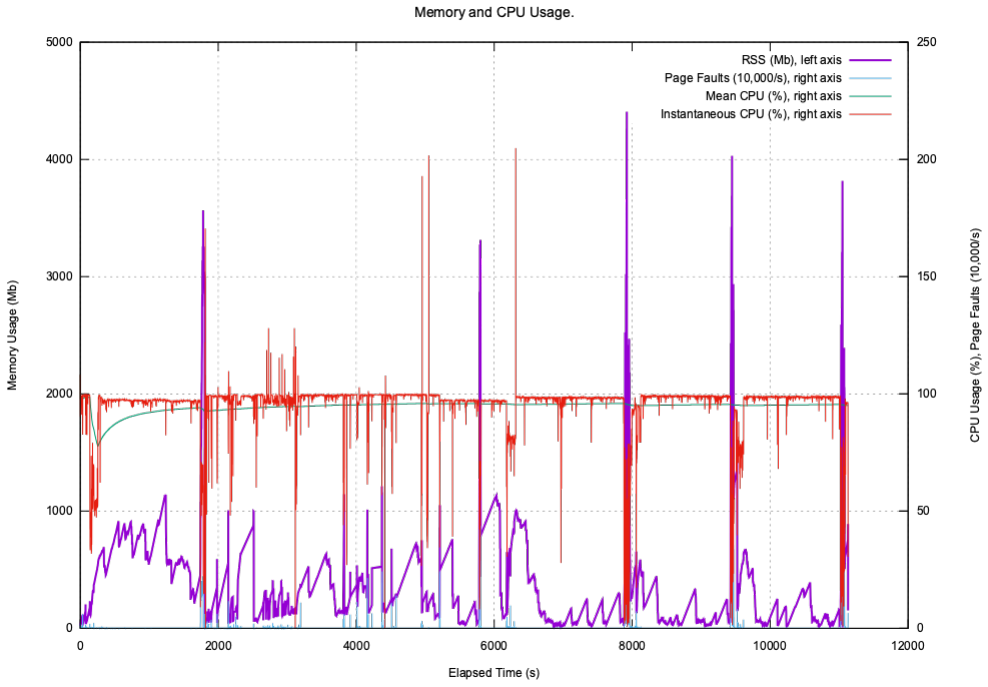
An archive of RP66V1 can be scanned to provide a summary in HTML by `TotalDepth.RP66V1.ScanHTML` This produces and HTML page with every EFLR and a summary of all the log frame data. This is exposed as a command line tool *tdrp66v1scanhtml*.

By default this processes every byte of the file so can take a long time with large files. Here is the execution time for processing all frames by RP66V1 file size and the size of each HTML file produced.



The asymptotic processing rate is around 800 ms/Mb.

And here is the memory and CPU usage:



Performance Improvements

With very large sets of frame data not every frame needs to be processed. The option `--frame-slice` can be used to sample a subset of frames. For example:

- `--frame-slice=1024,2048,64` will process every 64th frame from frame 1024 to 2048
- `--frame-slice=64` will process only 64 frames of those available (roughly evenly spaced from those available).

Multiprocessing will help proportionally. In one test case the processing time for the archive fell from 11,000 seconds to 800 seconds using `--frame-slice=,,64` and `--jobs=4` on a four core machine. The performance improvement of around x15 was attributed to x5 (frame slicing) and x3 (multiprocessing).

Converting RP66V1 to LAS

The test set was around 52 files that ranged in size from 400kb to 2.2GB totalling 12GB in all. The average file size was about 235Mb. These tests were run on a 2.7 GHz Intel Core i7 machine with 4 cores and hyper-threading.

An Example File

An example file in the tests archive is 2GB in size and contains this Log Pass, the size of the numpy frame to hold all the data is also shown:

Frame Array	Channels	Frames	Spacing	Size of Numpy array (bytes)
1B (O: 35 C: 0)	12	653880	0.1 inch	31,386,240
2B (O: 35 C: 0)	6	326940	0.2 inch	7,846,560
10B (O: 35 C: 0)	19	65388	1 inch	10,200,528
15B (O: 35 C: 0)	14	43592	1.5 inch	2,441,152
20B (O: 35 C: 0)	83	32694	2 inch	10,462,080
60B (O: 35 C: 0)	204	10899	6 inch	2,294,980,632
120B (O: 35 C: 0)	6	5449	12 inch	11,028,776

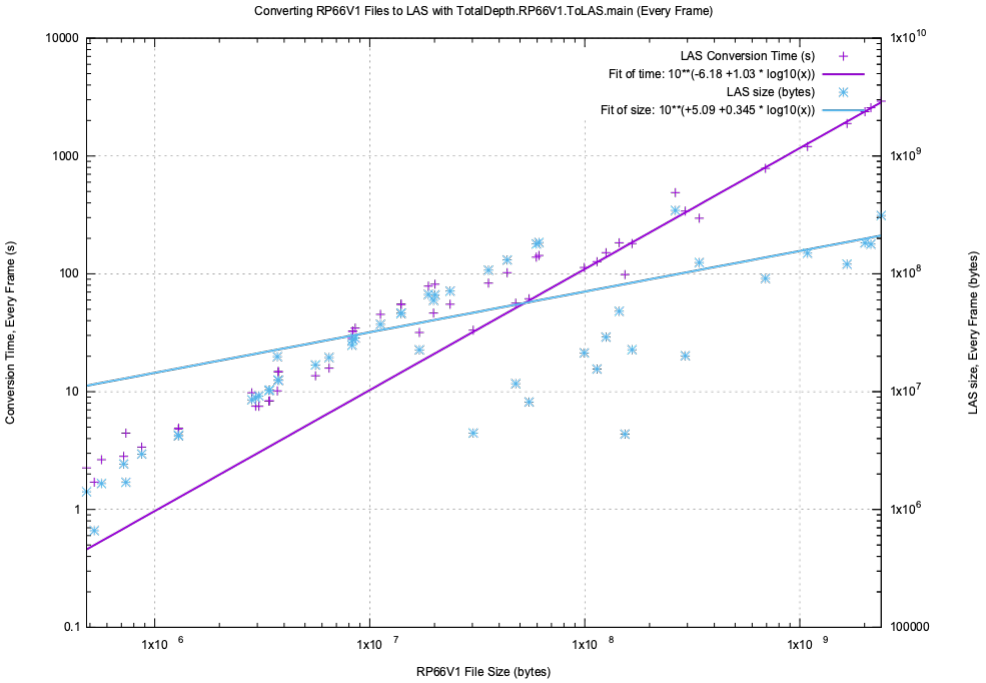
The 60B Frame Array contains some complex waveform data. Processing this produces seven LAS files, each LAS file contains all the parameter data and data from a single Frame Array. Depending on the frame slice the following processing times and LAS sizes were observed.

Frame Slice	Time (s)	LAS Size (bytes)	ms/Mb
1	2922	312,604,951	1282
4	855	100,570,379	375
16	303	47,562,819	133.4
64	188	34,309,331	82.8
256	156	30,995,584	68.6
512	152	30,444,007	66.9
1024	148	30,166,341	65.0

And or course limiting the number of channels has a proportionately similar effect.

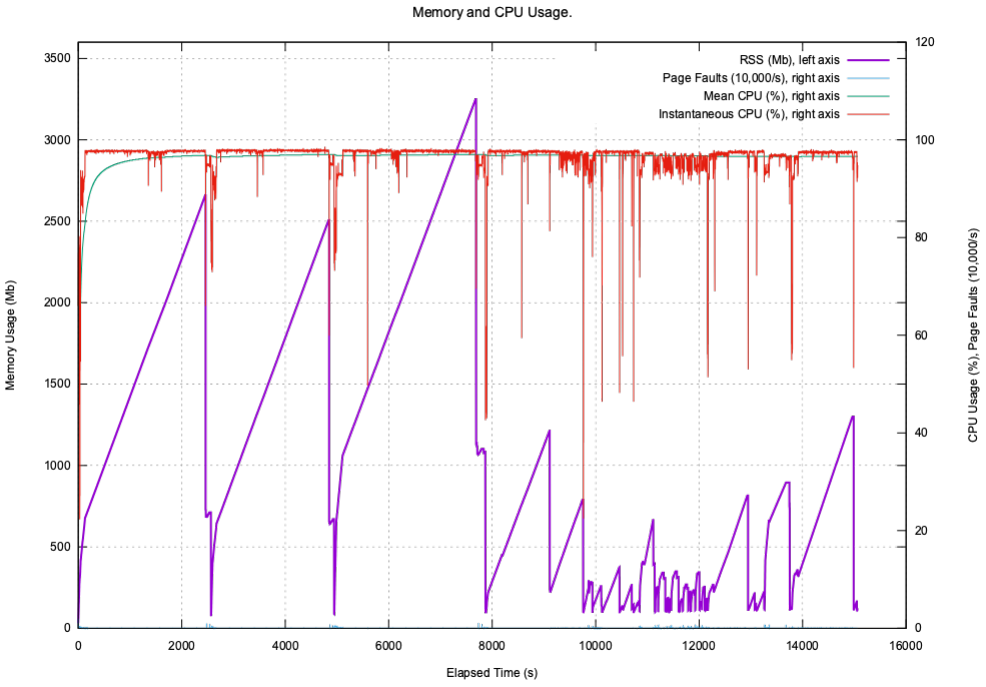
Processing the Test Archive to LAS

Here is the time taken to process each file plotted against the RP66V1 file size. Also plotted on the right scale is the total size of the LAS file(s). This is converting every frame to LAS (click to see the original):



The asymptotic processing rate is around 1230 ms/Mb. LAS files (in this test set) below 10Mb tend to be larger than the original, above 100Mb they tend to be around 20% smaller.

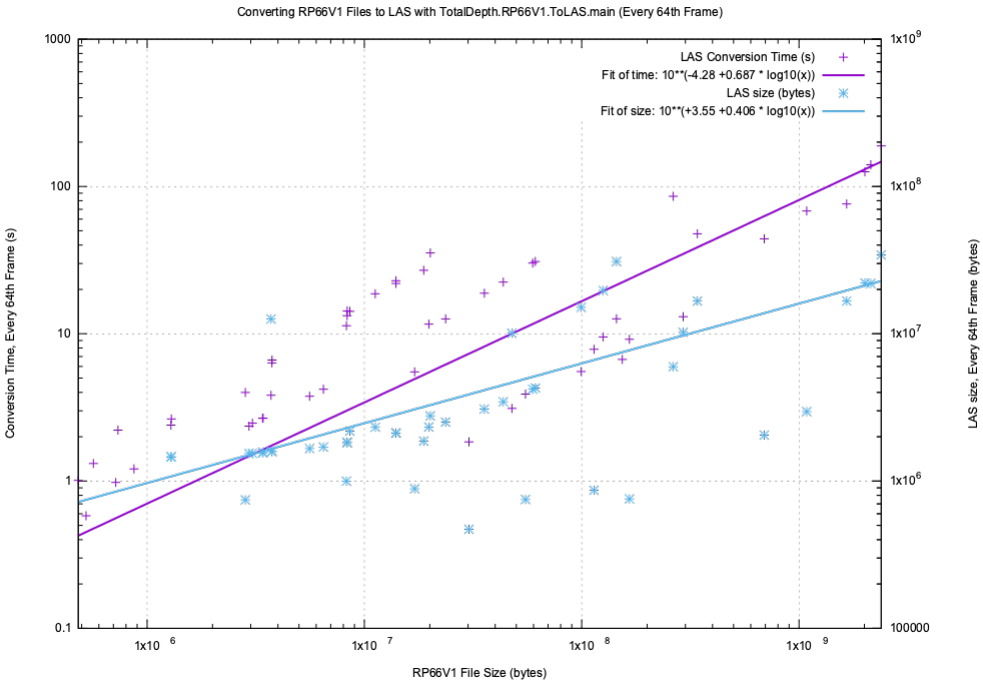
And this is the memory usage (click to see the original):



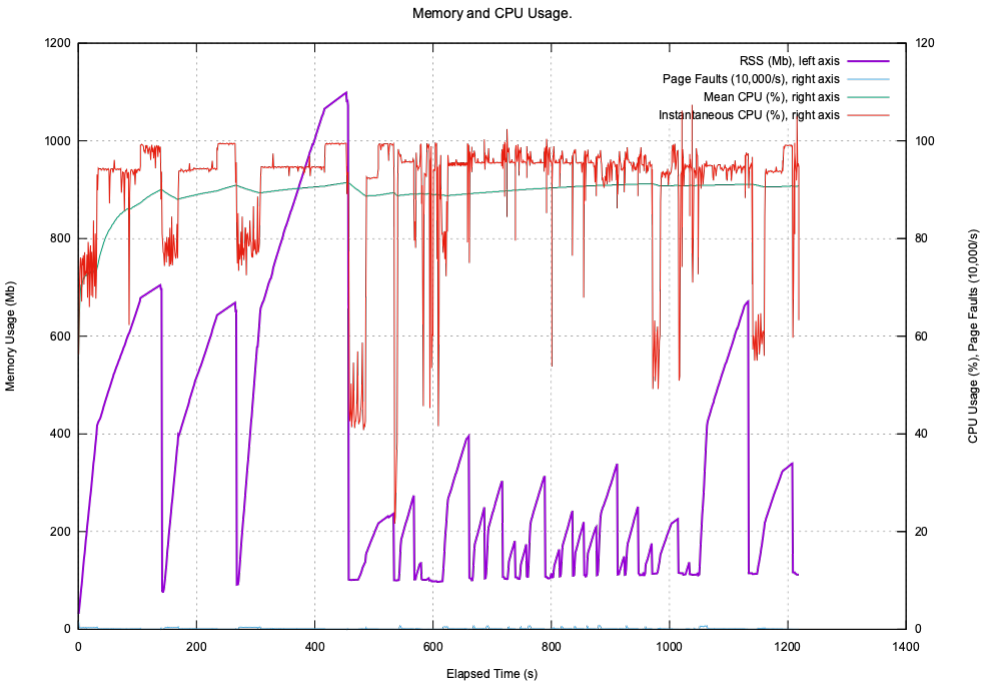
The peaks are caused by the multi-gigabyte Numpy arrays needed for some files.

Sub-sampling

Here is the performance of the same data with `--frame-slice=64` that just writes every 64th frame to LAS (click to see the original):

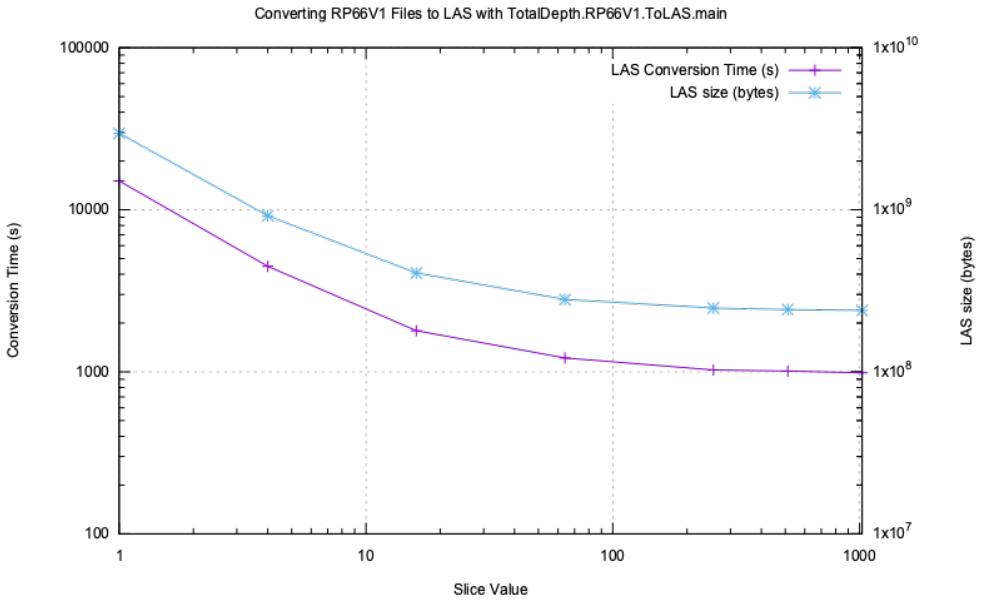


And this is the memory usage (click to see the original):



This is vastly reduced from the every frame case.

Frame slicing can improve the performance dramatically. Here is the time to process the test archive and the size of the finished LAS archive by slice, for example 64 on the X-axis is write only every 64th frame:



Multi-Processing

Using `--jobs` can also improve performance proportionally if you have lots of cores and good I/O.

1.7.7 Plotting Wireline Data

Aside from the LogPass (the data structure that holds the frame data) the plot layout needs to be specified. There are two ways that TotalDepth does this:

- A plot specification that is *integral* to the file. For example; LIS files may well have FILM and PRES tables within them that configure the plot as the recording engineer (or software) intended. In this case TotalDepth can make those plots directly.
- A plot specification that is specified *externally* to the file. TotalDepth supports one such way; using LgFormat XML files and these are described below.

Wireline Files With Integral Plot Data

This applies to LIS (and possibly RP66 files). The LAS file specification does not describe any plot specification at all.

LIS Plot Specification

There are a number of table-like Logical Records (type 34) within a Log Pass that can specify the plot layout. As a minimum TotalDepth.LIS needs a `FILM` and a `PRES` record.

FILM Record

A dump of a `FILM` Logical Record might look like this:

```
Table record (type 34) type: FILM

MNEM  GCOD  GDEC  DEST  DSCA
------------------------------

1     EEE   ----  PF2   D200
2     E4E   -4--  PF1   DM
```

The columns are:

Column	Description
`MNEM`	The (logical) name of the output, TotalDepth uses this as part of the plot file name.
`GCOD`	The coding of the tracks. `E4E` means three tracks which have linear, log (4 decades), linear. With three tracks the X axis track (depth for example) appears between the first and second tracks.
`GDEC`	The number of logarithmic decades for each track, `-` for linear.
`DEST`	The physical destination (in the logging unit) of the plot. Ignored.
`DSCA`	The depth scale, encoded. For example `D200` means 1:200.

PRES Record

A dump of a `PRES` Logical Record might look like this:

```
 Table record (type 34) type: PRES

MNEM  OUTP  STAT  TRAC  CODI  DEST  MODE      FILT          LEDG          REDG
-------------------------------------------------------------------------------

SP    SP    ALLO  T1    LLIN  1     SHIF      0.500000      -80.0000       20.0000
CALI  CALI  ALLO  T1    LDAS  1     SHIF      0.500000       5.00000       15.0000
MINV  MINV  DISA  T1    LLIN  1     SHIF      0.500000       30.0000       0.00000
MNOR  MNOR  DISA  T1    LDAS  1     SHIF      0.500000       30.0000       0.00000
LLD   LLD   ALLO  T23   LDAS  1     GRAD      0.500000      0.200000       2000.00
LLDB  LLD   ALLO  T2    HDAS  1     GRAD      0.500000       2000.00       200000.
LLG   LLG   DISA  T23   LDAS  1     GRAD      0.500000      0.200000       2000.00
LLGB  LLG   DISA  T2    HDAS  1     GRAD      0.500000       2000.00       200000.
LLS   LLS   ALLO  T23   LSPO  1     GRAD      0.500000      0.200000       2000.00
LLSB  LLS   ALLO  T2    HSPO  1     GRAD      0.500000       2000.00       200000.
MSFL  MSFL  ALLO  T23   LLIN  1     GRAD      0.500000      0.200000       2000.00
11    DUMM  DISA  T1    LLIN  NEIT  NB        0.500000       0.00000       1.00000
12    DUMM  DISA  T1    LLIN  NEIT  NB        0.500000       0.00000       1.00000
...
```

Column	Description
MNEM	The (logical) name of the output curve. Note multiple curves might come from one OUTP.
OUTP	The source of the curve.
STAT	Status, is this curve to be plotted.
TRAC	Which track the curve should be plotted on.
CODE	The line coding, dot, dash etc.
DEST	The logical film that this curve is sent to. Can be BOTH and so on.
MODE	What to do when the curve goes off scale (wrap round the track for example).
FILT	Filtering. Unused.
LEDG	Value of the left edge of the plot.
REDG	Value of the right edge of the plot.

Records Needed for a LIS Plot

As a minimum TotalDepth needs a FILM and PRES plot, using these TotalDepth can produce a plot identical to that when the data was recorded.

In future releases TotalDepth might also support these Logical Records for plotting:

- AREA: Describes shading between curves on a plot.
- PIP: Describes integration marks on the plot such as integrated transit time, integrated hole volume etc.
- VDL: Describes presentation of waveform amplitude data.

Wireline Files With External Plot Data

This can apply to any wireline file and it means using an external configuration file to decide the plot layout. For example; LAS files do not have any plot specification (the LAS standard precludes this). The presentation of the plot is made with an external plot specification.

These external plot specifications are numerous and varied. TotalDepth currently supports one such specification: the XML *LgFormat* or *LgSchema* files.

LgFormat XML Files

These appear to originate from some of Schlumberger's "free" software.

Here is an example LgFormat XML file for plotting HDT logs .TotalDepth supplies an number of examples of these.

There are a number of problems[1] with the LgFormat which is why TotalDepth only supports a subset of it while we look for something better.

[1] For example the LgCurve element content of <LeftLimit> and <RightLimit> is numeric, without units. This robs a compliant implementation of the opportunity of converting frame data, say in "V/V " to plot data say "PU " which might be far more appropriate. Truth in advertising prompts me to say that LIS PRES tables have the same flaw but they have a reasonable excuse that they are *per file* specification rather than a global specification.

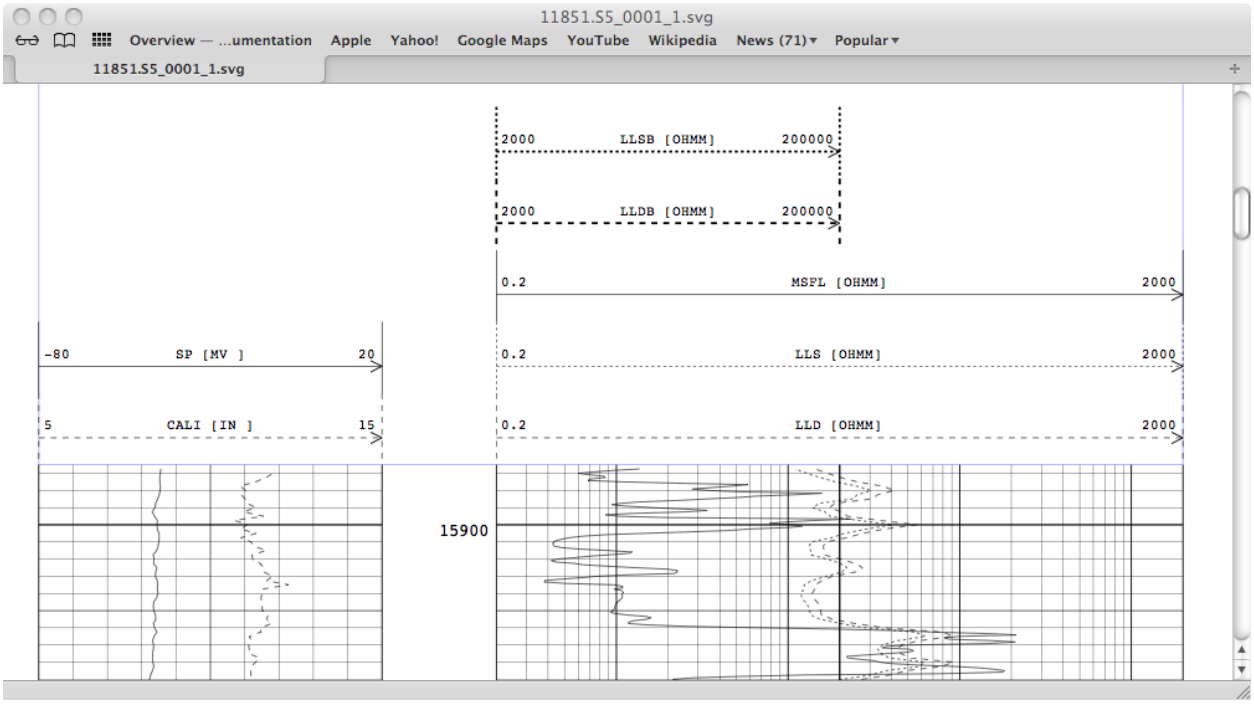
Internal Plot Algorithms

This describes some algorithms used by the `Plot` module. This is for information only as these algorithms are internal and the user has little or no control over them.

Plotting the Legends (Scales)

These at each end of the plot per curve and may span a number of tracks or be limited to half a track. Here is an example which has seven curves:

TODO: Finish this



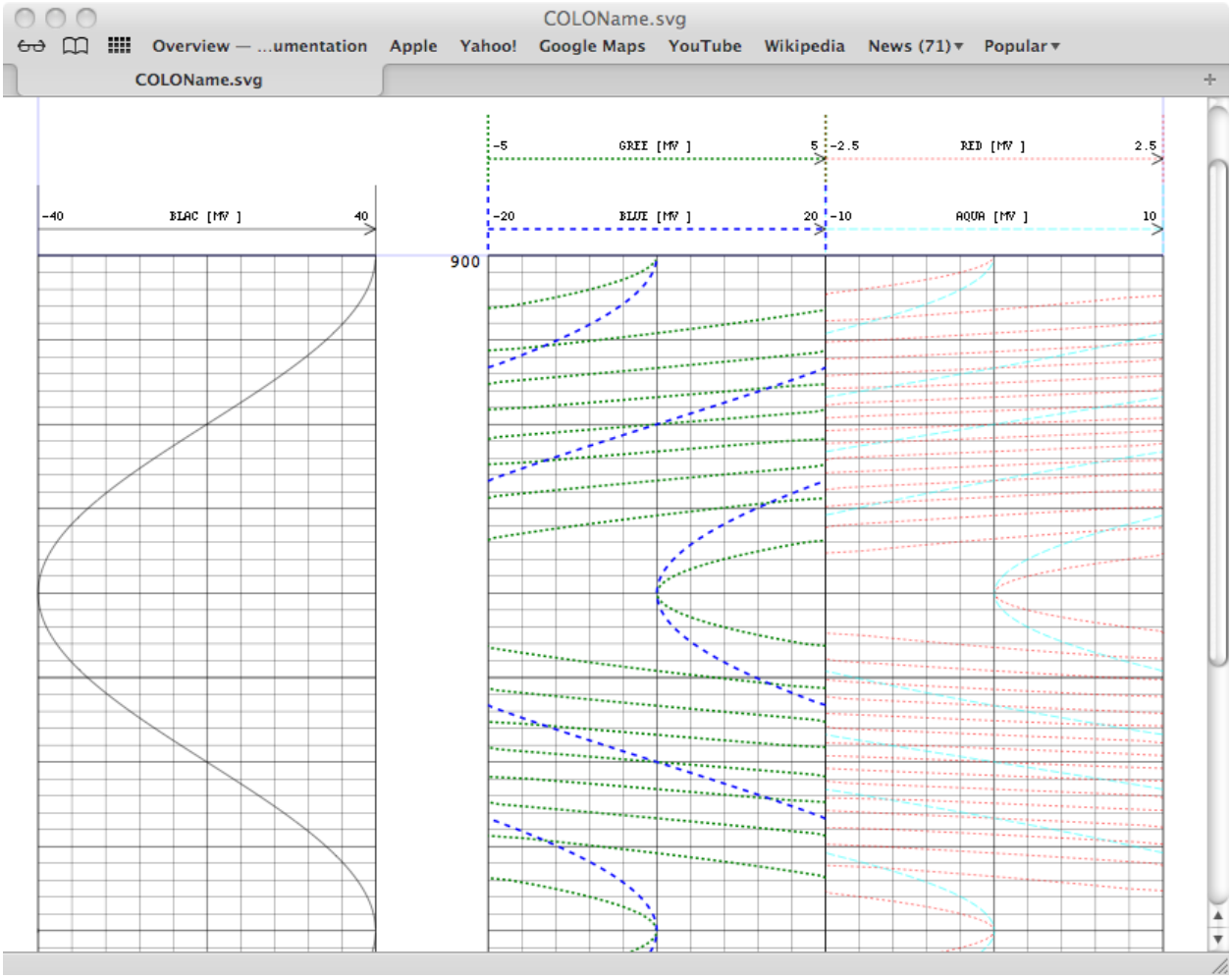
There are two important methods that implement this algorithm: `Plot.Plot._retCurvePlotScaleOrder()` and `Plot.Plot._plotScale()`

Interpolating Backups

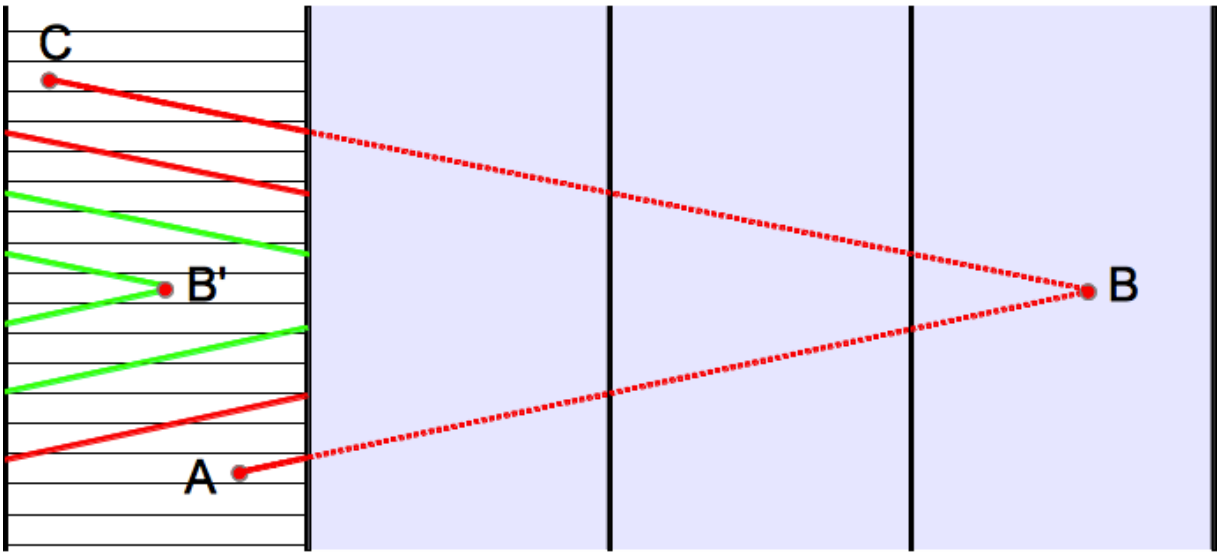
Industry standard wireline plots have the notion of curve backups where if the the trace for a particular curve goes off scale on one side of the track it might well reappear from the alternate side.

Here is a simulated example generated by `TestPlot.py` where a single curve (a sin curve with an amplitude of +/- 40) is plotted in full on track one. The same curve is plotted fully backed up:

- In green on track two with a scale of +/- 5
- In blue on track two with a scale of +/- 20
- In red on track three with a scale of +/- 2.5
- In aqua on track three with a scale of +/- 10



The algorithm that does this is probably best illustrated by example. The diagram below shows a curve (in the white track on the left) that goes from from point A goes off-scale to the right to point B in 'real' space. The line A-B is subdivided by track widths and these *crossing* lines are plotted by interpolation in the track space making the line space A-B' (so-called 'FlyBack' lines are omitted in the diagram). If the *Backup Mode* is *wrap* then both red and green lines in the track are plotted. In this case if the *Backup Mode* is *1* then only the red lines are plotted in the track i.e. only one backup allowed.



On the return B-C a similar process happens in reverse.

Backup Specification

A backup specification is expected for any plot *per-curve*. Internally a backup specification is a pair of integers. If the virtual track position is less than the value (if left) or greater than the value (if right) then it is off scale. Zero is a special case in that all virtual track positions are on scale.

Various values have been predefined and there are existing mappings from LIS data and LgFormat XML files. The following table describes all of this.

Spec.	Symbol	Description	LIS Mapping	LgFormat Mapping
(1, -1)	BACKUP_NONE	No backup	'NB ', 'GRAD'	
(0, 0)	BACKUP_ALL	Every backup i.e. 'wrap' Note: Plot.py has a way of limiting ludicrous backup lines to a sensible number; say 4	'WRAP'	'LG_WRAPPED' 'LG_X10'
(-1, 1)	BACKUP_ONE	Single backup to left or right	'SHIF'	'1'
(-2, 2)	BACKUP_TWICE	Two backups to left or right		'2'
(0, -1)	BACKUP_LEFT	Single backup to left only		'LG_LEFT_WRAPPED'
(1, 0)	BACKUP_RIGHT	Single backup to right only		'LG_RIGHT_WRAPPED'

Note that it is also common to have one curve with no backup and a second curve driven from the same output with a different scale acting as a backup. This has the advantage that a different coding and colour can be assigned to it. See *Plotting the Legends (Scales)* above for an example with a Laterlog plot.

Algorithm

The backup algorithm works as follows:

- For every point plotted an integer wrap value is calculated.
- If the wrap value is different from the previous wrap value then some interpolation is required and the `Plot._interpolateBackup()` method is called (below).
- Otherwise the point is plotted.

_interpolateBackup

The `Plot._interpolateBackup()` method manages all the different cases where a backup is needed given the backup configuration. It relies on `Plot._retInterpolateWrapPoints()` to generate a list of interpolated lines, including *crossing lines*, between one point and another.

_filterCrossLineList

A further refinement to the algorithm is limiting the number of *crossing lines*. Consider the SP curve represented by this data taken from a real example:

```
Depth (ft) SP (mV)
10245.0    332.724
10245.5    13716948.000
10246.0    41150204.000
10246.5    68583456.000
10247.0    96016712.000
10247.5    334.176
```

Clearly some episode happened between 10245.0 and 10245.5 feet that caused a jump of 13716615.276 mV. This could have been an recording disturbance (unlikely as where would you find 96kV at 10247 feet?) or an editing error. In any case on a scale of -20 to 80 mV means 137,166 (mostly spurious) wrap lines crossing the track. This can turn a 1.6Mb file into a 91Mb file! To stop this a arbitrary limit is made to the number of *crossing lines* (e.g. 4) between each Xaxis interval. This limit filters the *crossing line* list to an evenly distributed list of 4 (or thereabouts).

This is also applicable to RFT plots where it is common to plot the pressure modulo 10 psi in RHT3.

The method that does this is The `Plot._filterCrossLineList()` method

1.7.8 Data Quality

TODO:

1.7.9 TotalDepth and SaaS

This design shows the potential for running TotalDepth as *Software as a Service* (SaaS) or, if you prefer, *Cloud Computing*.

The design envisages a server running the full TotalDepth software and any number of clients running sub-sets of that software. Client software could be in-browser or on-device.

Network speed (i.e. latency+bandwidth) is crucial to the performance of this system and the network connection between the client and the server could be any of these (in order of slowness):

- A client and a server both running on the client's machine.
- A client device and a server running on a high speed LAN.
- A client device communicating with a server via a VPN.
- A client device communicating with a remote server via the internet.

TotalDepth accommodates all of these.

Processing Server Side Files

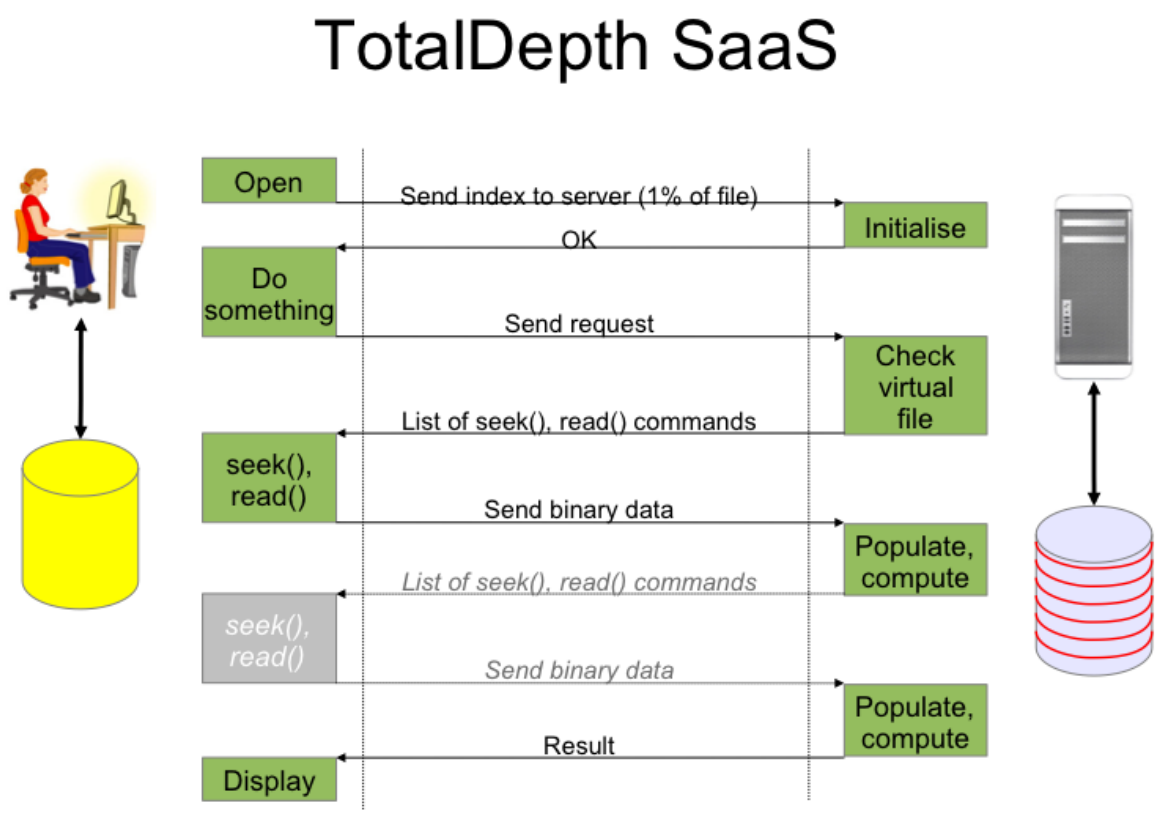
There is no particular challenge here as the server has all it needs to process any client instruction. The big disadvantage is that the client has to upload the file to the server before they can do anything. For small jobs on large files that can impose an disproportionate cost.

Processing Files that Remain on the Client's Device

The real challenge is: the client has a (very large?) file and wants to do something small (or at least incremental) with it, for example "Plot curves XYZ from 5840 to 5600 feet using format ABC" and the server only needs a tiny fraction of that file. Regardless of the network connection there should be some cunning in extracting the data that the client needs in minimal time. TotalDepth's indexing technique makes this possible.

This design shows how the client can work with the server without the tedious business of the client having to upload the complete file to the server. Instead the client and server cooperate to make sure that the minimum data is transferred for the immediate task in hand.

The design is illustrated below, the client is on the left:



Starting the session

The session is initialised by the user requesting the client to operate on a particular file on the clients device. The client passes basic file information to the server and receives a unique session ID.

The Session ID

The session ID could be a portable file ID, such as a checksum. Thus two users using the same file could benefit from each others read transactions. This would require copy-on-write and session ID modification.

Another optimisation when using a checksum is that if the server already has an exact copy of the file (using the checksum as a proxy for that) then the server could provide data from its own copy of the file without the user having to transfer any data to the server at all.

Session Bootstrapping

Given a valid session ID the client code reads the minimum structural information for the server to comprehend (and mirror) the file structure. This will vary according to the file format (and will vary according to file size).

LIS

The client code must be capable of processing TIF markers, Physical Record headers and trailers and Logical Record headers. The client passes this data as an ordered list of pairs `[(tell, bytes), ...]` where tell is the `size_t` position in the of the Physical Record (or TIF marker) that starts the Logical Record and `bytes` is the raw two bytes of the Logical Record Header.

LAS

As LAS files are small and do not lend themselves easily to efficient indexing then, most likely, it is worth passing the entire client file to the server.

RP66

This will be similar to LIS bootstrapping with the client code processing Visible Record Headers and Logical Record segments.

Index Completion

The bootstrap information passed to the server allows the server to construct a virtual, partial, logical image of the file. The server may request further information in the form of an ordered list of pairs `[(file position, number of bytes), ...]`. The client does a series of seek/read operations and passes the data back to the server as a list of pairs `[(file position, bytes), ...]`. This allows the server to complete the file index.

The total size of the data transferred to the server at this stage is typically 0.6% of the file size. See *Indexing LIS Files* for a description of how LIS files are indexed and *Multi-Processing* for a study of index sizes.

Rest of Session

Once bootstrapped any user request is passed by the client to the server, for example: “Plot curves ABC from 5840 to 5600 feet”.

The server then consults its virtual file image to see if it has the data to satisfy the request. If so then the result of the request is sent to the client.

If the server does not have sufficient data in its virtual file image then the server will send to the client an (ordered) list of pairs `[(file position, number of bytes), ...]`. The client does a series of seek/read operations on each pair and and passes the data back to the server as a list of pairs `[(file position, bytes), ...]`. The server adds these to its virtual file image. The server then can complete the request.

Over time the virtual file image would grow but only as fast as the user’s (new) demands.

Data Persistence

It is possible for the server to persist the virtual file image over any number of sessions, this would improve the performance even further.

SaaS Pros and Cons

Advantages

- Low barrier to use: browser based, no installation.
- Cross Platform: desktop, tablet, mobile etc.
- Minimum client code.
- Continuous software version update from the server.
- Integrated with other Internet services for example mapping data.
- Tailored appearance per client.
- Cloud availability behind the server.
- All usage data is available on server logs.

Disadvantages

- Requires network connection (could have a server version running locally).
- User agent variability.
- Speed/performance limited by network.
- Server infrastructure and investment.
- Continuous maintenance and support.
- Security.

1.7.10 Process Monitoring with `TotalDepth.common.process`

TotalDepth.common.process can monitor the memory and CPU usage of a running process. It does this by creating a thread which, at regular intervals, reports process data to the log file in JSON. The basic use is like this:

```
from TotalDepth.common import process

with process.log_process(1.0):
    # Your code here
```

Then *TotalDepth.common.process* will write process data as a single line in the log file every 1.0 seconds. The JSON data is preceded by the following, recognisable, entry in the log file:

```
2019-10-31 14:10:06,051 - process.py - 86611 - (ProcMon   ) - INFO     -␣
↪ProcessLoggingThread-JSON
```

The JSON data looks like this example (but on one line):

```
{
    "timestamp": "2019-10-31 14:10:06.051630",
    "memory_info":
        {
            "rss": 11939840,
            "vms": 4404228096,
            "pfaults": 531770,
            "pageins": 0
        },
    "cpu_times": {
        "user": 0.583945792,
        "system": 0.66087648,
        "children_user": 0.0,
        "children_system": 0.0
    },
    "elapsed_time": 12.42771601676941
}
```

Command Line Tools

Command line tools can add `process` capability with an argument parser created by *`TotalDepth.common.cmn_cmd_opts`* `arg_parser()`:

```
process.add_process_logger_to_argument_parser(parser)
```

This makes the `--log-process` option available which takes a numeric value as a float in seconds (default zero which means no process logging) for the logging interval. Your code pattern is then:

```
args = parser.parse_args()
if args.log_process > 0.0:
    with process.log_process(args.log_process):
        # Do something
```

Plotting the Data

`TotalDepth.common.process` can be used from the command line to extract the data from the log file and plot it with Gnuplot.

Example

Here we will create eight large, randomly sized strings and simulate doing some work:

```
import random
import time

from TotalDepth.common import process

with process.log_process(0.1):
    for i in range(8):
        size = random.randint(128, 128 + 256) * 1024 ** 2
        s = ' ' * (size)
        # Simulate 0.5 to 1.5 seconds of work.
```

(continues on next page)

(continued from previous page)

```
        time.sleep(0.5 + random.random())
        del s
        # Simulate 0.25 to 0.75 seconds of work.
        time.sleep(0.25 + random.random() / 2)
```

This will produce a log such as:

```
2019-10-31 14:09:53,726 - process.py - 86611 - (ProcMon   ) - INFO     -␣
↪ProcessLoggingThread-JSON {"timestamp": "2019-10-31 14:09:53.726676", "memory_info
↪": {"rss": 11898880, "vms": 4404228096, "pfaults": 3624, "pageins": 0}, "cpu_times
↪": {"user": 0.07540488, "system": 0.020255324, "children_user": 0.0, "children_
↪system": 0.0}, "elapsed_time": 0.10263395309448242}
2019-10-31 14:09:53,896 - process.py - 86611 - (ProcMon   ) - INFO     -␣
↪ProcessLoggingThread-JSON {"timestamp": "2019-10-31 14:09:53.896083", "memory_info
↪": {"rss": 162922496, "vms": 4555227136, "pfaults": 40495, "pageins": 0}, "cpu_times
↪": {"user": 0.108017992, "system": 0.056484236, "children_user": 0.0, "children_
↪system": 0.0}, "elapsed_time": 0.27210497856140137}
2019-10-31 14:09:53,997 - process.py - 86611 - (ProcMon   ) - INFO     -␣
↪ProcessLoggingThread-JSON {"timestamp": "2019-10-31 14:09:53.996930", "memory_info
↪": {"rss": 162930688, "vms": 4555227136, "pfaults": 40497, "pageins": 0}, "cpu_times
↪": {"user": 0.10846144, "system": 0.05655662, "children_user": 0.0, "children_system
↪": 0.0}, "elapsed_time": 0.373028039932251}
...
2019-10-31 14:10:06,051 - process.py - 86611 - (ProcMon   ) - INFO     -␣
↪ProcessLoggingThread-JSON {"timestamp": "2019-10-31 14:10:06.051630", "memory_info
↪": {"rss": 11939840, "vms": 4404228096, "pfaults": 531770, "pageins": 0}, "cpu_times
↪": {"user": 0.583945792, "system": 0.66087648, "children_user": 0.0, "children_
↪system": 0.0}, "elapsed_time": 12.42771601676941}
```

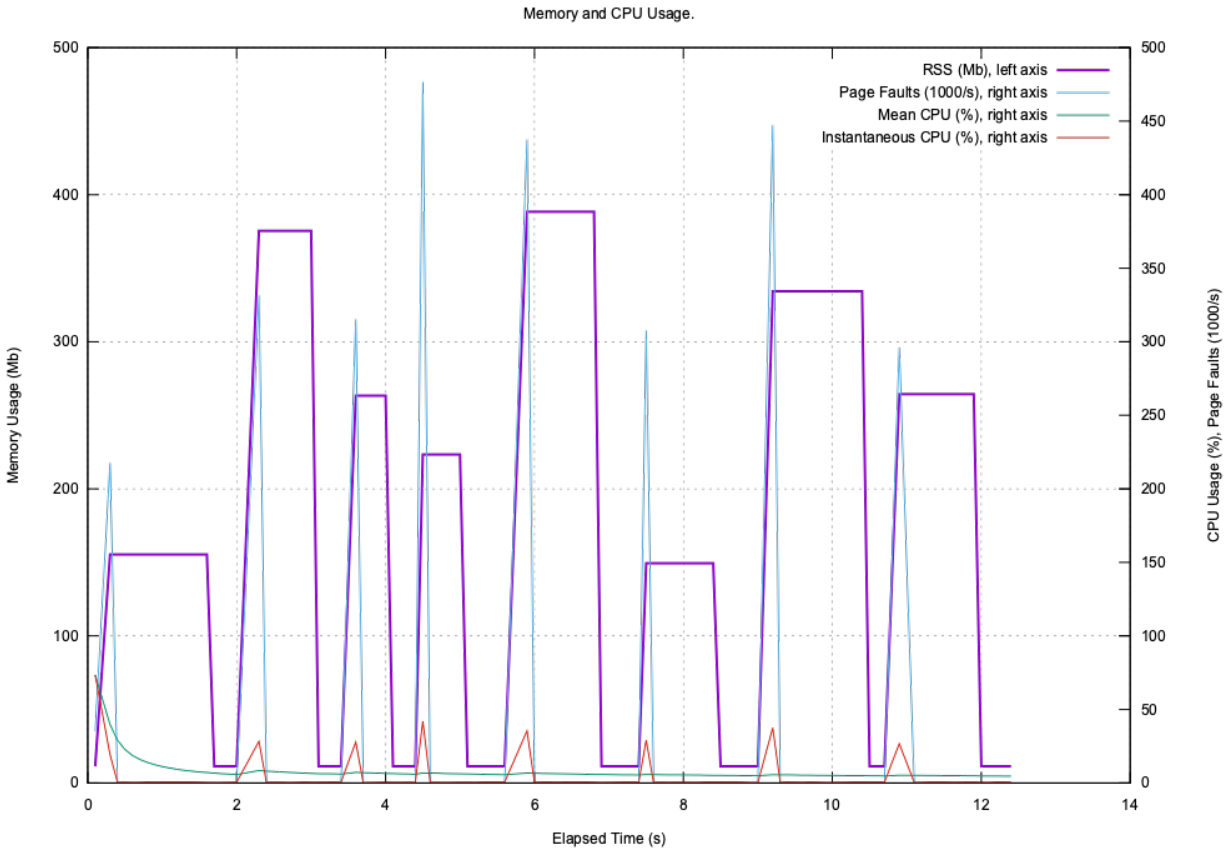
Then run *TotalDepth.common.process* CLI entry point with two arguments, the log file and a directory to write the Gnuplot data to.

```
$ python src/TotalDepth/common/process.py tmp/process_C.log tmp/gnuplot_process
2019-10-31 14:11:29,737 - gnuplot.py - 86631 - (MainThread) - INFO     - gnuplot␣
↪stdout: None
2019-10-31 14:11:29,741 - gnuplot.py - 86631 - (MainThread) - INFO     - Writing␣
↪gnuplot data "process_C.log" in path tmp/gnuplot_process
2019-10-31 14:11:29,782 - gnuplot.py - 86631 - (MainThread) - INFO     - gnuplot␣
↪stdout: None
```

In the output directory there is the log data extracted as `.dat` file, the Gnuplot specification as `.plt` file, and the plot itself in SVG as `process_C.log.svg`:

```
$ ls -l tmp/gnuplot_process/
total 360
-rw-r--r--  1 xxxxxxxx  staff  13679 31 Oct 14:11 process_C.log.dat
-rw-r--r--  1 xxxxxxxx  staff   1067 31 Oct 14:11 process_C.log.plt
-rw-r--r--@ 1 xxxxxxxx  staff  30878 31 Oct 14:11 process_C.log.svg
-rw-r--r--  1 xxxxxxxx  staff  32100 31 Oct 14:11 test.svg
```

Here is `process_C.log.svg`:



Adding Events as Graph Labels

You can also inject events into *TotalDepth.common.process* as string messages and these will be reproduced on the plot as labels. So adding one line of code:

```
import random
import time

from TotalDepth.common import process

with process.log_process(0.1):
    for i in range(8):
        size = random.randint(128, 128 + 256) * 1024 ** 2
        process.add_message_to_queue(f'String of {size:,d} bytes')
        s = ' ' * (size)
        # Simulate 0.5 to 1.5 seconds of work.
        time.sleep(0.5 + random.random())
        del s
        # Simulate 0.25 to 0.75 seconds of work.
        time.sleep(0.25 + random.random() / 2)
```

Adds that label into the JSON on the next write:

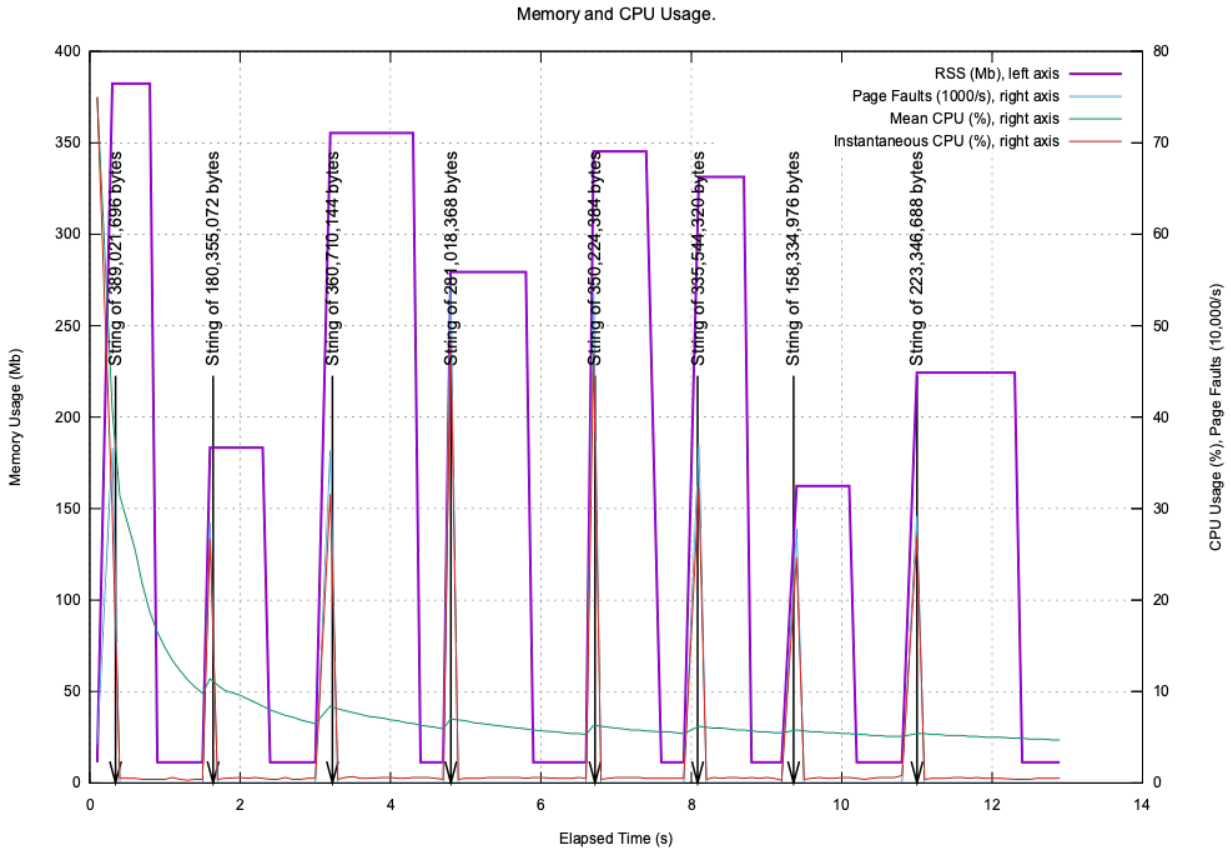
```
{
    "label": 'String of xxx bytes'
```

(continues on next page)

(continued from previous page)

```
    # ...
}
```

When plotted these labels appear on the plot:

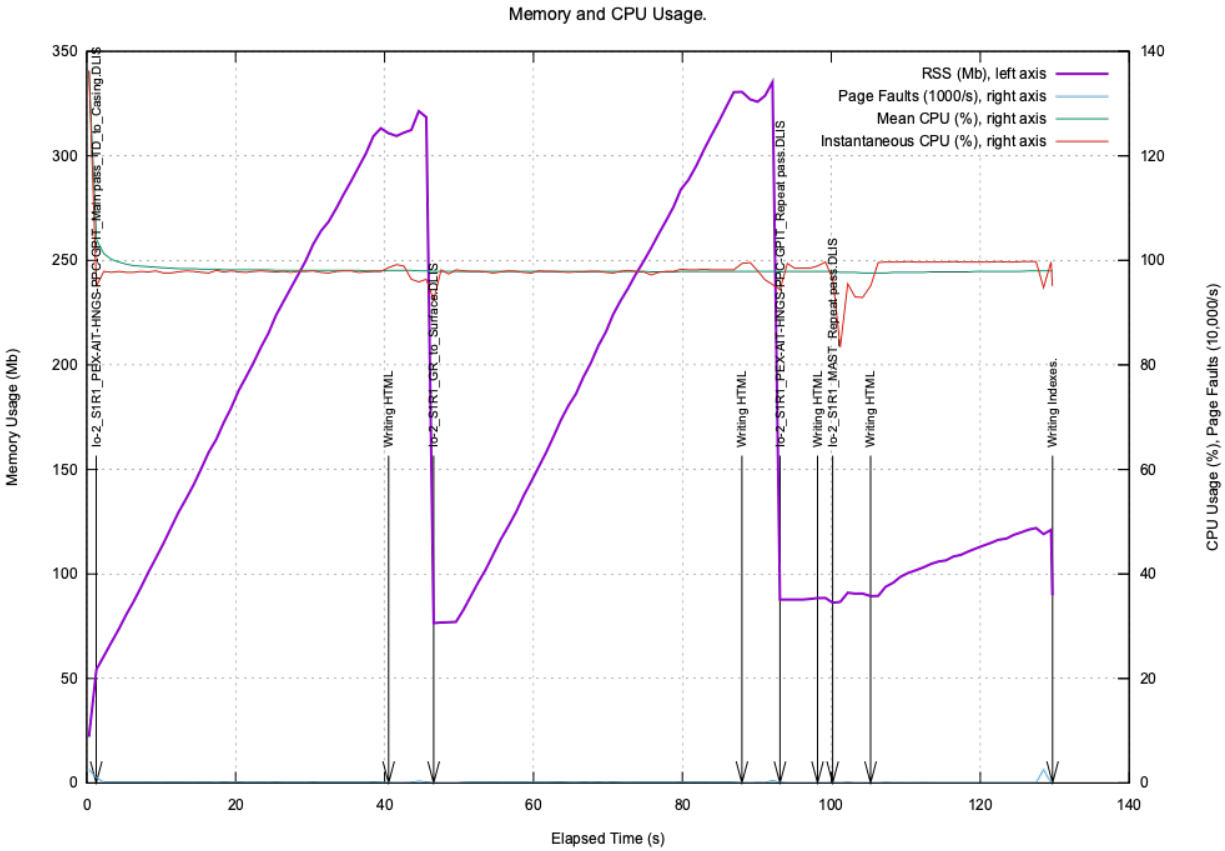


A Real World Example

Here is an example of running `TotalDepth.RP66V1.ScanHTML` on four files of sizes 75, 80, 8 and 500 MB. `TotalDepth.RP66V1.ScanHTML` essentially does two things:

- Creates an index of the RP66V1 file.
- Then iterates across that index writing HTML, this includes reading a (potentially) large number of frames depending on the file.

The start points of these operations are labeled on the graph.



The graphs clearly shows that for the last file reading the index is very quick but writing the HTML is comparatively slow. This is because that is an unusual file that deserves further investigation.

1.7.11 Performance Graphs via Gnuplot with `TotalDepth.util.gnuplot`

Todo: Finish this.

1.7.12 Total Depth and Units

This describes how TotalDepth handles unit conversions.

LIS Units

LIS units are identified by a four character, uppercase ASCII string. TotalDepth supports all known LIS units with the exchange values hard coded in *`TotalDepth.LIS.core.Units`* The API is described there.

References:

LIS unit conversion: *`TotalDepth.LIS.core.Units`*

General Units

Clearly hard coded data is somewhat limiting. RP66V2 specifies a unit convention whereby fundamental units can be combined, for example 'm/s' to 'ft/min', then the conversion factors could be calculated parsing those strings.

TotalDepth does not do this, instead opting for a short cut of pre-built lookup tables. This uses an online source of unit conversion, the primary source is Schlumberger's Oilfield Services Data Dictionary (OSDD)

This essentially provides tables such as:

```
Code    Name                Standard Form   Dimension   Scale                Offset
-------------------------------------------------------------------------------
DEGC    'degree celsius'    degC            Temperature 1                    -273.15
DEGF    'degree fahrenheit' degF            Temperature 0.555555555555556    -459.67
DEGK    'kelvin'            K               Temperature 1                    0
DEGR    'degree rankine'    degR            Temperature 0.555555555555556    0
```

So conversion from, say DEGC to DEGF of 0.0 is:

```
((value - DEGC.offset) * DEGC.scale) / DEGF.scale + DEGF.offset

((0.0 - -273.15) * 1.0) / 0.555555555555556 + -459.67 == 32.0
```

Unit Code

The unit is uniquely identified by the code, for example 'DEGC'. This is almost always the identifier in log data.

Standard Form

The 'standard form' is the RP66V2 definition of the name of the units. This can have multiple codes depending on the origin of the data. For 'degC' this is:

```
Code    Name                      Standard Form   Dimension   Scale                ␣
↪Offset
-------------------------------------------------------------------------------------
↪---
DEGC    'degree celsius'          degC            Temperature 1                     -
↪273.15
deg C   'degree celsius'          degC            Temperature 1                     -
↪273.15
oC      'GeoFrame legacy unit'    degC            Temperature 1                     -
↪273.15
```

Internal Representation of the Data

Each unit conversion is represented by a Unit class: *TotalDepth.common.units.Unit*. The entire conversion table is represented internally by a `Dict[str, Unit]` where the key is the Unit Code. At runtime the OSDD web page is read, parsed and the internal data structure cached. This happens the first time any of the APIs is accessed. On demand a data structure `typing.Dict[str, typing.List[str]]` mapping Standard Form to a list of Unit Codes is created and cached.

If the OSDD web page can not be read, perhaps because the user is offline, the code falls back to reading a static JSON file at `src/TotalDepth/common/data/osdd_units.json` that contains the last version of

the OSDD.This file is refreshed every time the integration test `tests.integration.common.test_units.test_slb_units_write_to_json()` is run.

Units Conversion

The simplest form of conversion is:

```
from TotalDepth.common import units

unit_from = units.slb_units('DEGC')
unit_to = units.slb_units('DEGF')
result = units.convert(100.0, unit_from, unit_to)  # Gives 212.0
```

To convert all the values in a numpy array inplace there is `units.convert_array(array: np.ndarray, unit_from: Unit, unit_to: Unit)`.

References:

Unit conversion: *TotalDepth.common.units*

1.7.13 Schlumberger Mnemonics

TotalDepth provides a Python API to the Schlumberger's Oilfield Services Data Dictionary (OSDD)

To quote the site:

This evergreen database delivers descriptions of more than 50,000 Schlumberger logging tools, analytical software, and log curves and parameters. It also provides definitions of physical property measurements and relevant units of measurement. Special tables enumerate mineral properties and depositional environments.

TotalDepth prvides Python APIs to access this data online. The values are cached so that repeated calls do not refer to the website. Here are the *TotalDepth.common.lookup_mnemonic* APIs:

Table 5: **Lookup Mnemonic API**

Function	Argument	Return Value	Typical Cache Size
slb_data_channel()	A data channel name from a logging tool.	*Channel*	128
slb_parameter()	A well site parameter.	*Parameter*	256
slb_logging_tool()	A logging tool name.	*LoggingTool*	64

Examples:

```
>>> from TotalDepth.common import lookup_mnemonic

>>> lookup_mnemonic.slb_data_channel('RHOB')
Channel(
    channel='RHOB',
    description='Bulk Density',
    unit_quantity='Density',
    property='Bulk_Density',
    related_tools=(
        ToolDescription(tool='ADN', description='Bulk Density'),
        ToolDescription(tool='ADN4', description='Bulk Density'),
        ToolDescription(tool='ADN6C', description='Bulk Density'),
        ToolDescription(tool='ADN8', description='Bulk Density'),
```

(continues on next page)

(continued from previous page)

```
        ToolDescription(tool='APS-C', description='Bulk Density'),
        ToolDescription(tool='DV6MT', description='Bulk Density'),
        ToolDescription(tool='ECO6', description='Bulk Density'),
        ToolDescription(tool='FGTC', description='Bulk Density'),
        ToolDescription(tool='HLDS-D', description='Bulk Density'),
        ToolDescription(tool='HLDT', description='Bulk Density'),
        ToolDescription(tool='HLDTA', description='Bulk Density'),
        ToolDescription(tool='ILDT-A', description='Bulk Density'),
        ToolDescription(tool='ILDT-B', description='Bulk Density'),
        ToolDescription(tool='LDS-C', description='Bulk Density'),
        ToolDescription(tool='LDT', description='Bulk Density'),
        ToolDescription(tool='LDTA', description='Bulk Density'),
        ToolDescription(tool='LDTC', description='Bulk Density'),
        ToolDescription(tool='LDTD', description='Bulk Density'),
        ToolDescription(tool='PGT', description='Bulk Density'),
        ToolDescription(tool='PGTE', description='Bulk Density'),
        ToolDescription(tool='PGTK', description='Bulk Density'),
        ToolDescription(tool='SADN8', description='Bulk Density'),
        ToolDescription(tool='TBT-A', description='Bulk Density'),
        ToolDescription(tool='V475', description='Bulk Density')
    ),
    related_products=(
        ProductDescription(product='CMR_DMRP', description='Bulk Density'),
        ProductDescription(product='CMR_DMRRXO', description='Bulk Density'),
        ProductDescription(product='ELAN-PLUS', description='Bulk Density'),
        ProductDescription(product='GEOSHARE', description='Bulk Density'),
        ProductDescription(product='HISTEC', description='Bulk Density'),
        ProductDescription(product='IESX', description='Bulk Density'),
        ProductDescription(product='IMPACT', description='Bulk Density'),
        ProductDescription(product='INVASION_FACTOR', description='Bulk Density'),
        ProductDescription(product='PETROSONIC', description='Bulk Density'),
        ProductDescription(product='POR_TX', description='Bulk Density'),
        ProductDescription(product='PREPLUS2_EC', description='Bulk Density'),
        ProductDescription(product='ROCKSOLID', description='Bulk Density'),
        ProductDescription(product='RUNIT_SYNTHETICS', description='Bulk Density'),
        ProductDescription(product='RWA_CLAY_CORR', description='Bulk Density'),
        ProductDescription(product='RWAC', description='Bulk Density'),
        ProductDescription(product='SCQ', description='Bulk Density'),
        ProductDescription(product='SONFRAC', description='Bulk Density'),
        ProductDescription(product='STPERM', description='Bulk Density')
    )
)

>>> lookup_mnemonic.slb_parameter('LATI')
Parameter(
    code='LATI',
    description='Latitude',
    unit_quantity='Dimensionless',
    property='Latitude',
    related_products=(
        ProductDescription(product='CSUD_WSD', description='Latitude'),
        ProductDescription(product='MAXIS_WSD', description='Latitude')
    )
)

>>> lookup_mnemonic.slb_logging_tool('HDT')
LoggingTool(
```

(continues on next page)

(continued from previous page)

```
    code='HDT',
    technology='Dipmeter',
    discipline='Geology',
    method='WIRELINE',
    description='High Resolution Dipmeter Tool',
    related_channels=(
        ChannelDescription(channel='AZIM', description='Measured Azimuth'),
        ChannelDescription(channel='AZIX', description='Azimuth of Reference Sensor␣
↪(Pad 1, if available)'),
        ChannelDescription(channel='C1', description='Caliper 1'),
        ChannelDescription(channel='C2', description='Caliper 2'),
        ChannelDescription(channel='DEVI', description='Hole Deviation'),
        ChannelDescription(channel='EWDR', description='East West Drift Component'),
        ChannelDescription(channel='FC0', description='HDT Fast Channel 0 (Speed␣
↪Button)'),
        ChannelDescription(channel='FC1', description='HDT Fast Channel 1'),
        ChannelDescription(channel='FC2', description='HDT Fast Channel 2'),
        ChannelDescription(channel='FC3', description='HDT Fast Channel 3'),
        ChannelDescription(channel='FC4', description='HDT Fast Channel 4'),
        ChannelDescription(channel='FEP', description='Far Electrode Potential'),
        ChannelDescription(channel='FEP1', description='Far Electrode Potential 1'),
        ChannelDescription(channel='FEP2', description='Far Electrode Potential 2'),
        ChannelDescription(channel='HAZI', description='Hole Azimuth Relative to True␣
↪North'),
        ChannelDescription(channel='NSDR', description='North South Drift'),
        ChannelDescription(channel='P1AZ', description='Pad 1 Azimuth in Horizontal␣
↪Plane (0 = True North)'),
        ChannelDescription(channel='PP', description='Pad Pressure'),
        ChannelDescription(channel='RAZI', description='Raw Azimuth'),
        ChannelDescription(channel='RB', description='Relative Bearing'),
        ChannelDescription(channel='RC', description='Reference Check'),
        ChannelDescription(channel='RC1', description='Raw C1'),
        ChannelDescription(channel='RC2', description='Raw C2'),
        ChannelDescription(channel='RDEV', description='Raw Deviation'),
        ChannelDescription(channel='REFE', description='Reference'),
        ChannelDescription(channel='RHDT', description='Raw HDT Data Block'),
        ChannelDescription(channel='RRB', description='Raw Relative Bearing'),
        ChannelDescription(channel='SDEV', description='Sonde Deviation'),
        ChannelDescription(channel='TEMP', description='Computed Borehole Temperature
↪'),
        ChannelDescription(channel='ZB', description='Zero Button')
    ),
    related_parameters=(
        ParameterDescription(parameter='AMOD', description='Averaging Mode Selection
↪'),
        ParameterDescription(parameter='AZIM', description='Well Section Azimuth'),
        ParameterDescription(parameter='DANG', description='Dip Angle of the Bedding
↪'),
        ParameterDescription(parameter='DAZI', description='Dip Azimuth of the Bedding
↪'),
        ParameterDescription(parameter='DEVI', description='Well Section Deviation'),
        ParameterDescription(parameter='DISO', description='DIP Information Source'),
        ParameterDescription(parameter='HDTT', description='HDT Sonde Type'),
        ParameterDescription(parameter='LATD', description='Latitude (N=+ S=-)'),
        ParameterDescription(parameter='LOND', description='Longitude'),
        ParameterDescription(parameter='MCT', description='Mechanical Cartridge Type
↪'),
```

(continues on next page)

(continued from previous page)

```
        ParameterDescription(parameter='MDEC', description='Magnetic Field Declination
↪'),
        ParameterDescription(parameter='MFIN', description='Magnetic Field Intensity
↪'),
        ParameterDescription(parameter='MINC', description='Magnetic Field Inclination
↪'),
        ParameterDescription(parameter='MTD', description='Measured Tie Depth'),
        ParameterDescription(parameter='RANO', description='Resistivity Anomaly␣
↪Selection'),
        ParameterDescription(parameter='SFAN', description='Scale Factor of Anomalies
↪'),
        ParameterDescription(parameter='STYP', description='Sonde Type')
    )
)
```

References:

Unit conversion: *TotalDepth.common.lookup_mnemonic*

1.8 Testing TotalDepth

Contents:

1.8.1 Unit Tests

This describes how unit testing is done in TotalDepth.

TotalDepth uses the following test frameworks:

- pytest - This is a basic minimum.
- pytest-cov - Pytest's wrapper around Ned Batchelor's *excellent* coverage tool.
- pytest-benchmark - For micro benchmarks.
- Airspeed Velocity - For timeline benchmarks.

Basic Testing

Simply:

```
$ pytest tests/
```

This should take only a few tens of seconds. If you include the slow tests with `--runslow` this will take many minutes.

Testing With Test Coverage

For complete coverage you need to run the slow tests and with the following command:

```
$ pytest --cov=TotalDepth --cov-report html --runslow tests
```

1.8.2 Testing the Plot Package

This is a comprehensive testing of your installation to see if LIS/LAS files can be written, read and plotted. The code writes specific LIS/LAS files in memory, reads them back with the appropriate parser then plots them as SVG files.

Running the test

TestPlot is a unit test which also has performance tests within it. It is invoked thus:

```
$ python tests/unit/test_util/test_plot/TestPlot.py
```

The response should be something like:

```
TestClass.py script version "0.1.0", dated 2010-08-02
Author: Paul Ross
Copyright (c) Paul Ross

test_00 (__main__.TestPlotRollStatic)
TestPlotRollStatic.test_00(): Tests setUp() and tearDown(). ... ok
test_01 (__main__.TestPlotRollStatic)
TestPlotRollStatic.test_01(): viewBox. ... ok

... 8<-------- snip, snip, snip -------->8

test_01 (__main__.TestPlotReadLIS_SingleSinCurve) ...  ExecTimerList [6]:
 Loading FrameSet          Size:    0.002 (MB) Time:    0.010 (s) Cost:   3838.393␣
↪(ms/MB)
 Initialising LIS plot     Size:    0.000 (MB) Time:    0.001 (s) Cost:        N/A␣
↪(ms/MB)
 Plotting Tracks           Size:    0.000 (MB) Time:    0.005 (s) Cost:        N/A␣
↪(ms/MB)
 Plotting XGrid            Size:    0.000 (MB) Time:    0.023 (s) Cost:        N/A␣
↪(ms/MB)
 Plotting scales (legends) Size:    0.000 (MB) Time:    0.013 (s) Cost:        N/A␣
↪(ms/MB)
 Plotting curves           Size:    0.002 (MB) Time:    0.141 (s) Cost:   9338.957␣
↪(ms/MB)  ok
test_00 (__main__.TestPlotReadLIS_SingleSquareCurveLowFreq)
TestPlotReadLIS_SingleSquareCurve.test_00(): Tests setUp() and tearDown(). ... ok

... 8<-------- snip, snip -------->8

test_12 (__main__.TestPlotReadLAS_XML_LgFormat)
TestPlotReadLAS_XML_LgFormat.test_12(): Plot from XML LgFormat files - density,␣
↪porosity and multiple gamma ray curves. ...  ExecTimerList [6]:
 Initialising LAS plot: "Porosity_GR_3Track" Size:    0.000 (MB) Time:    0.001 (s)␣
↪Cost:        N/A (ms/MB)
 Plotting API Header                         Size:    0.000 (MB) Time:    0.016 (s)␣
↪Cost:        N/A (ms/MB)
```

(continues on next page)

(continued from previous page)

```
 Plotting Tracks                                Size:    0.000 (MB) Time:    0.005 (s)␣
↪Cost:        N/A (ms/MB)
 Plotting XGrid                                 Size:    0.000 (MB) Time:    0.055 (s)␣
↪Cost:        N/A (ms/MB)
 Plotting scales (legends)                      Size:    0.000 (MB) Time:    0.013 (s)␣
↪Cost:        N/A (ms/MB)
 Plotting curves                                Size:    0.020 (MB) Time:    0.096 (s)␣
↪Cost:   5009.433 (ms/MB)  ok

----------------------------------------------------------------------
Ran 107 tests in 34.318s

OK
CPU time =   34.179 (S)
Bye, bye!
```

The important thing is that there should be no reported failures.

Results

You should find in the directory *<TOTALDEPTH>*`/tests/unit/test_util/test_plot/test_svg` a set of test plots.

Index

This directory has an *<TOTALDEPTH>*`/tests/unit/test_util/test_plot/test_svg/index.html` that looks like this:

file://localhost/Users/paulross/Documents/workspace/TotalDepth/src/TotalDepth/util/plot/test_s...

API Headers

1. link TestLogHeaderLIS.test_05(): Empty API header from LIS data (upright).
2. link TestLogHeaderLIS.test_06(): Empty API header from LIS data (rotated).
3. link TestLogHeaderLIS.test_10(): API header with CONS information from LIS data (upright).
4. link TestLogHeaderLIS.test_11(): API header with CONS information from LIS data (rotated).
5. link TestLogHeaderLAS.test_05(): Empty API header from LAS data (upright).
6. link TestLogHeaderLAS.test_06(): Empty API header from LAS data (rotated).
7. link TestLogHeaderLAS.test_10(): API header with CONS information from LAS data (upright).
8. link TestLogHeaderLAS.test_11(): API header with CONS information from LAS data (rotated).

Plots from LIS files

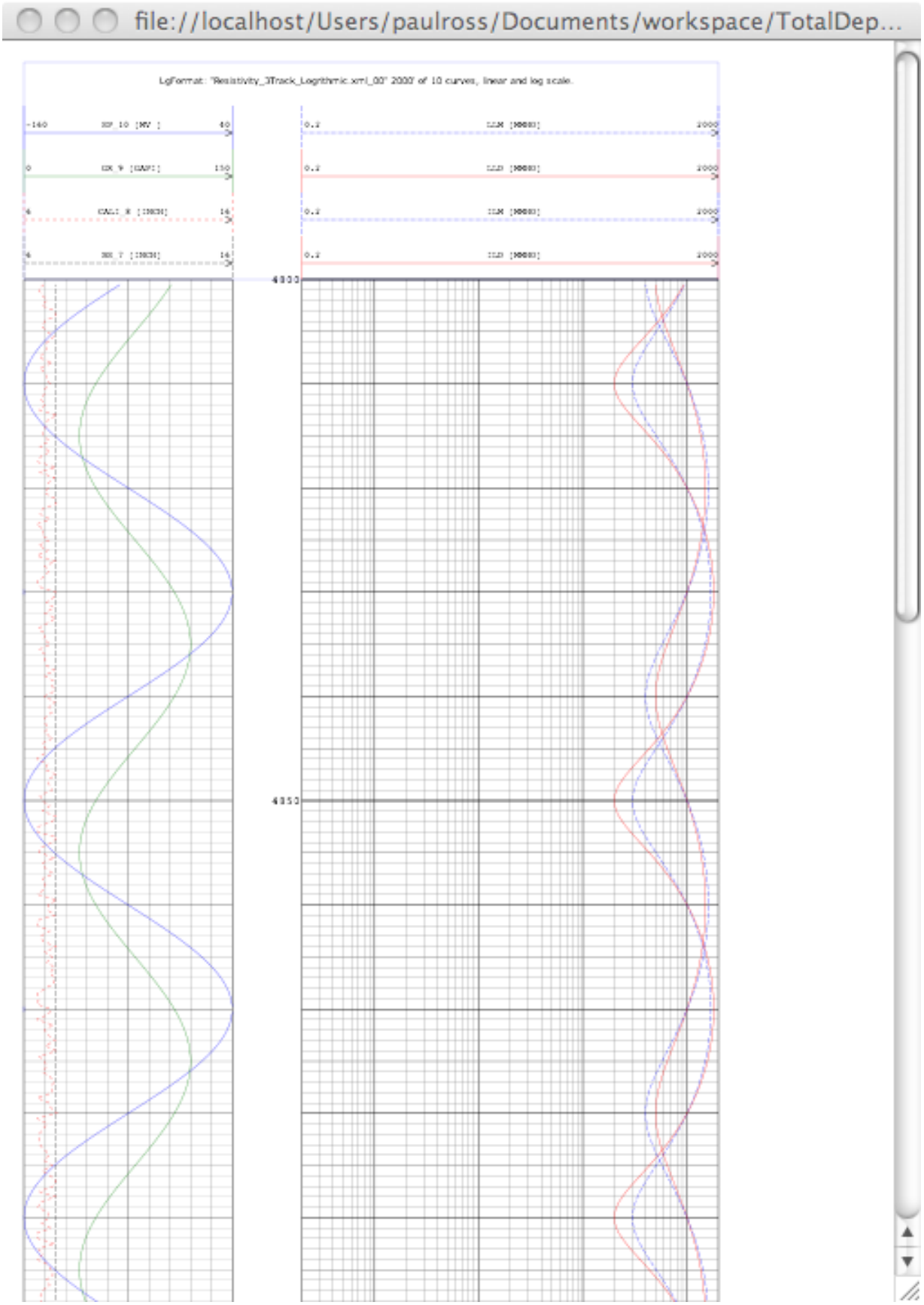
1. link TestPlotReadLIS_SingleSinCurve.test_01(): Sinusoidal SP plotted on a number of different scales.
2. link TestPlotReadLIS_SingleSquareCurveLowFreq.test_01(): Square wave with 4' spacing to check wrap interpolation.
3. link TestPlotReadLIS_SingleSquareCurveHighFreq.test_01(): Square wave with 0.5' spacing to check wrap interpolation.
4. link TestPlotReadLIS_HDT.test_01(): 200 feet of HDT.
5. link TestPlotReadLIS_SuperSampled.test_01(): Channels at 4' frame spacing, single, x8, x32 super-sampling.
6. link TestPlotReadLIS_COLO_Named.test_01(): Sinusoidal SP plotted on a number of different named colours.
7. link TestPlotReadLIS_COLO_Numbered.test_01(): Numbered colours 400 (red), 040 (green), 004 (blue).
8. link TestPlotReadLIS_COLO_Numbered_Comp.test_01(): Numbered colours 440 (yellow), 404 (magenta), 044 (cyan).
9. link TestPlotReadLIS_Perf_00.test_01(): Film 1 2000' of 10 curves, linear scale.
10. link TestPlotReadLIS_Perf_00.test_02(): Film 2 2000' of 10 curves, linear and log scale.
11. link LgFormat: "Triple_Combo_00" 2000 of 10 curves, linear and log scale.
12. link LgFormat: "Resistivity_3Track_Logrithmic.xml_00" 2000' of 10 curves, linear and log scale.
13. link TestPlotReadLIS_SingleSinCurve.test_01(): Sinusoidal SP plotted on a number of different scales with API header.

Plots from LAS files

1. link LgFormat: "Triple_Combo" 200 of 15 curves, linear and log scale from LAS file, DOWN log, no header.
2. link LgFormat: "Resistivity_3Track_Logrithmic.xml" 200' of 15 curves, linear and log scale from LAS file, DOWN log, no header.
3. link LgFormat: "Triple_Combo" 200 of 10 curves, linear and log scale from LAS file, DOWN log, with header.
4. link LgFormat: "Resistivity_3Track_Logrithmic.xml" 200' of 15 curves, linear and log scale from LAS file, DOWN log, with header.
5. link LgFormat: "Triple_Combo" 200 of 15 curves, linear and log scale from LAS file, UP log, no header.
6. link LgFormat: "Resistivity_3Track_Logrithmic.xml" 200' of 15 curves, linear and log scale from LAS file, UP log, no header.
7. link LgFormat: "Triple_Combo" 200 of 15 curves, linear and log scale from LAS file, UP log, with header.
8. link LgFormat: "Resistivity_3Track_Logrithmic.xml" 200' of 15 curves, linear and log scale from LAS file, UP log, with header.
9. link LgFormat: "Triple_Combo" 1000 of 15 curves, linear and log scale from LAS file, large down log, with header.
10. link LgFormat: "Resistivity_3Track_Logrithmic.xml" 1000' of 15 curves, linear and log scale from LAS file, large down log, with header.
11. link LgFormat: "Triple_Combo" 100 feet of 5 gamma ray curves.
12. link LgFormat: "Resistivity_3Track_Logrithmic.xml" 100 feet of 5 gamma ray curves.
13. link LgFormat: "Porosity_GR_3Track" 100 feet of Density, porosity and 5 gamma ray curves.

Example Plot

Navigate to a typical test LIS plot from that index such as:


file://localhost/Users/paulross/Documents/workspace/TotalDep...
LgFormat: 'Resistivity_3Track_Logrithmic.xml_00' 2000' of 10 curves, linear and log scale.
4900
4950

Cool. If you see that then your TotalDepth installation is pretty good!

1.8.3 Testing TotalDepth With Recorded Data

TotalDepth is tested against a diverse data set of real world files. The test set is split into small/medium/large datasets.

Small Test Set

todo Complete this

Medium Test Set

The Medium Size Test Set is 20,000+ files (130Gb+) of typical oilfield data. Here is the approximate breakdown of the test set:

File Type	Files	Total Size	Notes
BIT	~500	~1.5Gb	Largest file is around 6Mb.
LAS v1.2	~500	~1Gb	Largest file is around 16Mb.
LAS v2.0	~20,000	~30Gb	Largest file is around 250Mb (RP66V1 converted files are much larger).
LAS v3.0	A few	~0	Rarely present, their absence is not considered significant.
LIS	~2000	~2GB	Largest file is around 60Mb. Around half have TIF markers.
DLIS (RP66V1)	~800	~100GB	Largest file is around 4GB. About one quarter are corrupted by TIF markers.
DLIS (RP66V2)	0	0	Not present, their absence is not considered significant.
Other	Various	Various	Various files such as PDF, TIFF, miscellaneous binary files and unstructured ASCII. If present then their contents is not considered significant but archives containing these should be processed without drama.

The layout of the test set is typical of an oilfield repository, typically by well, with a well having an unspecified directory structure and a mix of file types in each directory.

The aim is that TotalDepth can process 97+% of the files in this archive in the formats that TotalDepth supports.

Large Test Set

todo Complete this

Synthetic Test Set

TotalDepth can generate arbitrary sized files in a number of formats and this test set is used for performance analysis. This test set is not distributed with TotalDepth but the means to create it is.

1.9 API Reference

Contents:

1.9.1 TotalDepth.DeTif

Strip TIF markers from a file or scan a file and reporting errors in TIF markers.

exception TotalDepth.DeTif.**DeTifException**

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

exception TotalDepth.DeTif.**DeTifExceptionCompare**

exception TotalDepth.DeTif.**DeTifExceptionRead**

class TotalDepth.DeTif.**TifMarker**(*tell*, *type*, *prev*, *next*)

property tell
Alias for field number 0

property type
Alias for field number 1

property prev
Alias for field number 2

property next
Alias for field number 3

__str__() → str
Return str(self).

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *tell: int*, *type: int*, *prev: int*, *next: int*)
Create new instance of TifMarker(tell, type, prev, next)

__repr__()
Return a nicely formatted representation string

TotalDepth.DeTif.**compare_next**(*this: TotalDepth.DeTif.TifMarker*, *nxt: TotalDepth.DeTif.TifMarker*) → None
Compares two adjacent TIF markers for consistency.

TotalDepth.DeTif.**has_tif_file**(*fobj: BinaryIO*) → bool
Returns True of the file in path apparently has TIF markers.

TotalDepth.DeTif.**has_tif**(*path: str*) → bool
Returns True of the file in path apparently has TIF markers.

TotalDepth.DeTif.**tif_scan_file_object**(*fobj: BinaryIO*) → List[TotalDepth.DeTif.TifMarker]
Scan a file object and return the list of TIF markers.

TotalDepth.DeTif.**tif_scan_path**(*path: str*) → List[TotalDepth.DeTif.TifMarker]
Scan a file at path and return the list of TIF markers.

TotalDepth.DeTif.**get_errors**(*tifs: List[TotalDepth.DeTif.TifMarker], file_size: int*) → List[str]
Return a list of TIF marker errors.

TotalDepth.DeTif.**strip_tif**(*file_in: BinaryIO*, *file_out: BinaryIO*) → Tuple[int, int]
Read file_in then strip TIF markers and write to file_out. The only error detected is negative reads. Returns a tuple of (tif_markers_stripped, bytes_written).

TotalDepth.DeTif.**strip_path**(*path_in: str*, *path_out: str*) → Tuple[int, int]
Read path_in then strip TIF markers and write to path_out.

TotalDepth.DeTif.**de_tif_file**(*path_in: str*, *path_out: str*, *nervous: bool*, *over_write: bool*) → Tuple[int, int, int]
De-TIFs a file path_in writing to path_out. Returns a tuple of (files_copied, tif_count, byte_count) where files_copied is 0 or 1, tif_count is the number of 12 byte TIF markers and byte_count is the number of bytes in path_in (if no file written) or path_out if the file is written.

1.9.2 TotalDepth.PlotLogs

Created on Jan 24, 2012

@author: p2ross

class TotalDepth.PlotLogs.**IndexTableValue**(*scale*, *evFirst*, *evLast*, *evInterval*, *curves*, *numPoints*, *outPath*)

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *scale*, *evFirst*, *evLast*, *evInterval*, *curves*, *numPoints*, *outPath*)
Create new instance of IndexTableValue(scale, evFirst, evLast, evInterval, curves, numPoints, outPath)

__repr__()
Return a nicely formatted representation string

property curves
Alias for field number 4

property evFirst
Alias for field number 1

property evInterval
Alias for field number 3

property evLast
Alias for field number 2

property numPoints
Alias for field number 5

property outPath
Alias for field number 6

property scale
Alias for field number 0

class TotalDepth.PlotLogs.**PlotLogInfo**
Class that collates information about the results of plotting log passes. This can, for example, write out an index.html page with links to SVG pages.

__init__()
Initialize self. See help(type(self)) for accurate signature.

__str__()
Return str(self).

addPlotResult(*theInPath*, *theOutPath*, *theLpIdx*, *theFilmID*, *theScale*, *theEvFirst*, *theEvLast*, *theCurveS*, *ptsPlotted*)
Adds a successful plot. theInPath - The file path to the input file. theOutPath - The file path to the output file. theLpIdx - Integer index of the LogPass in the input file. theFilmID - The FILM ID as a Mnem. theScale - Plot scale as an number. theEvFirst - The first X axis as an EngVal. theEvLast - The last X axis as an EngVal. theCurveS - A list of Mnem of the curves plotted. ptsPlotted - Number of points plotted.

writeHTML(*theFilePath*, *theDesc*)
Write the index.html table.

retRelPath(*d*, *f*)
Given directory d and file path of f this returns relative path to f from d.

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

class TotalDepth.PlotLogs.**PlotLogPasses**(*fpIn*, *fpOut*, *opts*)
Takes an input path, output path and generates SVG file(s) from LIS.

__init__(*fpIn*, *fpOut*, *opts*)
Constructor.

fpIn and fpOut are file or directory paths. fpOut will be created if necessary.

recursive is a flag to control directory recursion.

keepGoing is a flag passed to the LIS File.FileRead object.

lgFormatS is a list of strings the correspond to the LgFormat UniqueId XML attribute. If absent the LIS file FILM/PRES etc. tables are used.

apiHeader is a flag to control whether a API header is extracted from CONS tables is to be plotted on the top of the log.

property usesInternalRecords
True if we are going to use the internal records of a file to describe the plot. False if we are going to use external records, such as LgFormat XML files to describe the plot.

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

TotalDepth.PlotLogs.**plotLogPassesMP**(*dIn*, *dOut*, *opts*)
Multiprocessing code to plot log passes. Returns a PlotLogInfo object.

1.9.3 TotalDepth.BIT API Reference

Contents:

TotalDepth.BIT.ReadBIT

This processes Dresser Atlas BIT files. Dresser Atlas BIT files are TIF encoded and consist of a set of Log Passes:

- First block, this gives a description of the file.
- Subsequent blocks are frame data.

TIF markers are used to separate Log Passes and delineate the file. A TIF marker type 1 ends the Log Pass and a pair of type 1 markers ends the readable file.

Here is an example of TIF markers in a file:

```
$ tddetif -n data/DresserAtlasBIT/special/29_10-_3Z/DWL_FILE/29_10-_3Z_dwl_DWL_WIRE_
↪1644659.bit -v
Cmd: DWL_FILE/29_10-_3Z_dwl_DWL_WIRE_1644659.bit -v
Detected 187 TIF Markers in data/DresserAtlasBIT/special/29_10-_3Z/DWL_FILE/29_10-_3Z_
↪dwl_DWL_WIRE_1644659.bit
[     0] TifMarker: 0x00000000 Type: 0x00000000 Prev: 0x00000000 Next: 0x00000120␣
↪Length: 0x00000120 Payload: 0x00000114
[     1] TifMarker: 0x00000120 Type: 0x00000000 Prev: 0x00000000 Next: 0x000003ac␣
↪Length: 0x0000028c Payload: 0x00000280
[     2] TifMarker: 0x000003ac Type: 0x00000000 Prev: 0x00000120 Next: 0x00000638␣
↪Length: 0x0000028c Payload: 0x00000280
...
[    91] TifMarker: 0x0000e658 Type: 0x00000000 Prev: 0x0000e3cc Next: 0x0000e8e4␣
↪Length: 0x0000028c Payload: 0x00000280
[    92] TifMarker: 0x0000e8e4 Type: 0x00000000 Prev: 0x0000e658 Next: 0x0000eb70␣
↪Length: 0x0000028c Payload: 0x00000280
[    93] TifMarker: 0x0000eb70 Type: 0x00000001 Prev: 0x0000e8e4 Next: 0x0000eb7c␣
↪Length: 0x0000000c Payload: 0x00000000
[    94] TifMarker: 0x0000eb7c Type: 0x00000000 Prev: 0x0000eb70 Next: 0x0000ec9c␣
↪Length: 0x00000120 Payload: 0x00000114
[    95] TifMarker: 0x0000ec9c Type: 0x00000000 Prev: 0x0000eb7c Next: 0x0000ef28␣
↪Length: 0x0000028c Payload: 0x00000280
...
[   184] TifMarker: 0x0001cf48 Type: 0x00000000 Prev: 0x0001ccbc Next: 0x0001d1d4␣
↪Length: 0x0000028c Payload: 0x00000280
[   185] TifMarker: 0x0001d1d4 Type: 0x00000001 Prev: 0x0001cf48 Next: 0x0001d1e0␣
↪Length: 0x0000000c Payload: 0x00000000
[   186] TifMarker: 0x0001d1e0 Type: 0x00000001 Prev: 0x0001d1d4 Next: 0x0001d1ec␣
↪Length: 0x0000000c Payload: 0x00000000
Execution time =    0.020 (S)
Bye, bye!
```

Example first block without TIF markers, length 276 (0x114) bytes:

```
0000000c: 0002 0000 5348 454c 4c20 4558 5052 4f20  ....SHELL EXPRO
0000001c: 552e 4b2e 2020 2020 2020 3234 204f 4354  U.K.      24 OCT
0000002c: 2038 3420 2020 2020 204d 414e 5346 4945   84      MANSFIE
0000003c: 4c44 2f44 4f44 4453 2020 2020 2020 2020  LD/DODDS
0000004c: 2020 2020 2020 2020 2020 2020 000a 0018              ....
0000005c: 0054 2020 3220 3920 2f20 3120 3020 2d20  .T  2 9 / 1 0 -
0000006c: 3320 2020 2020 2020 2020 2020 2020 2020  3
```

(continues on next page)

(continued from previous page)

```
0000007c: 2020 2020 2020 2020 2020 2020 2020 2020
0000008c: 2020 2020 2020 2020 2020 2020 2020 2020
0000009c: 2020 2020 2020 2020 2020 2020 0012 000b                ....
000000ac: 0006 2020 000a 0000 434f 4e44 534e 2020  ..  ....CONDSN
000000bc: 5350 2020 4752 2020 4341 4c20 5445 4e20  SP  GR  CAL TEN
000000cc: 5350 4420 4143 5120 4143 2020 5254 2020  SPD ACQ AC  RT
000000dc: 2020 2020 2020 2020 2020 2020 2020 2020
000000ec: 2020 2020 2020 2020 2020 2020 2020 2020
000000fc: 2020 2020 2020 2020 443a 6600 4438 fe00          D:f.D8..
0000010c: 4040 0000 0000 0000 4210 0000 4d4e 3233  @@......B...MN23
0000011c: 394a 2031
```

Decomposed:

- 4 bytes unknown.
- 160 bytes ASCII description, there is some structure here but it is as yet unknown.
- C, the count of channels as big endian two byte format '>H' or '>h'.
- A two byte null
- Channel names, 4 bytes each, this is always 80 bytes long.
- Five 4 byte floats start, stop, step, 0, ???.
- Unknown tail of eight bytes.

Total = 4 + 160 + 2 + 2 + 80 + 5 * 4 + 8 = 276 (0x114) bytes.

Of note is that while the TIF markers are little endian many values within the file are big endian.

Decoding the Description

This is 160 bytes long. Example:

```
$ xxd -s +16 data/DresserAtlasBIT/special/29_10-_3Z/DWL_FILE/29_10-_3Z_dwl_DWL_WIRE_
↪1644659.bit | head -n 20
00000010: 5348 454c 4c20 4558 5052 4f20 552e 4b2e  SHELL EXPRO U.K.
00000020: 2020 2020 2020 3234 204f 4354 2038 3420        24 OCT 84
00000030: 2020 2020 204d 414e 5346 4945 4c44 2f44       MANSFIELD/D
00000040: 4f44 4453 2020 2020 2020 2020 2020 2020  ODDS
00000050: 2020 2020 2020 2020 000a 0018 0054 2020          .....T
00000060: 3220 3920 2f20 3120 3020 2d20 3320 2020  2 9 / 1 0 - 3
00000070: 2020 2020 2020 2020 2020 2020 2020 2020
00000080: 2020 2020 2020 2020 2020 2020 2020 2020
00000090: 2020 2020 2020 2020 2020 2020 2020 2020
000000a0: 2020 2020 2020 2020 0012 000b 0006 2020          ......
000000b0: 000a 0000 434f 4e44 534e 2020 5350 2020  ....CONDSN  SP
```

Looks like we have 4 * 16 + 8 = 72 bytes of ASCII to 0x58 as a description.

Either:

Then 24 bytes of binary data to 0x70 ???:

```
000a 0018 0054 2020 3220 3920 2f20 3120 3020 2d20 3320 2020
```

3 * 16 + 8 = 56 ASCII spaces. hmm divided by 4 is 19. hmm maybe together 24 + 56 = 80 and 80 / 4 is 20. Channel units?

Then 8 bytes of stuff: 0012 000b 0006 2020

Total is: 72 + 24 + 56 + 8 == 160

Or, alternatively: 72 (4 * 16 + 8) bytes of ASCII. Then five bytes: 000a 0018 00 Then 75 bytes of ASCII: 54 2020 3220 3920 2f20 3120 3020 2d20 3320 2020 . . . Then 8 bytes of stuff: 0012 000b 0006 2020 Total is: 72 + 5 + 75 + 8 == 160

Decoding the frames

Each subsequent block is a subdivided into the number of channels and read into that channel. For example a block of data 0x280 (640) bytes long with 10 channels is decomposed into 10 sub-blocks of 64 bytes each. Each sub-block contains 16 floats.

There is a directory that has both BIT and LIS files in it.

LIS: 29_10-_3Z/DWL_FILE/29_10-_3Z_dwl_DWL_WIRE_1988494.lis BIT: 29_10-_3Z_dwl_DWL_WIRE_1644659.bit and 29_10-_3Z_dwl_DWL_WIRE_1644660.bit

Log Pass 0 has X axis: 14950.000 (FT) to 14582.250 (FT) Interval -367.750 (FT) Total number of frames 1472 Overall frame spacing -0.250 (FT)

Corresponding BIT file 29_10-_3Z_dwl_DWL_WIRE_1644659.bit has:

```
LogPassRange(depth_from=14950.000891089492, depth_to=14590.000869631818, spacing=0.
↪2500000149011621, unknown_a=0.0,
unknown_b=16.000000953674373)
```

Frames from spacing: 1441

A striking feature of the LIS Log Pass 0 file is that the SP is fixed throughout at -249.709. This value appears nowhere else in the LIS Log Pass. The BIT file 29_10-_3Z_dwl_DWL_WIRE_1644659.bit corresponds with the following:

- The binary data is assumed to start at 0x128 + 4
- Each channel is sequential but read in blocks of 16 floats (64 bytes).
- After 16 floats are read for each channel (every 160 floats, or 640 bytes) then 12 bytes are read and discarded.
- **Although the BIT file states 1441 frames from spacing the LIS file has read 1472 (0x5c0) frames (modulo 16) with the** remaining values as 0.0001 for all channels.

The 12 bytes read after every 640 bytes look like this: 4 nulls, two values unknown, two nulls, two values unknown, two nulls. These are TIF markers.

The 0.0001 figure is actually 9.999999615829415e-05 or 0x3d 0x68 0xdb 0x8b

exception TotalDepth.BIT.ReadBIT.**ExceptionTotalDepthBIT**
Simple specialisation of an exception class for TotalDepth.BIT.

exception TotalDepth.BIT.ReadBIT.**ExceptionTotalDepthBIT_TIF**
When TIF markers go wrong.

exception TotalDepth.BIT.ReadBIT.**ExceptionTotalDepthBITFirstBlock**
Constructor from first block of data.

exception TotalDepth.BIT.ReadBIT.**ExceptionTotalDepthBITDataBlocks**
Constructor from first block of data.

TotalDepth.BIT.ReadBIT.**bytes_to_float**(*b: bytes*) → float
Returns a float from four bytes.

https://en.wikipedia.org/wiki/IBM_hexadecimal_floating-point

Example: -118.625 -> b'Âv '.

NOTE: This is the same as RP66V1 ISINGL (5) Representation Code.

TotalDepth.BIT.ReadBIT.**float_to_bytes**(*f: float*) → bytes
Returns four bytes from a float.

https://en.wikipedia.org/wiki/IBM_hexadecimal_floating-point

Example: b'Âv ' -> -118.625

NOTE: This is the same as RP66V1 ISINGL (5) Representation Code.

TotalDepth.BIT.ReadBIT.**read_float**(*file: BinaryIO*) → float
Returns a float from the current read position.

TotalDepth.BIT.ReadBIT.**gen_floats**(*b: bytes*) → Sequence[float]
Yields a sequence of floats from the bytes.

TotalDepth.BIT.ReadBIT.**is_bit_file**(*fobj: BinaryIO*) → bool
Returns True the file looks like a Western Atlas BIT file, False otherwise.

TotalDepth.BIT.ReadBIT.**is_bit_file_from_path**(*file_path: str*) → bool
Returns True the file looks like a Western Atlas BIT file, False otherwise.

class TotalDepth.BIT.ReadBIT.**TifMarker**
Contains a TIF marker with its file position.

property tell
Alias for field number 0

property type
Alias for field number 1

property prev
Alias for field number 2

property next
Alias for field number 3

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *tell: int*, *type: int*, *prev: int*, *next: int*)
Create new instance of TifMarker(tell, type, prev, next)

__repr__()
Return a nicely formatted representation string

class TotalDepth.BIT.ReadBIT.**TifType**
Type of TIF marker. Type 0 is normal data. Type 1 is end of Log Pass. Type 2 is end of file.

class TotalDepth.BIT.ReadBIT.**TifMarkedBytes**
This is yielded by yield_tif_blocks().

property tell
Alias for field number 0

property tif_type
Alias for field number 1

property payload
Alias for field number 2

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *tell: int*, *tif_type: TotalDepth.BIT.ReadBIT.TifType*, *payload: bytes*)
Create new instance of TifMarkedBytes(tell, tif_type, payload)

__repr__()
Return a nicely formatted representation string

TotalDepth.BIT.ReadBIT.**yield_tif_blocks**(*file: BinaryIO*) → Sequence[TotalDepth.BIT.ReadBIT.TifMarkedBytes]
Generate the payload from blocks of the file.

class TotalDepth.BIT.ReadBIT.**LogPassRange**
POD container for the five floats that describe the range of the BIT Log Pass.

property depth_from
Alias for field number 0

property depth_to
Alias for field number 1

property spacing
Alias for field number 2

property unknown_a
Alias for field number 3

property unknown_b
Alias for field number 4

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *depth_from: float*, *depth_to: float*, *spacing: float*, *unknown_a: float*, *unknown_b: float*)
Create new instance of LogPassRange(depth_from, depth_to, spacing, unknown_a, unknown_b)

__repr__()
Return a nicely formatted representation string

TotalDepth.BIT.ReadBIT.**read_bytes_from_offset**(*b: bytes*, *count: int*, *offset: int*) → Tuple[bytes, int]
Slices a bytes object and increments the offset. Usage:

```
result, offset = read_bytes_from_offset(b, count, offset)
```

class TotalDepth.BIT.ReadBIT.**BITFrameArray**(*ident: str*, *tif_block: TotalDepth.BIT.ReadBIT.TifMarkedBytes*)
Represents a Log Pass from a BIT file. This has a number of fields, some are BIT specific but `self.frame_array` is a *`TotalDepth.common.LogPass.FrameArray`*.

__init__(*ident: str*, *tif_block: TotalDepth.BIT.ReadBIT.TifMarkedBytes*)
Example initial block, length 0x114, 276 bytes:

```
0000006c: 0002 0000 5348 454c 4c20 4558 5052 4f20  ....SHELL EXPRO
0000007c: 552e 4b2e 2020 2020 2020 3234 204f 4354  U.K.      24 OCT
0000008c: 2038 3420 2020 2020 204d 414e 5346 4945   84      MANSFIE
0000009c: 4c44 2f44 4f44 4453 2020 2020 2020 2020  LD/DODDS
000000ac: 2020 2020 2020 2020 2020 2020 000a 0018                ....
000000bc: 0054 2020 3220 3920 2f20 3120 3020 2d20  .T  2 9 / 1 0 -
000000cc: 3320 2020 2020 2020 2020 2020 2020 2020  3
```

(continues on next page)

(continued from previous page)

```
0000007c: 2020 2020 2020 2020 2020 2020 2020 2020
0000008c: 2020 2020 2020 2020 2020 2020 2020 2020
0000009c: 2020 2020 2020 2020 2020 2020 0012 000b                ....
000000ac: 0006 2020 000a 0000 434f 4e44 534e 2020  ..  ....CONDSN
000000bc: 5350 2020 4752 2020 4341 4c20 5445 4e20  SP  GR  CAL TEN
000000cc: 5350 4420 4143 5120 4143 2020 5254 2020  SPD ACQ AC  RT
000000dc: 2020 2020 2020 2020 2020 2020 2020 2020
000000ec: 2020 2020 2020 2020 2020 2020 2020 2020
000000fc: 2020 2020 2020 2020 443a 6600 4438 fe00          D:f.D8..
0000010c: 4040 0000 0000 0000 4210 0000 4d4e 3233  @@......B...MN23
0000011c: 394a 2031
```

long_str() → str
Returns a multi-line string describing self.

add_block(*block: bytes*) → None
Adds a data block of frame data to my temporary data structure(s).

complete() → None
Converts the existing frame data to a LogPass.FrameArray. This adds a computed X-axis and removes temporary data structures.

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

TotalDepth.BIT.ReadBIT.**create_bit_frame_array_from_file**(*file: BinaryIO*) → List[TotalDepth.BIT.ReadBIT.BITFrameArray]
Given a file this returns a list of BITFrameArray objects.

TotalDepth.BIT.ReadBIT.**create_bit_frame_array_from_path**(*file_path: str*) → List[TotalDepth.BIT.ReadBIT.BITFrameArray]
Given a file path this returns a list of BITFrameArray objects or raises.

class TotalDepth.BIT.ReadBIT.**FileSizeTime**(*name*, *size*, *time*)

property name
Alias for field number 0

property size
Alias for field number 1

property time
Alias for field number 2

__str__()
Return str(self).

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *name: str*, *size: int*, *time: float*)
Create new instance of FileSizeTime(name, size, time)

__repr__()
Return a nicely formatted representation string

TotalDepth.BIT.ReadBIT.**print_summarise_frame_array**(*frame_array: TotalDepth.BIT.ReadBIT.BITFrameArray*) → None
Summarise all the channels in the Frame Array.

TotalDepth.BIT.ReadBIT.**print_process_file**(*file_path: str*, *verbose: int*, *summary: bool*) → TotalDepth.BIT.ReadBIT.FileSizeTime

Process a BIT file and print out a summary.

TotalDepth.BIT.ReadBIT.**print_process_directory**(*directory: str*, *recursive: bool*, *verbose: int*, *summary: bool*) → Dict[str, TotalDepth.BIT.ReadBIT.FileSizeTime]

Processes a complete directory for BIT files.

TotalDepth.BIT.ReadBIT.**main**() → int

Main entry point.

TotalDepth.BIT.ToLAS

This reads Dresser Atlas BIT files and writes LAS files.

TotalDepth.BIT.ToLAS.**bit_frame_array_to_las_file**(*bit_frame_array: TotalDepth.BIT.ReadBIT.BITFrameArray*, *input_file: str*, *logical_file_number: int*, *frame_slice: Optional[TotalDepth.common.Slice.Slice]*, *channel_name_sub_set: Set[str]*, *field_width: int*, *float_format: str*, *las_file: TextIO*) → None

Writes a single Frame Array to an open LAS file.

TotalDepth.BIT.ToLAS.**single_bit_path_to_las_path**(*bit_path: str*, *_array_reduction: str*, *path_out: str*, *frame_slice: TotalDepth.common.Slice.Slice*, *channels: Set[str]*, *field_width: int*, *float_format: str*) → TotalDepth.LAS.core.WriteLAS.LASWriteResult

Given a path to BIT file this reads the file and writes out a LAS file for each frame array. The LAS file names are numbered sequentially.

TotalDepth.BIT.ToLAS.**main**() → int

Main entry point.

1.9.4 TotalDepth.LAS API Reference

Contents:

TotalDepth.LAS.ReadLASFiles

Reads LAS files and provides statistics on their content.

This also measures the performance of the LAS parser.

Created on Jan 12, 2012

@author: paulross

TotalDepth.LAS.ReadLASFiles.**main**()

Main entry point.

TotalDepth.LAS.core.LASConstants

TotalDepth.LAS.core.LASRead

Reads LAS files.

Created on Jan 11, 2012

Research

Finding occurrences of `STEP`:

```
$ grep -rIh 'STEP\.' LAS_1.2_2.0/ | sed -e 's/^[[:space:]]*//' | tr -s ' ' | sort -b␣
↪| uniq
```

From that file occurrences of `STEP` of zero in around 24,000 files:

> $ grep -rI 'STEP.' LAS_1.2_2.0/ | sed -e 's/^[[:space:]]*//' | tr -s ' ' | grep 'STEP.F 0.00' | wc -l $ grep -rI 'STEP.' LAS_1.2_2.0/ | sed -e 's/^[[:space:]]*//' | tr -s ' ' | grep 'STEP.M 0.00' | wc -l $ grep -rI 'STEP.' LAS_1.2_2.0/ | sed -e 's/^[[:space:]]*//' | tr -s ' ' | grep 'STEP.S 0.00' | wc -l

F: 629 M: 114 S: 29 Total 772 or about 3.2%

exception TotalDepth.LAS.core.LASRead.**ExceptionLASRead**
Specialisation of exception for LASRead.

exception TotalDepth.LAS.core.LASRead.**ExceptionLASReadSection**
Specialisation of exception for LASRead when handling sections.

exception TotalDepth.LAS.core.LASRead.**ExceptionLASReadSectionArray**
Specialisation of exception for LASRead when handling array section.

exception TotalDepth.LAS.core.LASRead.**ExceptionLASReadData**
Specialisation of exception for LASRead when reading data.

exception TotalDepth.LAS.core.LASRead.**ExceptionLASKeyError**
Equivalent to KeyError when looking stuff up.

TotalDepth.LAS.core.LASRead.**LAS_FILE_EXT = '.las'**
LAS file extension

TotalDepth.LAS.core.LASRead.**LAS_FILE_EXT_LOWER = '.las'**
LAS lower case file extension

TotalDepth.LAS.core.LASRead.**has_las_extension**(*fp*)
Returns True if the file extansion is a LAS one.

TotalDepth.LAS.core.LASRead.**RE_COMMENT = re.compile('^\\s*#(.*)$')**
Regex to match a comment Section 5.1 of "LAS Version 2.0: A Digital Standard for Logs, Update February 2017"

TotalDepth.LAS.core.LASRead.**DEBUG_LINE_BY_LINE = False**
logger.debug call here can add about 50% of the processing time

TotalDepth.LAS.core.LASRead.**generate_lines**(*text_file*)
Given an file-like object this generates non-blank, non-comment lines. It's a co-routine so can accept a line to put back.

TotalDepth.LAS.core.LASRead.**SECT_TYPES = 'VWCPOA'**
All section identifiers

TotalDepth.LAS.core.LASRead.**SECT_TYPES_WITH_DATA_LINES = 'VWCP'**
Section with data lines

TotalDepth.LAS.core.LASRead.**SECT_TYPES_WITH_INDEX = 'VWCPA'**
Section with index value in column 0

TotalDepth.LAS.core.LASRead.**RE_SECT_HEAD = re.compile('^~([VWCPOA])(.+)*$')**
Regex to match a section head

TotalDepth.LAS.core.LASRead.**SECT_DESCRIPTION_MAP = {'A': 'ASCII Log Data', 'C': 'Curve Info**
Map of section identifiers to description

TotalDepth.LAS.core.LASRead.**RE_LINE_FIELD_0 = re.compile('^\\s*([^ .:]+)\\s*$')**
The 'MENM' field, no spaces, dots or colons. ONE group Perhaps Allow spaces in mnemonic. For example: "SWU -CPX.V/V 00 000 00 00: 75 Unlimited Formation Water Saturation (Complex Litho Model)" This is specifically excluded by LAS 2.0 (Page 4, 2017-01) but it happens. Also: "Spaces are permitted in front of the mnemonic and between the end of the mnemonic and the dot."

TotalDepth.LAS.core.LASRead.**RE_LINE_FIELD_1 = re.compile('^([^ :]+)*(.+)*$')**
The data field which is the middle of the line between the first '.' and last ':' This is composed of optional units and value. Units must follow the dot immediately and contain no colons or spaces Note: Group 2 may have leading and trailing spaces TWO groups

TotalDepth.LAS.core.LASRead.**string_to_value**(*value: Any*) → Any
Convert a value to an integer, float, bool or string. If a string it is stripped.

class TotalDepth.LAS.core.LASRead.**SectLine**
Captures the four fields from a single line: mnem unit valu desc

property mnem
Alias for field number 0

property unit
Alias for field number 1

property valu
Alias for field number 2

property desc
Alias for field number 3

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *mnem: str*, *unit: str*, *valu: Any*, *desc: str*)
Create new instance of SectLine(mnem, unit, valu, desc)

__repr__()
Return a nicely formatted representation string

class TotalDepth.LAS.core.LASRead.**LASSection**(*section_type: str*, *raise_on_error: bool = True*)
Contains data on a section.

SECTION_MNEMONIC_ORDER_AND_VALUES = {'V': (('VERS', (1.2, 2.0)), ('WRAP', (True, False
Contains the allowable values of certain mnemonics in a section. The appropriate section must have these mnemonics in the given order and the values of those mnemonics must be one of the given values.

__init__(*section_type: str*, *raise_on_error: bool = True*)
Initialize self. See help(type(self)) for accurate signature.

__str__()
Return str(self).

__len__()
Number of members.

__contains__(*mnemonic: str*)
Membership test.

add_member_line(*line_number*, *line*)
Given a line this decomposes it to its _members. line_number is the position of the line l in the file starting at 1. Empty lines are ignored.

create_index()
Creates an index of {key : ordinal, ... }. key is the first object in each member. This will be a MNEM for most sections but a depth as a float for an array section.

finalise()
Finalisation of section, this updates the internal representation.

__getitem__(*key*) → TotalDepth.LAS.core.LASRead.SectLine
Returns an entry, key can be int or str.

mnemonics()
Returns an list of mnemonics.

units()
Returns an list of mnemonic units.

keys()
Returns the keys in the internal map.

find(*m*)
Returns the member ordinal for mnemonic m or -1 if not found. This can be used for finding the array column for a particular curve.

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

class TotalDepth.LAS.core.LASRead.**LASSectionArray**(*section_type: str*, *wrap: bool*, *curve_section: TotalDepth.LAS.core.LASRead.LASSection*, *null: Union[int, float] = -999.25*, *raise_on_error: bool = True*)
Contains data on an array section.

__init__(*section_type: str*, *wrap: bool*, *curve_section: TotalDepth.LAS.core.LASRead.LASSection*, *null: Union[int, float] = -999.25*, *raise_on_error: bool = True*)
Initialize self. See help(type(self)) for accurate signature.

add_member_line(*line_number: int*, *line: str*) → None
Process a line in an array section.

__getitem__(*key*) → TotalDepth.common.LogPass.FrameChannel
Returns an entry, key can be int or str.

__len__()
Number of members.

create_index()
Creates an index of {key : ordinal, ... }. key is the first object in each member. This will be a MNEM for most sections but a depth as a float for an array section.

finalise()
Finalisation.

frame_size()
Returns the number of data points in a frame.

class TotalDepth.LAS.core.LASRead.**LASBase**(*identity: str*)
Base class for LAS reading and writing. This provides common functionality to child classes.

UNITS_LAS_TO_LIS = {'F': 'FEET', 'mts': 'M '}
Mapping of commonly seen LAS units to proper LIS units Both key and value must be strings

__init__(*identity: str*)
Initialize self. See help(type(self)) for accurate signature.

__len__()
Number of sections.

__getitem__(*key*)
Returns a section, key can be int or str.

generate_sections() → Sequence[TotalDepth.LAS.core.LASRead.LASSection]
Yields up each section.

property null_value
The NULL value, defaults to -999.25.

property x_axis_units
The X axis units.

property x_axis_start
The Xaxis start value as an EngVal.

property x_axis_stop
The Xaxis end value as an EngVal.

property x_axis_step
The Xaxis step value as an EngVal.

is_log_down() → bool
Returns True if X axis is increasing i.e. for time or down log.

get_wsd_mnemonic(*mnemonic*) → Tuple[Optional[str], Optional[str]]
Returns a tuple of (value, units) for a Mnemonic that may appear in either a Well section or a Parameter section. Units may be None if empty. Returns (None, None) if nothing found.

get_all_wsd_mnemonics() → Set[str]
Returns a set of mnemonics from the Well section and the Parameter section.

has_output_mnemonic(*mnemonic: str*)
Returns True if theMnem, a Mnem.Mnem() object is an output in the Curve section. It will use the alternate names table LGFORMAT_LAS from LASConstants to interpret curves that are not exact matches.

curve_mnemonics(*ordered=False*)
Returns list of curve names actually declared in the Curve section. List will be unordered if ordered is False.

curve_units_as_str(*m*)
Given a curve as a Mnem.Mnem() this returns the units as a string. May raise KeyError.

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

class TotalDepth.LAS.core.LASRead.**LASRead**(*file_path: Union[str, TextIO], file_identity: str = '', raise_on_error: bool = True*)
Reads a LAS file.

__init__ (*file_path: Union[str, TextIO], file_identity: str = '', raise_on_error: bool = True*)
Reads a LAS file from theFp that is either a string (file path) or a file like object. If raise_on_error is True then some errors will terminate processing, if False then some errors will be ignored.

number_of_frames() → int
Returns the number of frames of data in an 'A' record if I have one.

number_of_data_points()
Returns the number of frame data points in an 'A' record if I have one.

1.9.5 TotalDepth.LIS API Reference

Contents:

TotalDepth.LIS.core.EngVal (Engineering Values)

This module handles engineering values i.e. a real numeric value associated with a unit-of-measure.

Operator Overloading

EngVals can be operated on by a real number or an other EngVal.

Addition and subtraction

The operators `+, -, +=, -=` work on mixed real numbers and EngVals as expected. If two EngVal objects then unit conversion is performed before the operation. If the two EngVals have units in different categories an ExceptionUnit will be raised.

Division

Operators `/, /=` are implemented.

Division of an EngVal by a real number preserves the EngVal units.

Division of an EngVal by an EngVal results in the following:

- It preserves the EngVal units if the denominator units are DIMENSIONLESS.
- **The result will have DIMENSIONLESS EngVal units if the numerator and denominator** have units that are freely convertible i.e. in the same unit category.
- An ExceptionUnit will be raised.

Multiplication

Operators *, *= are implemented for reals and EngVals. If an EngVal the units must be DIMENSIONLESS.

Notes

rval operators are implemented and this can result in type promotion. For example the result of 4.0 * EngVal(16, b'INCH') is an EngVal(64.0, b'INCH').

Created on 24 Nov 2010

EngVal Reference

exception TotalDepth.LIS.core.EngVal.**ExceptionEngVal**

class TotalDepth.LIS.core.EngVal.**EngVal**(*theVal*, *theUom=Mnem(b'x00x00x00x00')*)

Represents an engineering value that consists of a numeric value and a unit of measure (usually a string).

__init__(*theVal*, *theUom=Mnem(b'\x00\x00\x00\x00')*)

Initialize self. See help(type(self)) for accurate signature.

dimensionless()

Returns True if the measure is dimensionless.

__str__()

String representation.

pStr()

Returns a 'pretty' ASCII string.

strFormat(*theFormat*, *incPrefix=True*)

Returns as as string with the value to the specified format (must be capable of handling floating point values.

__add__(*other*)

Overload self+other, returned result has the sum of self and other. other can be an EngVal or a Real number. The units chosen are self's.

__radd__(*other*)

Right value addition, see __add__().

__sub__(*other*)

Overload self-other, returned result has the sum of self and other. The units chosen are self's.

__rsub__(*other*)

Right value subtraction, see __sub__().

__mul__(*other*)

Overload self*other. other must be a real number or a dimensionless EngVal.

__rmul__(*other*)

Right value multiplication, see __mul__().

__truediv__(*other*)

Overload self/other. other must be a real number or an EngVal. If the units of other are dimensionless then treat as a real number. If the units are in the same category as me convert them and return a dimensionless EngVal.

__rtruediv__(*other*)
Right value true division, see __truediv__().

__iadd__(*other*)
Addition in place, other must be a real number or an EngVal where it is converted to my units.

__isub__(*other*)
Subtraction in place, other must be a real number or an EngVal where it is converted to my units.

__imul__(*other*)
Overload self *= *other*. other must be a real number or a dimensionless EngVal.

__itruediv__(*other*)
Overload self /= other. other must be a real number or a dimensionless EngVal.

__lt__(*other*)
True if self < other False otherwise. If other is an EngVal unit conversion is performed which may raise an ExceptionUnit. If other is a real number then self units are ignored.

__le__(*other*)
True if self <= other False otherwise. If other is an EngVal unit conversion is performed which may raise an ExceptionUnit. If other is a real number then self units are ignored.

__eq__(*other*)
True if self == other False otherwise. If other is an EngVal unit conversion is performed which may raise an ExceptionUnit. If other is a real number then self units are ignored.

__ne__(*other*)
True if self != other False otherwise. If other is an EngVal unit conversion is performed which may raise an ExceptionUnit. If other is a real number then self units are ignored.

__gt__(*other*)
True if self > other False otherwise. If other is an EngVal unit conversion is performed which may raise an ExceptionUnit. If other is a real number then self units are ignored.

__ge__(*other*)
True if self >= other False otherwise. If other is an EngVal unit conversion is performed which may raise an ExceptionUnit. If other is a real number then self units are ignored.

convert(*theUnits*)
Convert my value to the supplied units in-place. May raise an ExceptionUnits.

getInUnits(*theUnits*)
Returns my value to the supplied units. May raise an ExceptionUnits.

newEngValInUnits(*theUnits*)
Returns a new EngVal converting me to the supplied units. May raise an ExceptionUnits.

newEngValInOpticalUnits()
Returns a new EngVal converting me to the 'optical' units if possible. For example a value in b'.1IN" will be converted to b'FEET'.

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

class TotalDepth.LIS.core.EngVal.**EngValRc**(*theVal*, *theUom*, *theRc=None*)
An engineering value with a integer Representation Code.

__init__(*theVal*, *theUom*, *theRc=None*)
Initialize self. See help(type(self)) for accurate signature.

__str__()
String representation.

encode()
Encode my value to my RepCode returning a bytes object. May raise an ExceptionEngVal.

TotalDepth.LIS.core.File

Reads or writes LIS data to a physical file.

Created on 14 Nov 2010

exception TotalDepth.LIS.core.File.**ExceptionFile**
Specialisation of exception for Physical files.

exception TotalDepth.LIS.core.File.**ExceptionFileRead**
Specialisation of exception for reading Physical files.

exception TotalDepth.LIS.core.File.**ExceptionFileWrite**
Specialisation of exception for writing Physical files.

class TotalDepth.LIS.core.File.**FileBase**(*theFile*, *theFileId*, *mode*, *keepGoing*)
LIS file handler. This handles Physical Records (and TIF records).

__init__(*theFile*, *theFileId*, *mode*, *keepGoing*)
Initialize self. See help(type(self)) for accurate signature.

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

class TotalDepth.LIS.core.File.**FileRead**(*theFile*, *theFileId=None*, *keepGoing=False*, *pad_modulo: int = 0*, *pad_non_null: bool = False*)
LIS file reader, this offers the caller a number of incremental read operations. This handles Physical Records (and TIF records).

theFile - A file like object or string, if the latter it assumed to be a path.

theFileId - File identifier, this could be a path for example. If None the RawStream will try and cope with it.

keepGoing - If True we do our best to keep going.

__init__(*theFile*, *theFileId=None*, *keepGoing=False*, *pad_modulo: int = 0*, *pad_non_null: bool = False*)
Constructor with: theFile - A file like object or string, if the latter it assumed to be a path. theFileId - File identifier, this could be a path for example. If None the RawStream will try and cope with it. keepGoing - If True we do our best to keep going.

readLrBytes(*theLen=-1*)
Reads theLen LogicalData bytes and returns it or None if nothing left to read for this logical record. If theLen is -1 all the remaining Logical data is read. This positions the file at the _next_ Logical Record or EOF.

skipLrBytes(*theLen=-1*)
Skips logical data and returns a count of skipped bytes. If theLen is -1 all the remaining Logical data is read.

seekCurrentLrStart()
Setting the file position directly to the beginning of a PRH or TIF marker (if present) for the current Logical Record.

skipToNextLr()
Skips the rest of the current Logical Data and positions the file at the start of the next Logical Record.

tellLr()
Returns the absolute file position of the start current Logical record. This value can be safely used in seekLr.

tell()
Returns the absolute position of the file.

ldIndex()
Returns the index position in the current logical data.

seekLr(*offset*)
External setting of file position directly to the beginning of a PRH or TIF marker (if present). The caller is fully responsible for getting this right!

hasLd()
Returns True if there is logical data to be read, False otherwise. NOTE: This will return False on file initialisation and only return True once the Physical Record Header (i.e. one or more logical bytes) has been read.

rewind()
Sets the file position to the beginning of file.

property isEOF
True if at EOF.

unpack(*theStruct*)
Unpack some logical bytes using the supplied format. format ~ a struct.Struct object. Returns a tuple of the number of objects specified by the format or None.

class TotalDepth.LIS.core.File.**FileWrite**(*theFile*, *theFileId=None*, *keepGoing=False*, *hasTif=False*, *thePrLen=65535*, *thePrt=<TotalDepth.LIS.core.PhysRec.PhysRecTail object>*)
LIS file writer. This handles Physical Records (and TIF records).

theFile - A file like object or string, if the latter it assumed to be a path.

theFileId - File identifier, this could be a path for example. If None the RawStream will try and cope with it.

keepGoing - If True we do our best to keep going.

hasTif - Insert TIF markers or not.

thePrLen - Max Physical Record length, defaults to the maximum possible length.

thePrt - Physical Records Trailer settings (defaults to PhysRec.PhysRecTail()).

__init__(*theFile*, *theFileId=None*, *keepGoing=False*, *hasTif=False*, *thePrLen=65535*, *thePrt=<TotalDepth.LIS.core.PhysRec.PhysRecTail object>*)
Constructor with: theFile - A file like object or string, if the latter it assumed to be a path. theFileId - File identifier, this could be a path for example. If None the RawStream will try and cope with it. keepGoing - If True we do our best to keep going. hasTif - Insert TIF markers or not. thePrLen - Max Physical Record length, defaults to the maximum possible length. thePrt - Physical Records Trailer settings (defaults to PhysRec.PhysRecTail()).

write(*theLr*)
Writes the Logical Record to the file. Returns the tell() of the start of the LR.

close()
Closes the file.

class TotalDepth.LIS.core.File.**PhysicalRecordSettings**
Container for the settings to read Physical Records.

property pad_modulo
Alias for field number 0

property pad_non_null
Alias for field number 1

__str__() → str
Return str(self).

__repr__() → str
Return str(self).

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *pad_modulo: int*, *pad_non_null: bool*)
Create new instance of PhysicalRecordSettings(pad_modulo, pad_non_null)

TotalDepth.LIS.core.File.**best_physical_record_pad_settings**(*file_path_or_object: Union[str, _io.BytesIO]*, *pr_limit=0*) → Union[None, TotalDepth.LIS.core.File.PhysicalRecordSettings]
This attempts to find the best settings to read Physical Records. It returns a PhysicalRecordSettings on success or None on failure. pr_limit limits the number of Physical Records to test if you want higher performance otherwise all Physical Records are read.

Typically 38Mb file with 46276 Physical Records processed in 0.380 (s) so 10ms/Mb or 100Mb/s. This is proportionate so if limited to 100 records this would be around 0.001 (s)

TotalDepth.LIS.core.File.**file_read_with_best_physical_record_pad_settings**(*file_path_or_object: Union[str, _io.BytesIO]*, *file_id=None*, *pr_limit=0*) → Union[None, TotalDepth.LIS.core.File.F
Explores a LIS file with the 'best' Physical Record settings and returns a FileRead object or None.

TotalDepth.LIS.core.File.**scan_file_no_output**(*file_path_or_object: Union[str, _io.BytesIO]*, *keep_going: bool*, *pad_modulo: int*, *pad_non_null: bool*, *pr_limit=0*)
Reads a file as if it was a LIS file and returns the number of Physical Records.

Typically 38Mb file processed in 0.381 seconds so 10ms/Mb or 100Mb/s.

TotalDepth.LIS.core.File.**scan_file_with_different_padding**(*file_path_or_object: Union[str, _io.BytesIO]*, *keep_going: bool*, *pr_limit=0*) → Dict[TotalDepth.LIS.core.File.PhysicalRecordSe int]
Tries all different padding options and returns a dict with the number of Physical Records parsed.

TotalDepth.LIS.core.File.**ret_padding_options_with_max_records**(*pad_opts_to_prs: Dict[TotalDepth.LIS.core.File.PhysicalRec int]*) → List[TotalDepth.LIS.core.File.PhysicalRec

Returns the list of padding options that maximises the number of Physical Records parsed from the structure provided by scan_file_with_different_padding().

TotalDepth.LIS.core.FileIndexer

#Contents: Indexes LIS files.

Created on 10 Feb 2011

@author: p2ross

exception TotalDepth.LIS.core.FileIndexer.**ExceptionFileIndex**

Specialisation of exception for the LIS file indexer.

class TotalDepth.LIS.core.FileIndexer.**IndexObjBase**(*tell*, *lrType*, *theF*)

Base class for indexed objects.

tell - The file position of the Logical Record as an integer.

lrType - The Logical Record type as an integer.

theF - The LIS File object. The file ID is recorded for later error checking.

__init__(*tell*, *lrType*, *theF*)

Initialize self. See help(type(self)) for accurate signature.

__str__()

Return str(self).

tocStr()

Returns a 'pretty' string suitable for a table of contents.

property tell

The file offset of the Logical Record.

property lrType

The Logical Record type, in integer.

property logicalRecord

The underlying LogiRec object or None.

property isDelimiter

True if this represents a delimiter record e.g. File Head/Tail.

canAdd(*iflrType*)

Returns True if this can accumulate another IFLR.

iflrType()

Returns the IFLR type that this EFLR can describe.

setLogicalRecord(*theFile*)

Sets the logicalRecord property to an LogiRec object from theFile.

jsonObject()

Return an Python object that can be JSON encoded.

__weakref__

list of weak references to the object (if defined)

class TotalDepth.LIS.core.FileIndexer.**IndexNone**(*tell*, *lrType*, *theF*)
NULL class just takes the LR information and skips to next LR.

__init__(*tell*, *lrType*, *theF*)
Initialize self. See help(type(self)) for accurate signature.

setLogicalRecord(*theFile*)
Sets the logicalRecord property to an LogiRec object from theFile.

class TotalDepth.LIS.core.FileIndexer.**IndexUnknownInternalFormat**(*tell*, *lrType*, *theF*)
Binary verbatim class for things like encrypted records, images, raw table dumps and so on.

__init__(*tell*, *lrType*, *theF*)
Initialize self. See help(type(self)) for accurate signature.

setLogicalRecord(*theFile*)
Sets the logicalRecord property to an LogiRec.LrTable() object.

class TotalDepth.LIS.core.FileIndexer.**IndexLrFull**(*tell*, *lrType*, *theF*, *theClass*)
Takes a full LR and assigns it to self._lr.

theClass is a cls that is use to instantiate a Logical Record object at the current file position.

__init__(*tell*, *lrType*, *theF*, *theClass*)
Initialize self. See help(type(self)) for accurate signature.

setLogicalRecord(*theFile*)
Sets the logicalRecord property to an LogiRec object from theFile.

class TotalDepth.LIS.core.FileIndexer.**IndexFileHeadTail**(*tell*, *lrType*, *theF*, *theClass*)
Indexes a File header or trailer. The full Logical record is retained.

__init__(*tell*, *lrType*, *theF*, *theClass*)
Initialize self. See help(type(self)) for accurate signature.

tocStr()
Returns a 'pretty' string suitable for a table of contents.

class TotalDepth.LIS.core.FileIndexer.**IndexFileHead**(*tell*, *lrType*, *theF*)
Indexes a File header. The full Logical record is retained.

__init__(*tell*, *lrType*, *theF*)
Initialize self. See help(type(self)) for accurate signature.

class TotalDepth.LIS.core.FileIndexer.**IndexFileTail**(*tell*, *lrType*, *theF*)
Indexes a File trailer. The full Logical record is retained.

__init__(*tell*, *lrType*, *theF*)
Initialize self. See help(type(self)) for accurate signature.

class TotalDepth.LIS.core.FileIndexer.**IndexTapeHeadTail**(*tell*, *lrType*, *theF*, *theClass*)
Indexes a Tape header or trailer. The full Logical record is retained.

__init__(*tell*, *lrType*, *theF*, *theClass*)
Initialize self. See help(type(self)) for accurate signature.

tocStr()
Returns a 'pretty' string suitable for a table of contents.

class TotalDepth.LIS.core.FileIndexer.**IndexTapeHead**(*tell*, *lrType*, *theF*)
Indexes a Tape header. The full Logical record is retained.

__init__(*tell*, *lrType*, *theF*)
Initialize self. See help(type(self)) for accurate signature.

class TotalDepth.LIS.core.FileIndexer.**IndexTapeTail**(*tell*, *lrType*, *theF*)
Indexes a Tape trailer. The full Logical record is retained.

__init__(*tell*, *lrType*, *theF*)
Initialize self. See help(type(self)) for accurate signature.

class TotalDepth.LIS.core.FileIndexer.**IndexReelHeadTail**(*tell*, *lrType*, *theF*, *theClass*)
Indexes a Reel header or trailer. The full Logical record is retained.

__init__(*tell*, *lrType*, *theF*, *theClass*)
Initialize self. See help(type(self)) for accurate signature.

tocStr()
Returns a 'pretty' string suitable for a table of contents.

class TotalDepth.LIS.core.FileIndexer.**IndexReelHead**(*tell*, *lrType*, *theF*)
Indexes a Reel header. The full Logical record is retained.

__init__(*tell*, *lrType*, *theF*)
Initialize self. See help(type(self)) for accurate signature.

class TotalDepth.LIS.core.FileIndexer.**IndexReelTail**(*tell*, *lrType*, *theF*)
Indexes a Reel trailer. The full Logical record is retained.

__init__(*tell*, *lrType*, *theF*)
Initialize self. See help(type(self)) for accurate signature.

class TotalDepth.LIS.core.FileIndexer.**IndexTable**(*tell*, *lrType*, *theF*)
Table type logical records. Here we capture the first component block so that we know the name of the table.

__init__(*tell*, *lrType*, *theF*)
Initialize self. See help(type(self)) for accurate signature.

property name
The name of the table from the first component block.

__str__()
Return str(self).

tocStr()
Returns a 'pretty' string suitable for a table of contents.

setLogicalRecord(*theFile*)
Sets the logicalRecord property to an LogiRec.LrTable() object.

jsonObject()
Return an Python object that can be JSON encoded.

class TotalDepth.LIS.core.FileIndexer.**IndexLogPass**(*tell*, *lrType*, *theF*, *xAxisIndex=0*)
The index of a Log Pass. This contains a LogPass object.

xAxisIndex is the channel index that is regarded as the X axis. This is the indirect axis if present or defaults to channel 0.

__init__(*tell*, *lrType*, *theF*, *xAxisIndex=0*)
Initialize self. See help(type(self)) for accurate signature.

property logPass
The LogPass object.

__str__()
Return str(self).

tocStr()
Returns a 'pretty' string suitable for a table of contents.

canAdd(*iflrType*)
Returns the IFLR type that this EFLR can describe.

iflrType()
Returns the IFLR type that this EFLR can describe.

add(*tell*, *lrType*, *theF*)
Add an IFLR.

jsonObject()
Return an Python object that can be JSON encoded.

class TotalDepth.LIS.core.FileIndexer.**PlotRecordSet**
A POD class that can contain a set of references to the essential (plus optional) logical records for plotting.

__init__()
Initialize self. See help(type(self)) for accurate signature.

__str__()
Return str(self).

canPlotFromInternalRecords()
True if I have a valid value that could be yielded, i.e. the minimum information from the file that allows a plot. In practice this means a FILM, PRES record and a LogPass.

canPlotFromExternalRecords()
True if I have a valid value that could be yielded, i.e. the minimum information from the file that allows a plot using some external definition of what has to be plotted. In practice this means a LogPass.

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

class TotalDepth.LIS.core.FileIndexer.**FileIndex**(*theF*, *xAxisIndex=0*)
Create an index for the LIS file, theF is a LIS File object.

xAxisIndex is the channel index that is regarded as the X axis (default 0). This is currently ignored in the absence of a reasonable use case.

__init__(*theF*, *xAxisIndex=0*)
Initialize self. See help(type(self)) for accurate signature.

longDesc() → str
Returns a string that is the long description of this object.

property lrTypeS
Returns a list of Logical Record types (integers) that is in the index.

numLogPasses() → int
Returns the number of IndexLogPass objects in the index.

genLogPasses() → Sequence[TotalDepth.LIS.core.FileIndexer.IndexLogPass]
Yields all IndexLogPass objects in the index.

genPlotRecords(*fromInternalRecords=True*) → Sequence[TotalDepth.LIS.core.FileIndexer.PlotRecordSet]
This provides the minimal information for creating a Plot. It yields a PlotRecordSet that has (tell_FILM, tell_PRES, tell_AREA, tell_PIP, IndexLogPass) objects that are not separated by delimiter records. tell_FILM is the file offset of the FILM record. tell_PRES is the file offset of the PRES record. tell_AREA

is None or the file offset of the AREA record. tell_PIP is None or the file offset of the PIP record. IndexLogPass is a IndexLogPass object.

genAll() → Sequence[TotalDepth.LIS.core.FileIndexer.IndexObjBase]
Generates each index object (child class of IndexObjBase) for example a IndexLogPass.

jsonObject() → Dict[str, Any]
Return an Python object that can be JSON encoded.

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

Usage

Examples

Using a LIS Indexer

Source:

```
myFilePath = "Spam/Eggs.LIS"
# Open a LIS file
myFi = File.FileRead(myFilePath, theFileId=myFilePath, keepGoing=True)
# Index the LIS file
myIdx = FileIndexer.FileIndex(myFi)
# Ask the index for all the LogPass objects, these do not have the frameSet populated␣
↪yet
for aLr in myIdx.genAll():
    print(aLr)
```

Typical result:

```
tell: 0x00000000 type=128 <TotalDepth.LIS.core.FileIndexer.IndexFileHead object at␣
↪0x10197ff90>
tell: 0x00000050 type= 34 name=b'TOOL' <TotalDepth.LIS.core.FileIndexer.IndexTable␣
↪object at 0x10197ffd0>
tell: 0x000001ea type= 34 name=b'INPU' <TotalDepth.LIS.core.FileIndexer.IndexTable␣
↪object at 0x10197fd10>
tell: 0x00002342 type= 34 name=b'OUTP' <TotalDepth.LIS.core.FileIndexer.IndexTable␣
↪object at 0x101b0b1d0>
tell: 0x00003622 type= 34 name=b'CONS' <TotalDepth.LIS.core.FileIndexer.IndexTable␣
↪object at 0x101b0bed0>
tell: 0x0000456a type= 34 name=b'CONS' <TotalDepth.LIS.core.FileIndexer.IndexTable␣
↪object at 0x101b0b590>
tell: 0x000064aa type= 34 name=b'PRES' <TotalDepth.LIS.core.FileIndexer.IndexTable␣
↪object at 0x101b0b050>
tell: 0x00006efe type= 34 name=b'FILM' <TotalDepth.LIS.core.FileIndexer.IndexTable␣
↪object at 0x101b0bfd0>
tell: 0x00006fc6 type= 34 name=b'AREA' <TotalDepth.LIS.core.FileIndexer.IndexTable␣
↪object at 0x101b0bb50>
<TotalDepth.LIS.core.LogPass.LogPass object at 0x101b06490>
<TotalDepth.LIS.core.LogPass.LogPass object at 0x101b12410>
tell: 0x0017cbee type=129 <TotalDepth.LIS.core.FileIndexer.IndexFileTail object at␣
↪0x101b173d0>
```

Testing

The unit tests are in test/TestFileIndexer.Py

TotalDepth.LIS.core.FrameSet

This describes the LIS FrameSet that contains the binary frame data. It is effectively a wrapper around a 2-D numpy array with specific APIs to interface that with a channel specific shape. Importantly FrameSets can be partial in that they need not hold all the data for every frame and every channel. Instead they can hold data for the frames and channels specified by the caller.

The FrameSet module is located in `src/TotalDepth/LIS/core` and can be imported thus:

```
from TotalDepth.LIS.core import FrameSet
```

FrameSet Internals

The internal representation of the FrameSet uses only Python and numpy structures and types, it is thus ignorant of the LIS file format or data structures apart from that described below.

LIS Dependancies

In the current version of the code there are a number of dependencies on knowledge of the LIS file format (or at least TotalDepth's representation of that format):

- `ChArTe` objects are constructed from LIS Datum Specification Block objects.
- `XAxisDecl` consists of a straight copy of various Entry Block values.
- `XAxisDecl` relies on the LIS value of the up/down flag.
- `FrameSet` is constructed with a DFSR and extracts the absent value form the DFSR as well as constructing ChArTe and XAxisDecl objects.
- **`FrameSet` has quite a lot of dependencies of Representation codes these are mainly used in two ways.**
  - Interpreting Dipmeter sub-channels.
  - `setFrameBytes()` uses the RepCode module directly to convert bytes to values for a channel.

These dependencies restrict the use of FrameSet to processing LIS data, if they were removed then FrameSet could be used for other file formats but there is no obvious use case for that as, apart from LIS, TotalDepth supports LAS (trivially simple frame sets) and will support RP66 at some point. The latter may trigger a refactoring of the FrameSet module.

FrameSet API Reference

The FrameSet module provides a means of representing LIS frame data.

Created on 10 Jan 2011

exception TotalDepth.LIS.core.FrameSet.**ExceptionFrameSet**
Specialisation of exception for FrameSet.

exception TotalDepth.LIS.core.FrameSet.**ExceptionFrameSetEmpty**
Raised when an illegal operation is performed on a FrameSet.

exception TotalDepth.LIS.core.FrameSet.**ExceptionFrameSetNULLSpacing**
Raised when FrameSet depends on a frame spacing that can not be determined.

exception TotalDepth.LIS.core.FrameSet.**ExceptionFrameSetMixedChannels**
Raised when generating values for multiple channels where the channels are not of the same shape i.e. number of samples, butsts.

TotalDepth.LIS.core.FrameSet.**chkIdx**(*i*, *l*, *msg*)
Global function for checking indexes and raising an IndexError. msg is expected to have two format fields that take i and l respectively.

TotalDepth.LIS.core.FrameSet.**sliceDefaults**(*theSl*)
Returns a new slice with start=None as 0 and step=None as 1.

class TotalDepth.LIS.core.FrameSet.**DataSeqBase**
Base class for a sequence of objects.

__init__()
Initialize self. See help(type(self)) for accurate signature.

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

class TotalDepth.LIS.core.FrameSet.**SuChArTe**
Sub-channel Array Template.

property numValues
The total number of values in this sub-channel.

index(*theS*, *theB*)
Returns the index of a particular sample or channel with bounds checking.

WARNING: This not correct for Dipmeter sub-channels.

__str__()
String representation.

class TotalDepth.LIS.core.FrameSet.**ChArTe**(*theDsb*)
Channel Array Template. Constructed with a DatumSpecBlock object.

Implementation note: _index() and _dipmeterIndex() have no bounds checking but are significantly faster for those routines that access then with pre-checked limits.

__init__(*theDsb*)
Constructor with a DatumSpecBlock object.

__str__()
String representation.

property numSubChannels
Number of sub-channels.

property lisSize
Number of bytes per frame in the LIS representation.

subChOffsRange(*sc*)
Returns a range object that is the sub-channel offset in the frame relative to the start of the channel.

subChOffsSlice(*sc*, *chOfs=0*)
Returns a slice object that is the sub-channel offset in the frame relative to the start of the channel.

index(*theSc*, *theSa*, *theBu*)
Returns the index of a particular sub-channel, sample and burst. Order is (fastest changing first): burst, sample. For those with sub-channels: sub-channel, sample This has bounds checking.

dipmeterIndex(*theSc*, *theSa*, *theBu*)
Returns dipmeter data index. Bounds checking is performed.

class TotalDepth.LIS.core.FrameSet.**XAxisDecl**

property isLogUp
True if "up" log (x decreasing).

property isLogDown
True if "down" log (x decreasing).

property isIndirectX
True if has indirect X axis.

class TotalDepth.LIS.core.FrameSet.**FrameSet**(*theDfsr*, *theFrameSlice*, *theChS=None*, *xAxisIndex=0*)
Contains the representation of a list of Frames and thus a representation of and 'matrix' (non literal) of: (frame, channel, sub-channel, sample, burst). Commonly shortened to: (fr, ch, sc, sa, bu) Effectively this is a wrapper around a numpy 2-D array that we treat specially to get our 5-D array (LIS channels are not homogeneous).

Constructed with a DFSR, a slice of frame indexes and an optional list of external channel indexes (defaults to all channels). Duplicates in the theChS will be removed and it will be sorted.

xAxisIndex is the external channel index of the X axis (ignored if indirect X).

NUMPY_DATA_TYPE = 'float64'
Data type used in the underlying numpy array.

__init__(*theDfsr*, *theFrameSlice*, *theChS=None*, *xAxisIndex=0*)
Constructed with a DFSR, a slice of frame indexes and an optional list of external channel indexes (defaults to all channels). Duplicates in the theChS will be removed and it will be sorted. xAxisIndex is the external channel index of the X axis (ignored if indirect X).

__str__()
String representation.

longStr()
Returns a long string that describes me.

dumpFrames(*theS=<colorama.ansitowin32.StreamWrapper object>*)
Dump the frames to the stream.

property nbytes
Returns the number of bytes in the underlying array implementation.

property lisSize
The number of LIS bytes that make up this FrameSet.

property numFrames
The number of frames currently in this FrameSet.

property valuesPerFrame
The number of values in each frame currently in this FrameSet.

property numValues
The total number of values in the array.

property frames
Gives access to the raw numpy array.

frame(*fr*)
Returns a specific frame.

xAxisValue(*fr*)
Returns the X axis value for the frame when indirect X is used or None.

xAxisStep(*numFr*)
The distance stepped by numFr.

intFrameNum(*theExtFrameNum*)
Given an external frame number this returns the internal frame number. Will raise an IndexError if the internal frame number is out of range or the external frame number is not in the caller specified slice object.

extFrameNum(*theIntFrameNum*)
Given an internal frame number this returns the external frame number. This does _not_ test that the internal frame number exists.

property numChannels
The number of _internal_ channels.

numSubChannels(*theChExt*)
Number of sub-channels for an external channel.

numSamples(*theChExt*, *theSc*)
Number of samples for the external channel, sub-channel.

numBursts(*theChExt*, *theSc*)
Number of bursts for the external channel, sub-channel.

property isIndirectX
True if there is an indirect X axis, False otherwise.

property xAxisDecl
The XAxisDecl object created from the DFSR.

setFrameBytes(*by*, *fr*, *chFrom*, *chTo*)
Given bytes, convert from Rep Codes to 'float64' and populates the appropriate frame and channel(s). This has random write access so the caller needs to be sensitive to the frame and channel location. fr is the internal frame position to write the data to. chFrom, chTo are inclusive and represent external channel indices.

WARNING: It is rare to use this API directly, instead a FrameSet object is usually a member of a LogPass object and that uses this API as a LogPass is capable of finding and reading bytes objects in a file (a LogPass uses RLE and Type01Plan objects to do this efficiently).

setIndirectX(*fr*, *val*)
Sets an indirect X axis value directly, for example with an EXTRAPOLATE event.

internalChIdx(*chIdxExt*)
Return the internal channel index from the external one.

valueIdxStartExtCh(*ch*)
The index in the frame of the first value for external channel.

valueIdxInFrame(*ch*, *sc*, *sa*, *bu*)
The index in the frame of the value for (external channel, sub-channel, sample, burst). TODO: Provide an API that returns a numpy view of a ch/sc on the frameset.

value(*fr*, *ch*, *sc*, *sa*, *bu*)
Returns a single value from an: internal fr, external ch, sc, sa, bu. This is very good at random access but can be quite slow for iteration compared to the generators.

frameView(*chIdxExt*, *sc*)
Returns a numpy array that is a view of the current frame set for an external channel and sub channel.

frame_channel_sub_channel_values(*frame_index*, *channel_index*, *sub_channel_index*)
Returns a numpy array that is a view of the values for the frame, external channel and sub channel.

genExtChIndexes()
Generates an ordered list of external channel indexes for this (possibly) partial frameset.

genChScValues(*ch*, *sc=0*, *chIsExternal=True*)
Generates values for the external channel and sub channel. sc is ignored unless the channel has > 1 sub-channels. If chIsExternal is True then ch is the external channel index otherwise it is the internal index. The value order is sample/burst.

genChScPoints(*ch*, *sc=0*, *chIsExt=True*)
Generates (xAxis, values) as numbers for the external channel and sub-channel. sc is ignored unless the channel has > 1 sub-channels. If chIsExt is True then ch is the external channel index otherwise it is the internal index.

genMultipleChScPoints(*theChScS*, *chIsExt=True*)
Generates (xAxis, (values, ...)) as numbers for the list of (channel, sub-channel) indexes. An ExceptionFrameSetMixedChannels will be raised if all ch/sc channels are not of the same form i.e. number of samples and bursts. If chIsExt is True then ch is the external channel index otherwise it is the internal index.

genAll()
Yields 6 item tuples (fr ext, ch ext, sc, sa, bu value).

accumulate(*theAccs*)
Calls .add() on every accumulator (with a unary function) for every internal channel and returns an numpy array of (numSubCh, len(theAccs)) doubles. Each accumulator is expected to have __init__(), add() and value() that returns a double implemented. Will raise a ExceptionFrameSetEmpty if there are no values to analyse. Will return None if theAccs is zero length.

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

class TotalDepth.LIS.core.FrameSet.**AccMin**
Accumulates the minimum value.

__init__()
Initialize self. See help(type(self)) for accurate signature.

add(*v*)
Add a new value.

value()
Return the result.

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

class TotalDepth.LIS.core.FrameSet.**AccMax**
Accumulates the maximum value.

__init__()
Initialize self. See help(type(self)) for accurate signature.

add(*v*)
Add a new value.

value()
Return the result.

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

class TotalDepth.LIS.core.FrameSet.**AccMean**
Accumulates the mean value.

__init__()
Initialize self. See help(type(self)) for accurate signature.

add(*v*)
Add a new value.

value()
Return the result.

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

class TotalDepth.LIS.core.FrameSet.**AccStDev**
Accumulates the standard deviation.

__init__()
Initialize self. See help(type(self)) for accurate signature.

add(*v*)
Add a new value.

value()
Return the result.

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

class TotalDepth.LIS.core.FrameSet.**AccCount**
Accumulates the number of values.

__init__()
Initialize self. See help(type(self)) for accurate signature.

add(*v*)
Add a new value.

value()
Return the result.

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

class TotalDepth.LIS.core.FrameSet.**AccDelta**
Base class for acumulating a count of first order differences.

__init__()
Initialize self. See help(type(self)) for accurate signature.

add(*v*)
Add a new value.

value()
Return the result.

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

class TotalDepth.LIS.core.FrameSet.**AccInc**
Counting how many values are an increase from the previous value.

add(*v*)
Add a new value.

class TotalDepth.LIS.core.FrameSet.**AccEq**
Counting how many values are equal to the previous value.

add(*v*)
Add a new value.

class TotalDepth.LIS.core.FrameSet.**AccDec**
Counting how many values are less than the previous value.

add(*v*)
Add a new value.

class TotalDepth.LIS.core.FrameSet.**AccBias**
Measures increment, equal, decrement and computes bias which is: (inc - dec) / total.

__init__()
Initialize self. See help(type(self)) for accurate signature.

add(*v*)
Add a new value.

value()
Return the result.

class TotalDepth.LIS.core.FrameSet.**AccDrift**
Measures drift i.e. the movement between the first and the last value.

__init__()
Initialize self. See help(type(self)) for accurate signature.

add(*v*)
Add a new value.

value()
Return the result.

class TotalDepth.LIS.core.FrameSet.**AccActivity**
Measures curve activity.

__init__()
Initialize self. See help(type(self)) for accurate signature.

add(*v*)
Add a new value.

value()
Return the result.

FrameSet Usage

TODO:

TotalDepth.LIS.core.LogiRec (Logical Records)

Handles LIS Logical Records.

exception TotalDepth.LIS.core.LogiRec.**ExceptionLr**
Specialisation of exception for Logical Records.

exception TotalDepth.LIS.core.LogiRec.**ExceptionLrNotImplemented**
Logical Records that have no implementation here.

exception TotalDepth.LIS.core.LogiRec.**ExceptionCbWrite**
Raised when creating a component block with Python native types that has illogical or conflicting data.

TotalDepth.LIS.core.LogiRec.**LR_TYPE_NORMAL_DATA = 0**
0x00 Normal data record containing log data

TotalDepth.LIS.core.LogiRec.**LR_TYPE_ALTERNATE_DATA = 1**
0x01 Alternate data.

TotalDepth.LIS.core.LogiRec.**LR_TYPE_JOB_ID = 32**
0x20 Job identification

TotalDepth.LIS.core.LogiRec.**LR_TYPE_WELL_DATA = 34**
0x22 Well site data

TotalDepth.LIS.core.LogiRec.**LR_TYPE_TOOL_INFO = 39**
0x27 Tool string info

TotalDepth.LIS.core.LogiRec.**LR_TYPE_ENCRYPTED_TABLE = 42**
0x2a Encrypted table dump

TotalDepth.LIS.core.LogiRec.**LR_TYPE_TABLE_DUMP = 47**
0x2f Table dump

TotalDepth.LIS.core.LogiRec.**LR_TYPE_DATA_FORMAT = 64**
0x40 Data format specification record

TotalDepth.LIS.core.LogiRec.**LR_TYPE_DATA_DESCRIPTOR = 65**
0x41 Data descriptor (not defined in the LIS79 Description Reference Manual)

TotalDepth.LIS.core.LogiRec.**LR_TYPE_TU10_BOOT = 95**
0x5f TU10 software boot

TotalDepth.LIS.core.LogiRec.**LR_TYPE_BOOTSTRAP_LOADER = 96**
0x60 Bootstrap loader

TotalDepth.LIS.core.LogiRec.**LR_TYPE_CP_KERNEL = 97**
0x61 CP-kernel loader boot

TotalDepth.LIS.core.LogiRec.**LR_TYPE_PROGRAM_FILE_HEAD = 100**
0x64 Program file header

TotalDepth.LIS.core.LogiRec.**LR_TYPE_PROGRAM_OVER_HEAD = 101**
0x65 Program overlay header

TotalDepth.LIS.core.LogiRec.**LR_TYPE_PROGRAM_OVER_LOAD = 102**
0x66 Program overlay load

TotalDepth.LIS.core.LogiRec.**LR_TYPE_FILE_HEAD = 128**
0x80 File header

TotalDepth.LIS.core.LogiRec.**LR_TYPE_FILE_TAIL = 129**
0x81 File trailer

TotalDepth.LIS.core.LogiRec.**LR_TYPE_TAPE_HEAD = 130**
0x82 Tape header

TotalDepth.LIS.core.LogiRec.**LR_TYPE_TAPE_TAIL = 131**
0x83 Tape trailer

TotalDepth.LIS.core.LogiRec.**LR_TYPE_REEL_HEAD = 132**
0x84 Reel header

TotalDepth.LIS.core.LogiRec.**LR_TYPE_REEL_TAIL = 133**
0x85 Reel trailer

TotalDepth.LIS.core.LogiRec.**LR_TYPE_EOF = 137**
0x89 Logical EOF (end of file)

TotalDepth.LIS.core.LogiRec.**LR_TYPE_BOT = 138**
0x8a Logical BOT (beginning of tape)

TotalDepth.LIS.core.LogiRec.**LR_TYPE_EOT = 139**
0x8b Logical EOT (end of tape)

TotalDepth.LIS.core.LogiRec.**LR_TYPE_EOM = 141**
0x8d Logical EOM (end of medium)

TotalDepth.LIS.core.LogiRec.**LR_TYPE_OPERATOR_INPUT = 224**
0xe0 Operator command inputs

TotalDepth.LIS.core.LogiRec.**LR_TYPE_OPERATOR_RESPONSE = 225**
0xe1 Operator response inputs

TotalDepth.LIS.core.LogiRec.**LR_TYPE_SYSTEM_OUTPUT = 227**
0xe3 System outputs to operator

TotalDepth.LIS.core.LogiRec.**LR_TYPE_FLIC_COMMENT = 232**
0xe8 FLIC comment

TotalDepth.LIS.core.LogiRec.**LR_TYPE_BLANK_RECORD = 234**
0xea Blank record/CSU comment

TotalDepth.LIS.core.LogiRec.**LR_TYPE_PICTURE = 85**
0x55 Picture

TotalDepth.LIS.core.LogiRec.**LR_TYPE_IMAGE = 86**
0x56 Image

TotalDepth.LIS.core.LogiRec.**LR_TYPE_ALL = (0, 1, 32, 34, 39, 42, 47, 64, 65, 95, 96, 97, 1**
All possible Logical Records Types

TotalDepth.LIS.core.LogiRec.**LR_TYPE_LOG_DATA = (0, 1)**
Logical Records Types for Log data

TotalDepth.LIS.core.LogiRec.**LR_TYPE_TABLE_DATA = (32, 34, 39)**
Logical Records Types for Table data

TotalDepth.LIS.core.LogiRec.**LR_TYPE_DELIMITER_START = (132, 130, 128)**
Logical Records Types for delimiter start records.

TotalDepth.LIS.core.LogiRec.**LR_TYPE_DELIMITER_END = (133, 131, 129)**
Logical Records Types for delimiter end records.

TotalDepth.LIS.core.LogiRec.**LR_TYPE_DELIMITER = (132, 130, 128, 133, 131, 129)**
Logical Records Types for all delimiter records. Delimeter records 'bookend' dynamic and static data that makes up a LIS file.

TotalDepth.LIS.core.LogiRec.**LR_TYPE_MARKER = (137, 138, 139, 141)**
Logical Records Types for all marker records. Marker records terminate original physical media such as a tape.

TotalDepth.LIS.core.LogiRec.**LR_TYPE_DELIMITER_MARKER = (132, 130, 128, 133, 131, 129, 137,**
Logical Records Types for all delimiter and marker records.

TotalDepth.LIS.core.LogiRec.**LR_FIXED_FORMAT = {95: 594, 101: 52, 128: 58, 129: 58, 130**
Table of fixed format Logical Record types and their fixed length in bytes.

TotalDepth.LIS.core.LogiRec.**isDelimiter**(*theType*)
Returns True if the Logical Record Type is a Delimiter record.

TotalDepth.LIS.core.LogiRec.**LR_TYPE_UNKNOWN_INTERNAL_FORMAT = (42, 47, 95, 96, 97, 100, 10**
Logical Records Types with no known format so just treat these as unformatted binary data

TotalDepth.LIS.core.LogiRec.**LR_DESCRIPTION_MAP = {0: 'Normal data record containing log da**
Map of {Logical Records Type : description, ... }

TotalDepth.LIS.core.LogiRec.**LR_DESCRIPTION_UNKNOWN = 'Unknown Logical Record type.'**
Description string for unknown Logical Records Type

TotalDepth.LIS.core.LogiRec.**STRUCT_LR_HEAD = <Struct object>**
Logical Record header (type and attributes)

TotalDepth.LIS.core.LogiRec.**STRUCT_LR_FILE_HEAD_TAIL = <Struct object>**
Logical Record field interpretation via the struct module

TotalDepth.LIS.core.LogiRec.**STRUCT_LR_REEL_TAPE_HEAD_TAIL = <Struct object>**
Logical Record reel/tape head/tail via the struct module

TotalDepth.LIS.core.LogiRec.**STRUCT_COMPONENT_BLOCK_PREAMBLE = <Struct object>**
Component Block preamble as a struct.Struct()

TotalDepth.LIS.core.LogiRec.**STRUCT_ENTRY_BLOCK_PREAMBLE = <Struct object>**
Entry Block preamble as a struct.Struct()

TotalDepth.LIS.core.LogiRec.**STRUCT_DSB = <Struct object>**
Datum Specification Block structure. NOTE: Due to the funny way API codes are done we read then as a single 32 bit unsigned integer and then do a decimal masking operation to extract the four sub-fields

class TotalDepth.LIS.core.LogiRec.**LrBase**(*theType*, *theAttr*)
Base class for Logical Records. Constructed with and integer type and integer attributes.

__init__(*theType*, *theAttr*)
Base class constructor, theType and theAttr are bytes.

init(*theLen*)
Returns a string of spaces of the supplied length.

__str__()
String representation.

property desc
Description ot the LR type.

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

class TotalDepth.LIS.core.LogiRec.**LrMarker**(*theType*, *theAttr*)
A marker record such as EOF BOT EOT EOM.

__init__(*theType*, *theAttr*)
Base class constructor, theType and theAttr are bytes.

class TotalDepth.LIS.core.LogiRec.**LrEOF**(*attr=0*)
A EOF marker record.

__init__(*attr=0*)
Base class constructor, theType and theAttr are bytes.

class TotalDepth.LIS.core.LogiRec.**LrEOFRead**(*theFile*)
A EOF marker record read from a file.

__init__(*theFile*)
Base class constructor, theType and theAttr are bytes.

class TotalDepth.LIS.core.LogiRec.**LrBOT**(*attr=0*)
A BOT marker record.

__init__(*attr=0*)
Base class constructor, theType and theAttr are bytes.

class TotalDepth.LIS.core.LogiRec.**LrBOTRead**(*theFile*)
A BOT marker record read from a file.

__init__(*theFile*)
Base class constructor, theType and theAttr are bytes.

class TotalDepth.LIS.core.LogiRec.**LrEOT**(*attr=0*)
A EOT marker record.

__init__(*attr=0*)
Base class constructor, theType and theAttr are bytes.

class TotalDepth.LIS.core.LogiRec.**LrEOTRead**(*theFile*)
A EOT marker record read from a file.

__init__(*theFile*)
Base class constructor, theType and theAttr are bytes.

class TotalDepth.LIS.core.LogiRec.**LrEOM**(*attr=0*)
A EOM marker record.

__init__(*attr=0*)
Base class constructor, theType and theAttr are bytes.

class TotalDepth.LIS.core.LogiRec.**LrEOMRead**(*theFile*)
A EOM marker record read from a file.

__init__(*theFile*)
Base class constructor, theType and theAttr are bytes.

class TotalDepth.LIS.core.LogiRec.**LrWithDateField**(*theType*, *theAttr*)
ABC for classes that have the YY/MM/DD date field.

__init__(*theType*, *theAttr*)
Base class constructor, theType and theAttr are bytes.

property ymd
Returns the YY/MM/DD date field to a year, month, day tuple or None.

class TotalDepth.LIS.core.LogiRec.**LrFileHeadTail**(*theType*, *theAttr*)
Parent class of FileHead, FileTail that have identical structure.

__init__(*theType*, *theAttr*)
Base class constructor, theType and theAttr are bytes.

read(*theFile*)
Read from a LIS physical file.

property contFileName
Continuation file name.

class TotalDepth.LIS.core.LogiRec.**LrFileHead**(*theType*, *theAttr*)
Specific class of File Head.

__init__(*theType*, *theAttr*)
Base class constructor, theType and theAttr are bytes.

property prevFileName
Previous file name.

class TotalDepth.LIS.core.LogiRec.**LrFileHeadRead**(*theFile*)
Specific class of File head read from a file.

__init__(*theFile*)
Base class constructor, theType and theAttr are bytes.

class TotalDepth.LIS.core.LogiRec.**LrFileTail**(*theType*, *theAttr*)
Specific class of File Tail.

__init__(*theType*, *theAttr*)
Base class constructor, theType and theAttr are bytes.

property nextFileName
Next file name.

class TotalDepth.LIS.core.LogiRec.**LrFileTailRead**(*theFile*)
Specific class of File tail read from a file.

__init__(*theFile*)
Base class constructor, theType and theAttr are bytes.

class TotalDepth.LIS.core.LogiRec.**LrReelTapeHeadTail**(*theType*, *theAttr*)
Parent class of Reel/Tape Head/Tail that have identical structure.

__init__(*theType*, *theAttr*)
Base class constructor, theType and theAttr are bytes.

class TotalDepth.LIS.core.LogiRec.**LrTapeHeadTail**(*theType*, *theAttr*)
Tape head or tail Logical Record.

__init__(*theType*, *theAttr*)
Base class constructor, theType and theAttr are bytes.

class TotalDepth.LIS.core.LogiRec.**LrTapeHead**(*theType*, *theAttr*)
Tape head Logical Record.

__init__(*theType*, *theAttr*)
Base class constructor, theType and theAttr are bytes.

property prevTapeName
Previous tape name.

class TotalDepth.LIS.core.LogiRec.**LrTapeHeadRead**(*theFile*)
Specific class of Tape head read from a file.

__init__(*theFile*)
Base class constructor, theType and theAttr are bytes.

class TotalDepth.LIS.core.LogiRec.**LrTapeTail**(*theType*, *theAttr*)
Tape tail Logical Record.

__init__(*theType*, *theAttr*)
Base class constructor, theType and theAttr are bytes.

property nextTapeName
Next tape name.

class TotalDepth.LIS.core.LogiRec.**LrTapeTailRead**(*theFile*)
Specific class of Tape tail read from a file.

__init__(*theFile*)
Base class constructor, theType and theAttr are bytes.

class TotalDepth.LIS.core.LogiRec.**LrReelHeadTail**(*theType*, *theAttr*)
Reel head or tail Logical Record.

__init__(*theType*, *theAttr*)
Base class constructor, theType and theAttr are bytes.

class TotalDepth.LIS.core.LogiRec.**LrReelHead**(*theType*, *theAttr*)
Reel head Logical Record.

__init__(*theType*, *theAttr*)
Base class constructor, theType and theAttr are bytes.

property prevReelName
Previous reel name.

class TotalDepth.LIS.core.LogiRec.**LrReelHeadRead**(*theFile*)
Specific class of Reel head read from a file.

__init__(*theFile*)
Base class constructor, theType and theAttr are bytes.

class TotalDepth.LIS.core.LogiRec.**LrReelTail**(*theType*, *theAttr*)
Reel tail Logical Record.

__init__(*theType*, *theAttr*)
Base class constructor, theType and theAttr are bytes.

property nextReelName
Next reel name.

class TotalDepth.LIS.core.LogiRec.**LrReelTailRead**(*theFile*)
Specific class of Reel tail read from a file.

__init__(*theFile*)
Base class constructor, theType and theAttr are bytes.

class TotalDepth.LIS.core.LogiRec.**LrMisc**(*theType*, *theAttr*)
Miscellaneous Logical Record.

__init__(*theType*, *theAttr*)
Base class constructor, theType and theAttr are bytes.

class TotalDepth.LIS.core.LogiRec.**LrMiscRead**(*theFile: TotalDepth.LIS.core.File.FileRead*)
Miscellaneous Logical Record read from a LIS file.

__init__(*theFile: TotalDepth.LIS.core.File.FileRead*)
Base class constructor, theType and theAttr are bytes.

class TotalDepth.LIS.core.LogiRec.**LrUnknown**(*theType*, *theAttr*)
Logical Record that does not fall into any other category.

__init__(*theType*, *theAttr*)
Base class constructor, theType and theAttr are bytes.

class TotalDepth.LIS.core.LogiRec.**LrUnkownRead**(*theFile: TotalDepth.LIS.core.File.FileRead*)
Logical Record that does not fall into any other category.

__init__(*theFile: TotalDepth.LIS.core.File.FileRead*)
Base class constructor, theType and theAttr are bytes.

TotalDepth.LIS.core.LogiRec.**COMPONENT_BLOCK_TABLE = 73**
Type of first Component Blocks in the table

TotalDepth.LIS.core.LogiRec.**COMPONENT_BLOCK_DATUM_BLOCK_START = 0**
Type of first Component Blocks in the Datum Block i.e. row

TotalDepth.LIS.core.LogiRec.**COMPONENT_BLOCK_DATUM_BLOCK_ENTRY = 69**
Component Block type that describes an entry in a Datum Block i.e. a cell

exception TotalDepth.LIS.core.LogiRec.**ExceptionLrTable**
Specialisation of exception for Table Logical Records.

exception TotalDepth.LIS.core.LogiRec.**ExceptionLrTableInit**
Table __init__() issues.

exception TotalDepth.LIS.core.LogiRec.**ExceptionLrTableInternaStructuresCorrupt**
Raised when there are inconsistencies with the IR of the table.

exception TotalDepth.LIS.core.LogiRec.**ExceptionLrTableCompose**
Table creation (not from file) issues.

exception TotalDepth.LIS.core.LogiRec.**ExceptionLrTableRow**
TableRow issues.

exception TotalDepth.LIS.core.LogiRec.**ExceptionLrTableRowInit**
TableRow __init__() issues.

exception TotalDepth.LIS.core.LogiRec.**ExceptionCbEngValInit**
CbEngVal.__init__() issues such as unknown rep code.

class TotalDepth.LIS.core.LogiRec.**CbEngVal**
Contains the data from a Component Block and has an EngVal

CB_TYPES = (73, 0, 69)
Allowable Component Block types

__init__()
Initialize self. See help(type(self)) for accurate signature.

__str__()
Return str(self).

setValue(*v*)
Sets the value to v.

lisBytes()
Return a bytes() array from the internal representation.

property value
The value of the Component Block or None.

property status
Returns True if the value is b'ALLO', False otherwise.

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

class TotalDepth.LIS.core.LogiRec.**CbEngValRead**(*theFile*)
Contains the data from a Component Block and has an EngVal read from a file.

__init__(*theFile*)
Initialise. This will raise a TypeError if theFile.unpack returns None i.e. when not enough data to create a Component Block.

class TotalDepth.LIS.core.LogiRec.**CbEngValWrite**(*t*, *v*, *m*, ***kwargs*)
A Component Block and has an EngVal created directly.

__init__(*t*, *v*, *m*, ***kwargs*)
Initialise component block with type, value, mnemonic and optional key words.

class TotalDepth.LIS.core.LogiRec.**TableRow**(*theCb*)
Represents a row of a table and consists of CbEngVal objects.

__init__(*theCb*)
Initialize self. See help(type(self)) for accurate signature.

genCells()
yields each CbEngVal.

__len__()
The number of cells in the row.

property value
The name of the row i.e. the value of block 0.

addCb(*theCb*)
Adds a component block onto the end of the row. Returns the mnemonic field from the component block.

__getitem__(*key*)
If key is an integer or slice this returns a CbEngVal by index(es). If key is a bytes() object then this returns a CbEngVal by label. May raise a KeyError or IndexError.

__contains__(*key*)
Returns True if this row has a column named key.

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

class TotalDepth.LIS.core.LogiRec.**LrTable**(*theType*, *theAttr*)
Table-like Logical Record.

__init__(*theType*, *theAttr*)
Base class constructor. theType, theAttr are byte objects i.e. integers 0 to 255

genRows()
yields each TableRow. TODO: parameter to sort or reverse.

genRowNames(*sort=0*)
yields each TableRow name. If sort > 0 then the results are sorted. If sort < 0 the results are reverse sorted

retRowByMnem(*m: TotalDepth.LIS.core.Mnem.Mnem*) → TotalDepth.LIS.core.LogiRec.TableRow
Returns the TableRow from a Mnem.Mnem object. i.e. the table row that has a b'MNEM' column whose value matches m. May raise a KeyError. Note: an IndexError would mean that self.__mnemRowIndex is corrupt.

property isSingleParam
True it this table is a list of single parameters (i.e. type 0 blocks).

property value
The name of the table or None i.e. the value of the first block.

__getitem__(*key*)
If key is an integer or slice this returns block by index(es). If key is a bytes() object then this returns row by label. May raise a KeyError or IndexError.

__contains__(*key*)
Returns True if this row has a row named key.

__len__()
Number of rows in the table.

rowLabels()
Returns dictionary view of row values (unordered).

rowMnems()
Returns dictionary view of row Mnem.Mnem objects (unordered).

colLabels()
Returns ordered super-set of column values. Not all rows have these.

startNewRow(*theCbEv*)
Starts a new row from the component block. Returns theCbEv.mnem value.

addDatumBlock(*theCbEv*)
Adds a component block to the last row. Returns theCbEv value.

genRowValuesInColOrder(*theRow*)
Yields table cells in a particular row in column order.

genLisBytes()
Yields chunks of binary LIS data (actually each component block).

class TotalDepth.LIS.core.LogiRec.**LrTableRead**(*theFile*)
A table-like Logical Record read from a LIS file.

__init__(*theFile*)
Base class constructor. theType, theAttr are byte objects i.e. integers 0 to 255

class TotalDepth.LIS.core.LogiRec.**LrTableWrite**(*theType*, *theName*, *theMnemS*, *theTable*)
Creates a table from internal Python data structures.

theType is the table type e.g. b'FILM'

theMnemS is the list of column names, theTable members must fit this size.

If an element of the table is a tuple or a list it is assumed to be (value, units).

__init__(*theType*, *theName*, *theMnemS*, *theTable*)
Construct a table from internal data that must be bytes/float/int. theType is the table type e.g. b'FILM' theMnemS is the list of column names, theTable members must fit this size. If an element of the table is a tuple or a list it is assumed to be (value, units).

exception TotalDepth.LIS.core.LogiRec.**ExceptionEntryBlock**
Specialisation of exception for Entry Blocks.

exception TotalDepth.LIS.core.LogiRec.**ExceptionEntryBlockSetInit**
Exception for EntryBlockSet.__init__().

class TotalDepth.LIS.core.LogiRec.**EntryBlock**

lisBytes()
Returns the LIS bytes for the Entry Block.

class TotalDepth.LIS.core.LogiRec.**EntryBlockRead**
An entry block read from a LIS file.

static __new__(*self*, *theFile*)
Create new instance of EntryBlock(type, size, repCode, value)

class TotalDepth.LIS.core.LogiRec.**EntryBlockSet**
Represents the set of Entry Blocks in a DFSR.

ATTR_MAP = {'absentValue': 12, 'dataType': 1, 'depthRepCode': 15, 'depthUnits': 14
Map of supported attributes i.e. those that are 'interesting'

EB_DOC = {0: 'Terminator, size is chosen to make total size even.', 1: 'Data Record '
Documentation about each Entry Block

BLOCKS_TO_SKIP = (10,)
List of block numbers that are not written out, also _setLisSizeEven() and lisSize() ignore these.

__init__()
Initialize self. See help(type(self)) for accurate signature.

__str__()
Return str(self).

__getattr__(*name*)
Returns the Entry Block corresponding to the name.

__getitem__(*key*)
This returns an Entry block by integer index.

property logUp
True if the logging direction is up (X decreasing). Note: not logUp and not logDown is possible to be True e.g. time log.

property logDown
True if the logging direction is down (X increasing). Note: not logUp and not logDown is possible to be True e.g. time log.

property xInc
True if the logging X increases (down or time log).

property opticalLogScale
Returns the Units corresponding to Entry Block 5: 'Optical Log Scale' This will be a LENG or TIME unit or empty if undefined.

lisSize()
Returns the totla size of the Entry Block set.

setEntryBlock(*theEb*)
Sets an Entry Block.

readFromFile(*theFile*)
Reads from a File object. NOTE: theFile.hasLd() must be True so the Logical Record Header must have been read already.

lisBytes()
Returns the Entry Block set as an array of bytes.

lisByteList()
Returns a list of bytes() objects, one for each entry block.

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

exception TotalDepth.LIS.core.LogiRec.**ExceptionDatumSpecBlock**
Specialisation of exception for Datum Specification Blocks.

class TotalDepth.LIS.core.LogiRec.**DatumSpecBlock**
This represents as Datum Specification Block.

__init__()
Initialize self. See help(type(self)) for accurate signature.

property isNull
True if this block is compromised in any way and should be ignored when composing a DFSR. The critical test is whether the data from this channel will be in the frame.

samples(*theSc*)
Returns the number of samples in a sub-channel.

bursts(*theSc*)
Returns the (samples, burst) for a sub-channel (bursts are invariant over sub-channels).

values()
Returns the total number of discrete values per frame for a single channel.

subChMnem(*theSc*)
Returns the curve Mnemonic for a particular sub-channel or None if unknown.

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

class TotalDepth.LIS.core.LogiRec.**DatumSpecBlockRead**(*theF*)
This represents as Datum Specification Block read from a file.

__init__(*theF*)
Initialize self. See help(type(self)) for accurate signature.

class TotalDepth.LIS.core.LogiRec.**LrDFSR**(*theType*, *theAttr*)
Data Format Specification Record.

__init__(*theType*, *theAttr*)
Base class constructor, theType and theAttr are bytes.

class TotalDepth.LIS.core.LogiRec.**LrDFSRRead**(*theFile*)
Data Format Specification Record read from a file.

__init__(*theFile*)
Base class constructor, theType and theAttr are bytes.

class TotalDepth.LIS.core.LogiRec.**LrNormalAlternateData**(*theType*, *theAttr*)
Class for Normal and Alternate data i.e. curve data.

__init__(*theType*, *theAttr*)
Base class constructor, theType and theAttr are bytes.

class TotalDepth.LIS.core.LogiRec.**LrNormalAlternateDataRead**(*theFile*)
Class for Normal and Alternate data i.e. curve data.

__init__(*theFile*)
Base class constructor, theType and theAttr are bytes.

class TotalDepth.LIS.core.LogiRec.**LrFactory**
Provides a despatch mechanism for generating Logical Records. This can be sub-classed to create different sets of Logical Records. For example the Indexer creates minimal logical records.

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

__init__()
Initialize self. See help(type(self)) for accurate signature.

class TotalDepth.LIS.core.LogiRec.**LrFactoryRead**
A factory for generating complete Logical Records from a file.

__init__()
Initialize self. See help(type(self)) for accurate signature.

retLrFromFile(*theFile*)
Given a LIS file this reads one Logical Record, and returns the appropriate Logical Record object or None.

TotalDepth.LIS.core.LogPass

This describes the LIS LogPass class that encapsulates a Log Pass. A log pass is the binary log data plus the record (DFSR) that describes the format of the binary data. Internally the binary data is converted to a FrameSet that wraps a numpy array.

The LogPass module is located in `src/TotalDepth/LIS/core` and can be imported thus:

```
from TotalDepth.LIS.core import LogPass
```

LogPass API Reference

The LogPass module contains a single class LogPass that is fundamental to the way that TotalDepth handles LIS binary data.

A Log Pass is defined as a single continuous recording of log data. "Main Log", "Repeat Section" are seperate examples of Log Pass(es).

exception TotalDepth.LIS.core.LogPass.**ExceptionLogPass**
Specialisation of exception for LogPass.

exception TotalDepth.LIS.core.LogPass.**ExceptionLogPassCtor**
Specialisation of exception for LogPass __init__().

exception TotalDepth.LIS.core.LogPass.**ExceptionLogPassNoType01Data**
Raised on access when there is no frame data loaded by addType01Data().

exception TotalDepth.LIS.core.LogPass.**ExceptionLogPassNoFrameSet**
Raised when FrameSet access is required but there has been no call to setFrameSet().

exception TotalDepth.LIS.core.LogPass.**ExceptionLogPassKeyError**
Raised when and internal KeyError is raised in a FrameSet access.

TotalDepth.LIS.core.LogPass.**EVENT_SEEK_LR = 'seekLr'**
Event to seek to the start of a new Logical Record

TotalDepth.LIS.core.LogPass.**EVENT_READ = 'read'**
Event to read bytes from frame

TotalDepth.LIS.core.LogPass.**EVENT_SKIP = 'skip'**
Event to skip bytes in frame

TotalDepth.LIS.core.LogPass.**EVENT_EXTRAPOLATE = 'extrapolate'**
Event to extrapolate X axis

class TotalDepth.LIS.core.LogPass.**LogPass**(*theDfsr*, *theFileId*, *xAxisIndex=0*)
Contains the information about a log pass which is defined as a LIS Data Format Specification Record (DFSR) plus any number of Type 0/1 Logical Records.

A LogPass must be created with a LIS DFSR and a LIS file ID. It can be then informed with addType01Data() which records the file positions of Type 0/1 records. When required the actual frame data can be populated with setFrameSet().

theDfsr - The DFSR.

theFileId - The ID of the file, this will be checked against any File object passed to me.

xAxisIndex - The index of the DSB block that describes the X axis, if indirect X this is ignored.

__init__(*theDfsr*, *theFileId*, *xAxisIndex=0*)
Constructed with an EFLR i.e. a DFSR

theDfsr - The DFSR.

theFileId - The ID of the file, this will be checked against any File object passed to me.

xAxisIndex - The index of the DSB block that describes the X axis, if indirect X this is ignored.

property dfsr
The DFSR used for construction.

property type01Plan
The Frame plan, a Type01Plan.FrameSetPlan() object.

property rle
The Logical Record Run Length Encoding as a Rle.RLEType01() object.

property frameSet
The Frame Set as a FrameSet.FrameSet() object or None if not initialised.

property iflrType
Returns the IFLR type that this LogPass describes.

property xAxisIndex
The channel index that corresponds to the X axis.

property isIndirectX
True if indirect X axis, False if explicit X axis channel.

property numBytes
The number of bytes in the underlying frame set (i.e. LIS) representation for the curve data. Returns None if the frame set is not initialised.

property nullValue
The NULL or absent value as specified in the DFSR.

longStr()
Returns a long (multiline) descriptive string.

__str__()
Returns a long (multiline) descriptive string.

frameSetLongStr()
Returns a long (multiline) descriptive string of the Frame Set or N/A if not initialised.

curveUnitsAsStr(*chMnem*)
Given a curve as a Mnem.Mnem() this returns the units as a string.

curveUnits(*chMnem*)
Given a curve as a Mnem.Mnem() this returns the units as a bytes object.

property xAxisFirstVal
The numerical value of the X axis of the first frame.

property xAxisLastVal
The numerical value of the X axis of the last frame.

property xAxisFirstEngVal
The EngVal (value, units) of the X axis of the first frame.

property xAxisLastEngVal
The EngVal (value, units) of the X axis of the first frame.

property xAxisSpacing
The numerical value of the X axis frame spacing. This is is the extreme range of X axis values divided by the number of frames - 1. This is +ve if the Xaxis increases, -ve if it decreases.

property xAxisUnits
The units of the X axis.

property totalFrames
The total number of frames.

property xAxisFirstValOptical
The numerical value of the X axis of the first frame in 'optical' units.

property xAxisLastValOptical
The numerical value of the X axis of the last frame in 'optical' units.

property xAxisSpacingOptical
The numerical value of the X axis frame spacing. This is is the extreme range of X axis values divided by the number of frames - 1.

property xAxisUnitsOptical
Returns the actual units to 'optical' i.e. user friendly units. For example if the Xaxis was in b'.1IN' the 'optical' units would be b'FEET".

frameFromX(*theEv*)
Returns the estimated frame number from the X axis, and EngValue.

hasOutpMnem(*theMnem*)
Returns True is theMnem is in this LogPass (i.e. is in the DFSR).

outpMnemS()
Returns all of the OUTP Mnems in this LogPass (i.e. is in the DFSR).

retExtChIndexList(*theMnemS*)
Returns a sorted, unique list of external channel indexes for a list of mnemonics. May raise a Exception-LogPassKeyError.

genFrameSetHeadings()
This generates a name and units for each value in a frame in the current frame set. It is useful for heading up a frame dump.

genFrameSetScNameUnit(*toAscii=True*)
This generates a name and units for sub-channel in a frame in the current frame set. It is useful for heading up a accumulate() dump.

genFrameSetChIndexScNameUnit(*toAscii=True*)
This generates an index, name and units for sub-channel in a frame in the current frame set. Like genFrameSetScNameUnit() but with the channel index as well.

addType01Data(*tellLr*, *lrType*, *lrLen*, *xAxisVal*)
Add an Type 0/1 logical record entry.

tellLr - the Logical Record start position in the file.

lrType - the type of the IFLR. Will raise an ExceptionLogPass if this does not match the DFSR.

lrLen - the length of the Logical record, not including the LRH.

xAxisVal - the value of the X Axis of the first frame of the Logical Record.

setFrameSetChX(*theFi*, *theChS*, *Xstart*, *Xstop*, *frStep=1*)
Loads a FramesSet using 'external' values from a File object. theChS is a list of channel mnemonics or None for all channels. Xstart, Xstop are EngVal of the start stop. frStep is not number of frames to step over, default means all frames.

setFrameSet(*theFile*, *theFrSl=None*, *theChList=None*)
Populates the frames set.

theFile - The File object. Will raise an ExceptionLogPass is the file ID does not match that in the constructor.

theFrSl - A slice object that describes when LogPass frames are to be used (default all). Will raise an ExceptionLogPass is there are no frames to load i.e. addType01Data() has not been called.

theChList - A list of external channel indexes (i.e. DSB block indexes) to populate the frame set with (default all).

genOutpPoints(*theMnem*)
Wrapper around the frameset generator, in fact this returns exactly that generator.

jsonObject()
Return an Python object that can be JSON encoded.

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

gen_mnemonic_units() → Sequence[Tuple[bytes, bytes]]
Yields a sequence of (mnemonic, units) for each of the recorded Datum Specification Blocks. This does not include sub-channels.

LogPass Usage

Typically a LogPass will be created directly or via a LIS FileIndexer. The latter technique is recommended as it is simpler.

Direct usage

LogPass objects are used via a three step process and this reflects the sequential process of reading a LIS file:

1. Construction with a DFSR object (once).
2. Updating with the location of the binary data records that contain the frame data (once per IFLR).
3. Populating the FrameSet with real values from the file (many times).

Construction with a DFSR

A LogPass object needs to be constructed with an instance of a LogiRec.LrDFSRRead. The LogPass will take a reference to the DFSR so the caller need not. The caller can always access the DFSR with the `.dfsr` property.

Assuming myF is a LIS File object positioned at the start of the DFSR:

```
myLp = LogPass.LogPass(LogiRec.LrDFSRRead(myF), 'FileID')
```

At this stage no FrameSet is created so the resource usage is minimal.

Add Binary Data Records

The method `addType01Data(tellLr, lrType, lrLen, xAxisVal)` adds the position of an IFLR that contain frame data. This method takes these arguments:

Argument	Description
tellLr	The Logical Record start position (as a `size_t`) in the file.
lrType	The type of the IFLR [0 | 1]. Will raise an ExceptionLogPass if this does not match the DFSR.
lrLen	The length of the Logical Record in bytes, not including the LRH.
xAxisVal	The value of the X Axis as a number of the first frame of the Logical Record.

This call should be made for each relevant IFLR as they are encountered in sequence.

At this stage no FrameSet is created and the arguments are encoded into an RLE object so the resource usage is minimal.

Populating the FrameSet

Once the preceding stages have been done the LogPass can be populated any number of times from the LIS file.

The method `setFrameSet(theFile, theFrSl=None, theChList=None)` constructs a new frame set with the appropriate values.

Argument	Description
theFile	The File object. Will raise an ExceptionLogPass is the file ID does not match that used in the constructor.
theFrSl	A slice object that describes when LogPass frames are to be used (default all). Will raise an ExceptionLogPass if there are no frames to load i.e. addType01Data() has not been called.
theChList	A list of external channel indexes (i.e. DSB block indexes) to populate the frame set with (default all).

Examples

Setting a frame set for all frames and all channels:

```
myLogPass.setFrameSet(myFile)
# The above line is equivalent to:
myLogPass.setFrameSet(myFile, theFrSl=None, theChList=None)
```

Setting a frame set for frames [0:16:4] i.e. frame indexes (0,4,8,12) and channels [0, 4, 7]

```
myLogPass.setFrameSet(myFile, theFrSl=slice(0,16,4), theChList=[0, 4, 7])
```

Any previous FrameSet will be freed and a new FrameSet of the appropriate dimension is created so the resource usage can be significant.

Using a LIS Indexer

A *TotalDepth.LIS.core.FileIndexer* [`TotalDepth.LIS.core.FileIndexer.FileIndex`] object will perform the necessary construction of a LogPass and the population with Logical Record positions with `addType01Data()` leaving the user just to call `setFrameSet()`. Thus a FileIndex object imposes low resource usage until the user wishes to populate the frame set.

An indexer will index a LIS file that has multiple Log Passes (e.g. repeat section, main log etc.) so the indexer provides an iteration method for Log Passes:

```
myFilePath = "Spam/Eggs.LIS"
# Open a LIS file, keepGoing for luck!
myFi = File.FileRead(myFilePath, theFileId=myFilePath, keepGoing=True)
# Index the LIS file
myIdx = FileIndexer.FileIndex(myFi)
# Ask the index for all the LogPass objects, these do not have the frameSet populated␣
↪yet
for aLp in myIdx.genLogPasses():
    # Load the FrameSet, all channels [None], all frames [None]
    aLp.logPass.setFrameSet(myFi, None, None)
    # The FrameSet is fully populated here...
    # Do something with it...
```

Testing

The unit tests are in `test/TestLogPass.py`. This should take under a second to execute.

Running the tests under coverage:

```
$ coverage run test/TestLogPass.py
TestClass.py script version "0.8.0", dated 10 Jan 2011
Author: Paul Ross
Copyright (c) Paul Ross

testSetUpTearDown (__main__.TestLogPass_LowLevel)
TestLogPass_LowLevel: Tests setUp() and tearDown(). ... ok
test_00 (__main__.TestLogPass_LowLevel)
TestLogPass_LowLevel.test_00(): Construction. ... ok
test_01 (__main__.TestLogPass_LowLevel)
```

(continues on next page)

(continued from previous page)

```
TestLogPass_LowLevel.test_01(): _sliceFromList(). ... ok
test_02 (__main__.TestLogPass_LowLevel)
TestLogPass_LowLevel.test_02(): exercise various properties. ... ok
test_10 (__main__.TestLogPass_LowLevel)
TestLogPass_LowLevel.test_10(): nullValue(). ... ok

8<-------------- snip ------------------>8

test_00 (__main__.TestLogPass_UpIndirect)
TestLogPass_UpIndirect.test_00(): 3 LR, 5 fr, 4 ch. setFrameSet() All. ... ok
test_01 (__main__.TestLogPass_UpIndirect)
TestLogPass_UpIndirect.test_01(): 3 LR, 5 fr, 4 ch. setFrameSet() theFrSl=slice(0,16,
↪2). ... ok
test_02 (__main__.TestLogPass_UpIndirect)
TestLogPass_UpIndirect.test_02(): 3 LR, 5 fr, 4 ch. setFrameSet() theFrSl=slice(2,4,
↪2). ... ok
test_03 (__main__.TestLogPass_UpIndirect)
TestLogPass_UpIndirect.test_03(): 3 LR, 5 fr, 4 ch. setFrameSet() theFrSl=slice(1,5,
↪2). ... ok
test_10 (__main__.TestLogPass_UpIndirect)
TestLogPass_UpIndirect.test_10(): genFrameSetHeadings() ... ok

----------------------------------------------------------------------
Ran 54 tests in 0.507s

OK
CPU time =    0.510 (S)
Bye, bye!

$ coverage report -m
Name     Stmts   Miss  Cover   Missing
-------------------------------------------------------------------------------------------
↪--------------------

8<-------------- snip ------------------>8

LogPass    273     27    90%   142, 167, 182, 187, 203, 216-218, 232, 258, 273-275,␣
↪319, 324, 412, 416, 457-459, 513, 527, 541, 557, 572, 646-647

8<-------------- snip ------------------>8

-------------------------------------------------------------------------------------------
↪--------------------
TOTAL     4586   1783    61%
```

TotalDepth.LIS.core.Mnem (Mnemonics)

Represents a MNEM (Mnemonic) as bytes. Created on May 26, 2011

@author: paulross

TotalDepth.LIS.core.Mnem.**ORDS_WS = (32, 9, 10, 13, 11, 12)**
A tuple of the ordinal values of whitespace characters

TotalDepth.LIS.core.Mnem.**ORDS_REPLACE = (0, 32, 9, 10, 13, 11, 12)**
A tuple of the ordinal values of characters that can be replaced with PAD_CHAR

TotalDepth.LIS.core.Mnem.**LEN_MNEM = 4**
Standard LIS mnemonic length

class TotalDepth.LIS.core.Mnem.**Mnem**(*m*, *len_mnem=4*)
Represents a four byte mnemonic where tailing nulls and spaces are not considered significant. This preserves original length but replaces trailing and whitespace characters with the PAD_CHAR.

m must be a bytes object or an 'ascii' str that can be converted to a bytes object.

If len_mnem is positive then m is truncated or padded as necessary to achieve that length.

If len_mnem is zero then all of m is considered significant, no padding is performed.

If len_mnem is less than zero then all characters of m are considered significant and padding up to -1*len_mnem characters of m is performed if m smaller than that.

__init__(*m*, *len_mnem=4*)
Constructor that prunes trailing nulls and spaces.

property m
The raw bytes of the mnemonic.

__str__()
String representation.

pStr(*strip=False*)
Returns a 'pretty' ascii string. If strip then trailing padding is removed.

__repr__()
repr() representation.

__hash__()
Hashing, this makes bytes() and Mnem() objects interchangeable.

__eq__(*other*)
True if self == other False otherwise. If other is not a Mnem it is coerced into one before the comparison is made..

__ne__(*other*)
True if self != other False otherwise. If other is not a Mnem it is coerced into one before the comparison is made..

__lt__(*other*)
True if self < other False otherwise. If other is not a Mnem it is coerced into one before the comparison is made..

__iter__()
Byte by byte iteration.

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

TotalDepth.LIS.core.PhysRec (Physical Record Handler)

Physical Record Attributes

Bit positions here are different from the LIS standard Section 2.3.1.1 and Figure 2.11 on page 2-17.

There they describe the PRL as bits 0-15 and thee PR attributes as bits 16-31.

Our attributes bits are from 15-0 so our attribute bits are 31 - the standard's.

Bit	Description
15	Unused, reserved
14	Physical record type (only 0 is defined)
13-12	00 - No checksum, 01 - 16bit checksum, 10, 11 - Undefined
11	Unused, reserved
10	If 1 File number is present in trailer
09	If 1 Record number is present in trailer
08	Unused
07	Unused, reserved
06	If 1 then a previous parity error has occurred
05	If 1 then a previous checksum error has occurred
04	Unused
03	Unused, reserved
02	Unused
01	If 1 there is a predecessor Physical Record
00	If 1 there is a succcessor Physical Record

exception TotalDepth.LIS.core.PhysRec.**ExceptionPhysRec**
Specialisation of exception for Physical Records.

exception TotalDepth.LIS.core.PhysRec.**ExceptionPhysRecEOF**
Physical Record unexpected EOF.

exception TotalDepth.LIS.core.PhysRec.**ExceptionPhysRecUndefinedChecksum**
Physical Record encountered undefined checksum bit.

exception TotalDepth.LIS.core.PhysRec.**ExceptionPhysRecUnknownType**
Encountered unknown type 1 Physical Record.

exception TotalDepth.LIS.core.PhysRec.**ExceptionPhysRecWrite**
Physical Record writing.

exception TotalDepth.LIS.core.PhysRec.**ExceptionPhysRecTail**
Physical Record Trailer exception.

TotalDepth.LIS.core.PhysRec.**PR_PRH_LEN_FORMAT = <Struct object>**
PR Header 4 bytes long, two big-endian 16 bit numbers The struct.Struct() format for the Physical Record Header

TotalDepth.LIS.core.PhysRec.**PR_PRH_ATTR_FORMAT = <Struct object>**
The struct.Struct() format for the Physical Record Header attributes

TotalDepth.LIS.core.PhysRec.**PR_PRH_LENGTH = 4**
The length of the Physical Record Header

TotalDepth.LIS.core.PhysRec.**PR_ATTRIBUTE_BITS = 16**
Number of bits in the 2 byte attributes

TotalDepth.LIS.core.PhysRec.**PR_SUCCESSOR_ATTRIBUTE_BIT = 0**
Successor bit position

TotalDepth.LIS.core.PhysRec.**PR_PREDECESSOR_ATTRIBUTE_BIT = 1**
Predessor bit position

TotalDepth.LIS.core.PhysRec.**PR_OLD_CHECK_ERROR_BIT = 5**
Checksum error bit position

TotalDepth.LIS.core.PhysRec.**PR_OLD_PARITY_ERROR_BIT = 6**
Parity error bit position

TotalDepth.LIS.core.PhysRec.**PR_RECORD_NUMBER_BIT = 9**
Bit position to indicate there is a record number in the trailer

TotalDepth.LIS.core.PhysRec.**PR_FILE_NUMBER_BIT = 10**
Bit position to indicate there is a file number in the trailer

TotalDepth.LIS.core.PhysRec.**PR_CHECKSUM_BIT = 12**
Bit position to indicate there is a 16bit checksum in the trailer

TotalDepth.LIS.core.PhysRec.**PR_CHECKSUM_UNDEFINED_BIT = 13**
Bit position to indicate checksum is undefined

TotalDepth.LIS.core.PhysRec.**PR_TYPE_BIT = 14**
Bit position to indicate Physical Record Type

TotalDepth.LIS.core.PhysRec.**PR_ATTRIBUTE_UNUSED_ONLY_MASK = 276**
Unused only bits - i.e. Unused but not Unused, reserved

TotalDepth.LIS.core.PhysRec.**PR_ATTRIBUTE_UNUSED_RESERVED_MASK = 34952**
Unused, reserved bit mask

TotalDepth.LIS.core.PhysRec.**PR_ATTRIBUTE_UNUSED_MASK = 0**
Unused and Unused, reserved bits, 0x899C

TotalDepth.LIS.core.PhysRec.**PR_PRT_REC_NUM_FORMAT = <Struct object>**
The struct.Struct() format for the Physical Record Trailer record number, unsigned short.

TotalDepth.LIS.core.PhysRec.**PR_PRT_REC_NUM_LEN = 2**
The length of the Physical Record Trailer for the record number

TotalDepth.LIS.core.PhysRec.**PR_PRT_REC_NUM_MIN = 0**
The minimum record number, two byte unsigned.

TotalDepth.LIS.core.PhysRec.**PR_PRT_REC_NUM_MAX = 65535**
The maximum record number, two byte unsigned.

TotalDepth.LIS.core.PhysRec.**PR_PRT_FILE_NUM_FORMAT = <Struct object>**
The struct.Struct() format for the Physical Record Trailer file number, unsigned short.

TotalDepth.LIS.core.PhysRec.**PR_PRT_FILE_NUM_LEN = 2**
The length of the Physical Record Trailer for the file number

TotalDepth.LIS.core.PhysRec.**PR_PRT_FILE_NUM_MIN = 0**
The minimum file number, two byte unsigned.

TotalDepth.LIS.core.PhysRec.**PR_PRT_FILE_NUM_MAX = 65535**
The maximum file number, two byte unsigned.

TotalDepth.LIS.core.PhysRec.**PR_PRT_CHECKSUM_FORMAT = <Struct object>**
The struct.Struct() format for the Physical Record Trailer checksum, unsigned short.

TotalDepth.LIS.core.PhysRec.**PR_PRT_CHECKSUM_LEN = 2**
The length of the Physical Record Trailer for the checksum

TotalDepth.LIS.core.PhysRec.**PR_MAX_LENGTH = 65535**
Maximum possible Physical Record length represented by an unsigned 16 bit int

class TotalDepth.LIS.core.PhysRec.**PhysRecBase**(*theFileId: str*, *keepGoing: bool*)
Base class for physical record read and write. TODO: Checksum reading, writing and testing.

__init__(*theFileId: str*, *keepGoing: bool*)
Constructor, initialise data common to child classes.

theFileId - The ID of the file being read, usually the path.

keepGoing - If True continue parsing files that are not standard compliant.

close()
Close the underlying stream, further operations will raise a ValueError.

strHeader(*inc_attributes_short: bool*)
Returns the header string to go at the top of a list of __str__(). If inc_attributes_short then the header is suitable for using the string created by attribute_str_short().

__str__()
Return str(self).

attribute_str() → str
Human readable attributes.

attribute_str_short() → str
Human readable attributes.

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

class TotalDepth.LIS.core.PhysRec.**PhysRecRead**(*theFile*, *theFileId: str = ''*, *keepGoing: bool = False*, *pad_modulo: int = 0*, *pad_non_null: bool = False*)
Specialisation of PhysRecBase for reading streams.

__init__(*theFile*, *theFileId: str = ''*, *keepGoing: bool = False*, *pad_modulo: int = 0*, *pad_non_null: bool = False*)
Constructor with a file path or file-like object. TODO: checksum.

pad_modulo - If non zero this is used to consume pad bytes. Usually this is 4 so that tell() will be increased to % pad_modulo.

pad_is_null - If True only consume pad bytes if they are ‘’.

__str__()
Return str(self).

readLrBytes(*theSize=-1*, *theLd=None*)
Reads theSize logical data bytes and returns it as a bytes() object. If theSize is -1 all logical data for this logical record is returned. If theLd is not None it is extended and returned, otherwise a new bytes() object is created and returned. Returns None on end of logical record.

skipLrBytes(*theSize=-1*)
Skips logical data and returns a count of skipped bytes. If theSize is -1 all logical data for this logical record is skipped positioning the stream at end of this logical record (Note: not the beginning of the next logical record). Returns 0 on end of logical record.

skipToNextLr()
Skips all remaining logical data, the PR trailer, and the next PR header. This positions stream at the start

of the next logical record. May raise an ExceptionPhysRecEOF if there is no further Logical or Physical record. Returns the number of Logical Data bytes skipped.

tellLr()
Returns the absolute file position of the start current Logical record. This value can be safely used in seekLr.

tell()
Returns the absolute position of the file.

seekLr(*offset*)
External setting of file position directly to the beginning of a PRH or TIF marker (if present). The caller is fully responsible for getting this right!

seekCurrentLrStart()
Setting the file position directly to the beginning of a PRH or TIF marker (if present) for the current Logical Rcord.

hasLd()
Returns True if there is logical data to be read, False otherwise. NOTE: This will return False on file initialisation and only return True once the Physical Record Header has been read.

ldRemaingInPr()
Returns the number of bytes remaining in this particular Physical Record. NOTE: The can be 0 and hasLd() be True if at the end of a Physical Record that has a successor record.

genLd()
A generator that produces a tuple of (logical data, isLrStart) where: logical data - A bytes() object for the logical data in the current PR. isLrStart - A boolean that is True of that LogicalData is the start of a logical record. NOTE: This rewinds the current state of this instance.

genPr()
A generator that iterates through all the Physical Records from the current position. This seeks through the Logical Data so should be as fast as possible.

class TotalDepth.LIS.core.PhysRec.**PhysRecTail**(*hasRecNum=False*, *fileNum=None*, *hasCheckSum=False*)
Represents Physical Record Tail fields.

__init__(*hasRecNum=False*, *fileNum=None*, *hasCheckSum=False*)
These three fields are Rep Code type 79 (16 bit signed integer).

__str__()
Return str(self).

static normalise_integer(*value: int*, *value_min: int*, *value_max*) → int
Given a value this normalises it to be within the range value_min <= value <= value_max.

property prhAttr
Returns the PRH attributes, to be or'd with any other attributes.

hasTail()
Returns True if any PRT field is present, False otherwise.

computeCheckSum(*theB*)
Computes the checksum of the byte stream.

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

class TotalDepth.LIS.core.PhysRec.**PhysRecWrite**(*theFile*, *theFileId=None*, *keepGoing=False*, *hasTif=False*, *thePrLen=65535*, *thePrt=<TotalDepth.LIS.core.PhysRec.PhysRecTail object>*)

Specialisation of PhysRecBase for writing to files.

__init__(*theFile*, *theFileId=None*, *keepGoing=False*, *hasTif=False*, *thePrLen=65535*, *thePrt=<TotalDepth.LIS.core.PhysRec.PhysRecTail object>*)

Constructor with: theFile - A file like object or string, if the latter it assumed to be a path. theFileId - File identifier, this could be a path for example. If None the RawStream will try and cope with it. keepGoing - If True we do our best to keep going. hasTif - Insert TIF markers or not. thePrLen - Max Physical Record length, defaults to the maximum possible length. thePrt - Physical Records Trailer settings (defaults to PhysRec.PhysRecTail()).

close()

Close the Physical Record Handler and the underlying stream.

writeLr(*theLr*)

Splits a Logical Record into into Physical Records and writes them to the stream. These Physical Records have trailer records if required. Returns the tell() of the start of the LR.

TotalDepth.LIS.core.RawStream (Raw Stream Handler)

The RawStream handler provides low-level stream I/O functionality.

exception TotalDepth.LIS.core.RawStream.**ExceptionRawStream**

Specialisation of exception for RawStream.

exception TotalDepth.LIS.core.RawStream.**ExceptionRawStreamEOF**

RawStream premature EOF.

class TotalDepth.LIS.core.RawStream.**RawStream**(*f*, *mode='rb'*, *fileId=None*)

Class that creates a I/O stream from a file path or file-like object and provides various low level functionality on it such as unpacking.

f - A file like object or string, if the latter it assumed to be a path.

mode - The file mode, defaults to binary read.

fileId - If f is a string is this is present then this is used as the file name. If f is not a string then f.name is used with fileId as a fallback.

__init__(*f*, *mode='rb'*, *fileId=None*)

Construct with: f - A file like object or string, if the latter it assumed to be a path. mode - The file mode, defaults to binary read. fileId - If f is a string is this is present then this is used as the file name. If f is not a string then f.name is used with fileId as a fallback.

__enter__()

Context Manager support.

__exit__(*exc_type*, *exc_value*, *traceback*)

Context manager finalisation, this closes the underlying stream.

property stream

Exposes the underlying stream.

tell()

Return the file's current position, like stdio's ftell.

seek (*offset*, *whence=0*)
Set the file's current position, like stdio's fseek. The whence argument is optional and defaults to os.SEEK_SET or 0 (absolute file positioning); other values are os.SEEK_CUR or 1 (seek relative to the current position) and os.SEEK_END or 2 (seek relative to the file's end). There is no return value. Not all file objects are seekable.

read (*theLen*)
Reads and returns theLen bytes.

write (*theB*)
Writes theB bytes.

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

close ()
Closes the underlying stream.

readAndUnpack (*theStruct*)
Reads from the stream and unpacks binary data according to the struct module format. This returns a tuple.

theStruct - A formated instance of struct.Struct().

packAndWrite (*theStruct*, **args*)
Packs binary data from args and writes it to the stream.

theStruct - A formated instance of struct.Struct().

args - The data to write.

TotalDepth.LIS.core.RepCode (Representation Codes)

This describes how TotalDepth.LIS handles representation codes, it covers several modules.

Design

There is a top level module `RepCode` that imports all sub-modules and provides some fundamental definitions. Sub-modules `pRepCode` and `cRepCOde` provide alternative implementations in Python or Cython.

`RepCode` Module

Description

This provides the main interface to Representation Code processing as well as some fundamental definitions.

This module is the top level module that imports other sub-modules implemented in Python, Cython or C/C++. The Cython implementations take precedence as this module imports the sub-modules thus:

```
from TotalDepth.LIS.core.pRepCode import *
from TotalDepth.LIS.core.cRepCode import *
```

Usage

It is designed to use thus:

```
from TotalDepth.LIS.core import RepCode
```

Reference

Module for translating raw LIS to an appropriate internal type for LIS Representation Codes ('RepCodes').

This aggregates pRepCode and cRepCode with the latter overwriting the former.

Created on 11 Nov 2010

@author: p2ross

exception TotalDepth.LIS.core.RepCode.**ExceptionRepCodeRead**
Exception for unknown Representation codes in look up tables.

exception TotalDepth.LIS.core.RepCode.**ExceptionRepCodeWrite**
Exception for unknown Representation codes in look up tables.

exception TotalDepth.LIS.core.RepCode.**ExceptionRepCodeNoLength**
Exception for indeterminate length when using Rep Code 65.

TotalDepth.LIS.core.RepCode.**read49**(*theFile*)
Returns a Representation Code 49 value from a File object.

TotalDepth.LIS.core.RepCode.**readBytes49**(*arg*)
Returns a Representation Code 49 value from a bytes object.

TotalDepth.LIS.core.RepCode.**read50**(*theFile*)
Returns a Representation Code 50 value from a File object.

TotalDepth.LIS.core.RepCode.**readBytes50**(*arg*)
Returns a Representation Code 50 value from a bytes object.

TotalDepth.LIS.core.RepCode.**read56**(*theFile*)
Returns a Representation Code 56 value from a File object.

TotalDepth.LIS.core.RepCode.**readBytes56**(*arg*)
Returns a Representation Code 56 value from a bytes object.

TotalDepth.LIS.core.RepCode.**read66**(*theFile*)
Returns a Representation Code 66 value from a File object.

TotalDepth.LIS.core.RepCode.**readBytes66**(*arg*)
Returns a Representation Code 66 value from a bytes object.

TotalDepth.LIS.core.RepCode.**writeBytes66**(*v*)
Converts a value to a Rep Code 66 and returns the bytes.

TotalDepth.LIS.core.RepCode.**read68**(*theFile*)
Returns a Representation Code 68 value from a File object.

TotalDepth.LIS.core.RepCode.**readBytes68**(*arg*)
Returns a Representation Code 68 value from a bytes object.

TotalDepth.LIS.core.RepCode.**writeBytes68**(*v*)
Converts a value to a Rep Code 68 and returns the bytes.

TotalDepth.LIS.core.RepCode.**read70**(*theFile*)
Returns a Representation Code 70 value from a File object.

TotalDepth.LIS.core.RepCode.**readBytes70**(*arg*)
Returns a Representation Code 70 value from a bytes object.

TotalDepth.LIS.core.RepCode.**read73**(*theFile*)
Returns a Representation Code 73 value from a File object.

TotalDepth.LIS.core.RepCode.**readBytes73**(*arg*)
Returns a Representation Code 73 value from a bytes object.

TotalDepth.LIS.core.RepCode.**writeBytes73**(*v*)
Converts a value to a Rep Code 73 and returns the bytes.

TotalDepth.LIS.core.RepCode.**read77**(*theFile*)
Returns a Representation Code 77 value from a File object.

TotalDepth.LIS.core.RepCode.**readBytes77**(*arg*)
Returns a Representation Code 77 value from a bytes object.

TotalDepth.LIS.core.RepCode.**read79**(*theFile*)
Returns a Representation Code 79 value from a File object.

TotalDepth.LIS.core.RepCode.**readBytes79**(*arg*)
Returns a Representation Code 79 value from a bytes object.

TotalDepth.LIS.core.RepCode.**readBytes130**(*by*)
Reads Dipmeter RepCode 130 and returns a list of (integer) values.

TotalDepth.LIS.core.RepCode.**readBytes234**(*by*)
Reads Dipmeter RepCode 234 and returns a list of (integer) values.

TotalDepth.LIS.core.RepCode.**readRepCode**(*theRc*, *theFile*, *theLen=None*)
Reads a Representation Code from the file and returns a value. If theRc is 65 (string) then theLen must be supplied, up to that length of bytes will be returned.

TotalDepth.LIS.core.RepCode.**readBytes**(*theRc*, *theB*, *theLen=None*)
Reads a Representation Code from the a bytes() object. If theRc is 65 (string) then theLen must be supplied, up to that length of bytes will be returned.

TotalDepth.LIS.core.RepCode.**writeBytes**(*v*, *r*)
Takes a value v and a Representation Code r and converts this to a bytes() object.

pRepCode Module

This contains Python implementations.

Usage

This module is not designed to imported directly, use `RepCode` instead. This module can be imported only for test purposes thus:

```
from TotalDepth.LIS.core import pRepCode
```

Reference

Python module for translating raw LIS to an appropriate internal type for LIS Representation Codes ('RepCodes').

Created on 11 Nov 2010

@author: p2ross

exception TotalDepth.LIS.core.pRepCode.**ExceptionRepCode**
Specialisation of exception for Representation codes.

exception TotalDepth.LIS.core.pRepCode.**ExceptionRepCodeUnknown**
Specialisation of exception for unknown Representation codes.

TotalDepth.LIS.core.pRepCode.**isInt**(*r*)
Returns True if the Rep Code is represented by an integer.

TotalDepth.LIS.core.pRepCode.**lisSize**(*r*)
Returns the size in bytes for a single instance of a representation code. Zero means variable length. May raise ExceptionRepCodeUnknown.

TotalDepth.LIS.core.pRepCode.**wordLength**(*r*)
Returns the word length in bytes used by a representation code. NOTE: This is subtly different from lisSize as it take into account dipmeter sub-channels Zero means variable length. May raise ExceptionRepCodeUnknown.

TotalDepth.LIS.core.pRepCode.**maxValue**(*r*)
Returns the maximum value for various representation codes. May raise ExceptionRepCodeUnknown.

TotalDepth.LIS.core.pRepCode.**minValue**(*r*)
Returns the minimum value for various representation codes. May raise ExceptionRepCodeUnknown.

TotalDepth.LIS.core.pRepCode.**minMaxValue**(*r*)
Returns a pair; (minimum value, maximum value) for various representation codes. May raise ExceptionRepCodeUnknown.

TotalDepth.LIS.core.pRepCode.**from49**(*theWord*)
Returns a double from Rep code 49 0x31, 16bit floating point representation. Value +153 is 0100 1100 1000 1000 or 0x4C88. Value -153 is 1011 0011 1000 1000 or 0xB388.

TotalDepth.LIS.core.pRepCode.**to49**(*theVal*)
Converts a double to Rep code 49 0x31, 16bit floating point representation. Value +153 is 0100 1100 1000 1000 or 0x4C88. Value -153 is 1011 0011 1000 1000 or 0xB388.

TotalDepth.LIS.core.pRepCode.**read49**(*theFile*)
Returns a Representation Code 49 value from a File object.

TotalDepth.LIS.core.pRepCode.**from50**(*theWord*)
Returns a double from Rep code 0x32, 32bit floating point representation. Value +153 is 0x00084C80 Value -153 is 0x0008B380

TotalDepth.LIS.core.pRepCode.**to50**(*theVal*)
Converts a double to Rep code 0x32, 32bit floating point representation.

TotalDepth.LIS.core.pRepCode.**read50**(*theFile*)
Returns a Representation Code 50 value from a File object.

TotalDepth.LIS.core.pRepCode.**from56**(*theWord*)
Returns an integer from Rep code 0x38, signed char representation.

TotalDepth.LIS.core.pRepCode.**to56**(*theVal*)
Converts a double to Rep code 0x38, signed char representation.

TotalDepth.LIS.core.pRepCode.**read56**(*theFile*)
Returns a Representation Code 56 value from a File object.

TotalDepth.LIS.core.pRepCode.**from66**(*theWord*)
Returns a Rep code 0x42, unsigned byte from theWord, expected to be an integer.

TotalDepth.LIS.core.pRepCode.**to66**(*theVal*)
Converts theVal to representation code 66 integer from theWord.

TotalDepth.LIS.core.pRepCode.**read66**(*theFile*)
Returns a Representation Code 66 value from a File object.

TotalDepth.LIS.core.pRepCode.**from68**(*theWord*)
Returns a double from a Rep code 68 word (a 32 bit integer).

TotalDepth.LIS.core.pRepCode.**to68**(*v*)
Returns Representation code 68 as a 32 bit integer from a double.

TotalDepth.LIS.core.pRepCode.**read68**(*theFile*)
Returns a Representation Code 68 value from a File object.

TotalDepth.LIS.core.pRepCode.**from70**(*theWord*)
Returns a double from a Rep code 0x46, 32bit fixed point.

TotalDepth.LIS.core.pRepCode.**to70**(*v*)
Returns Rep code 0x46, 32bit fixed point from an int or double.

TotalDepth.LIS.core.pRepCode.**read70**(*theFile*)
Returns a Representation Code 70 value from a File object.

TotalDepth.LIS.core.pRepCode.**from73**(*theWord*)
Returns a integer from a Rep code 0x49, 32bit signed integer.

TotalDepth.LIS.core.pRepCode.**to73**(*v*)
Returns Rep code 0x49, 32bit signed integer from an int or double.

TotalDepth.LIS.core.pRepCode.**read73**(*theFile*)
Returns a Representation Code 73 value from a File object.

TotalDepth.LIS.core.pRepCode.**from77**(*theWord*)
Returns a Rep code 0x4D, 8bit mask from theWord, expected to be an integer.

TotalDepth.LIS.core.pRepCode.**to77**(*theVal*)
Converts theVal to Rep code 0x4D, 8bit mask from theVal.

TotalDepth.LIS.core.pRepCode.**read77**(*theFile*)
Returns a Representation Code 77 value from a File object.

TotalDepth.LIS.core.pRepCode.**from79**(*theWord*)
Returns an integer from Rep code 0x4F, 16bit signed integer.

TotalDepth.LIS.core.pRepCode.**to79**(*theVal*)
Converts a double to Rep code 0x4F, 16bit signed integer.

TotalDepth.LIS.core.pRepCode.**read79**(*theFile*)
Returns a Representation Code 79 value from a File object.

cRepCode Module

This contains Cython implementations. By the import mechanism used be RepCode these implementations take precedence over the implementations in pRepCode.

Usage

This module is not designed to imported directly, use RepCode instead. This module can be imported only for test purposes thus:

```
from TotalDepth.LIS.core import cRepCode
```

Reference

Cython module for translating raw LIS to an appropriate internal type for LIS Representation Codes ('RepCodes').

Internal types are double, int or string.

TotalDepth.LIS.core.cRepCode.**from50**()
: Returns a double from Rep code 0x32, 32bit floating point representation. Value +153 is 0x00084C80 Value -153 is 0x0008B380

TotalDepth.LIS.core.cRepCode.**from56**()
: Returns a integer from a Rep code 56 word (a 8 bit signed integer).

TotalDepth.LIS.core.cRepCode.**from66**()
: Returns a integer from a Rep code 66 word (a 8 bit unsigned integer).

TotalDepth.LIS.core.cRepCode.**from68**()
: Returns a double from a Rep code 68 word (a 32 bit integer).

TotalDepth.LIS.core.cRepCode.**from73**()
: Returns a integer from a Rep code 73 word (a 32 bit integer).

TotalDepth.LIS.core.cRepCode.**from77**()
: Returns a integer from a Rep code 77 word (a 8 bit unsigned integer).

TotalDepth.LIS.core.cRepCode.**from79**()
: Returns a integer from a Rep code 79 word (a 16 bit integer).

TotalDepth.LIS.core.cRepCode.**to68**()
: Returns Representation code 68 as a 32 bit integer from a double.

Testing

The unit tests are in test/TestRepCode.py and test/TestRepCode68.py.

TotalDepth.LIS.core.Rle (Run Length Encoding)

The RLE module provides Run Length Encoding suitable for recording the file positions of a set of LIS Logical Records that represent frame data.

Created on 5 Jan 2011

class RLEItem()

A generic item in a Run Length Encoding list.

class RLE()

A generic Run Length Encoding list.

class RLEItemType01()

A specialised item in a Run Length Encoding list for type 0/1 LIS Logical Records.

class RLEType01()

A specialised Run Length Encoding list for type 0/1 LIS Logical Records.

API Reference

class TotalDepth.LIS.core.Rle.**RLEItemType01**(*tellLrPos*, *numFrameS*, *xAxisValue*)

Specialisation of an RLEItem for type 0 and type 1 LIS Logical Records. This is a RLEItem for the Logical Record but within we have a RLE() object for the X axis values.

tellLrPos - the position in the LIS file of the start of the Logical Record.

numFrameS - integer number of frames in this Logical Record.

xAxisValue - The value of the X axis of the first frame in the Logical Record.

__init__(*tellLrPos*, *numFrameS*, *xAxisValue*)

Initialize self. See help(type(self)) for accurate signature.

__str__()

String representation.

property numFrames

Total number of frames.

add(*tellLrPos*, *numFrameS*, *xAxisValue*)

Returns True if v has been absorbed in this entry. False means a new entry is required. A new entry is required if the tellLrPos is not regular or numFrameS is different than before.

values()

Generates ordered tuples of (value, number of frames, xaxis value).

value(*i*)

Returns the i'th tuple of (i, (value, number of frames, xaxis value)).

totalFrames()
Returns the total number of frames in this RLE item.

tellLrForFrame(*fNum*)
Returns the Logical Record position that contains the integer frame number.

xAxisFirst()
Returns the first X-axis value loaded.

xAxisLast()
Returns the first X-axis value loaded.

class TotalDepth.LIS.core.Rle.**RLEType01**(*theXUnits*, **args*)
Class that represents Run Length Encoding for type 0/1 logical records.

theXUnits - the X axis units.

__init__(*theXUnits*, **args*)
Constructor, optionally takes a unary function to convert all values with.

__str__()
String representation.

property xAxisUnits
X axis units.

property hasXaxisData
True if there is X axis data.

add(*tellLrPos*, *numFrameS*, *xAxisValue*)
Adds a value to this RLE object.

tellLrForFrame(*fNum*)
Returns the (lr_seek, frame_offset) i.e. the Logical Record position that contains the integer frame number and the number of excess frames.

totalFrames()
Returns the total number of frames in this RLE object.

xAxisFirst()
Returns the first X-axis value loaded or None if nothing loaded.

xAxisLast()
Returns the last X-axis value at the satart of the last Logical Record loaded or None if nothing loaded.

xAxisLastFrame()
Returns the last X-axis value of the last frame loaded or None if nothing loaded.

frameSpacing()
Returns the frame spacing from the first/last entries, or None if nothing loaded. Returned value is -ve for decreasing X (up logs), +ve for increasing X (down and time logs).

TotalDepth.LIS.core.TifMarker (TIF Marker Handling)

TIF Markers

These are 3x32bit little-endian integers at the beginning of each Physical Record.

Word[0]

This is the TIF set type, 0 for a normal TIF set. 1 for an EOF set.

Word[1]

This is the physical file location of the start of the previous set.

Word[2]

This is the physical file location of the start of the next set.

A dump looks like this:

```
           0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00000000: 00 00 00 00 00 00 00 00 4A 00 00 00 00 3E 00 00
00000010: 80 00 32 30 30 30 39 39 2E 44 41 54 20 20 20 20
00000020: 20 20 20 20 20 20 20 20 20 20 20 20 39 39 2F 30
00000030: 34 2F 32 39 20 20 31 30 32 34 20 20 20 20 20 20
00000040: 20 20 20 20 20 20 2E 20 20 20 00 00 00 00 00 00
00000050: 00 00 56 04 00 00 04 00 00 01 40 00 01 01 42 00
00000060: 02 01 42 00 03 04 49 00 00 02 34 04 01 42 01 05
...
00000450: 20 20 44 45 47 20 00 00 00 00 4A 00 00 00 62 08
00000460: 00 00 04 00 00 03 00 00 00 00 00 00 00 04 20 20
00000470: 00 01 44 20 20 20 20 20 31 30 41 20 20 20 20 20
...
00000860: 46 4E 00 00 00 00 56 04 00 00 6E 0C 00 00 04 00
00000870: 00 03 4F 52 20 20 20 20 20 20 20 20 20 20 20 20
...
...
...
0016E770: 18 00 BA 83 18 00 00 00 00 00 30 E5 16 00 C0 E7
0016E780: 16 00 00 3E 00 00 81 00 32 30 30 30 39 39 2E 44
...
0016E7B0: 20 20 20 20 20 20 20 20 20 20 20 20 2E 20 20 20
0016E7C0: 01 00 00 00 76 E7 16 00 CC E7 16 00 01 00 00 00
0016E7D0: C0 E7 16 00 D8 E7 16 00
```

Or:

```
tell()       TIF type           TIF back           TIF next
00000000:    00 00 00 00        00 00 00 00        00 00 00 4A
0000004A:    00 00 00 00        00 00 00 00        00 00 04 56
00000456:    00 00 00 00        00 00 00 4A        00 00 08 62
00000862:    00 00 00 00        00 00 04 56        00 00 0C 6E
...
```

(continues on next page)

(continued from previous page)

```
0016E776:    00 00 00 00        00 16 E5 30        00 16 E7 C0
0016E7C0:    00 00 00 01        00 16 E7 76        00 16 E7 CC
0016E7CC:    00 00 00 01        00 16 E7 C0        00 16 E7 D8
0016E7D8: EOF
```

exception TotalDepth.LIS.core.TifMarker.**ExceptionTifMarker**
Specialisation of exception for Physical Records.

TotalDepth.LIS.core.TifMarker.**TIF_WORD_BYTES = 4**
Number of bytes in a TIF word

TotalDepth.LIS.core.TifMarker.**TIF_NUM_WORDS = 3**
Number of words in a TIF marker

TotalDepth.LIS.core.TifMarker.**TIF_TOTAL_BYTES = 12**
Number of bytes in a TIF marker

TotalDepth.LIS.core.TifMarker.**TIF_WORD_FORMAT = <Struct object>**
struct.Struct() format for a TIF word

TotalDepth.LIS.core.TifMarker.**TIF_WORD_FORMAT_WRONG_SEX = <Struct object>**
struct.Struct() format for a TIF word written wrongly as little-endian

TotalDepth.LIS.core.TifMarker.**TIF_WORD_ALL_FORMAT_WRONG_SEX = <Struct object>**
struct.Struct() format for a TIF marker written wrongly as little-endian

TotalDepth.LIS.core.TifMarker.**TIF_FIRST_WORD_LIMIT = 65547**
The maximum possible size of the first 'next' word. If larger than this then the words are written wrongly as little-endian and need to be reversed This is calculated as the maximum PR length + TIF bytes.

class TotalDepth.LIS.core.TifMarker.**TifMarkerBase**(*raiseOnError=True*)
Base class for TIF markers.

__init__(*raiseOnError=True*)
Constructor, initialises internals.

strHeader()
Header string for an ASCII dump.

__str__()
String representation.

markers()
Current values of markers as a tuple of three integers.

property eof
True if I have encountered a EOF marker.

reset()
Resets the TIF markers to all zero, this means hasPrevious is False.

reportError(*theMsg*)
Reports the error. I constructed with raiseOnError as True this will raise a ExceptionTifMarker otherwise it will write the error to the log.

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

class TotalDepth.LIS.core.TifMarker.**TifMarkerRead**(*theStream*, *allowPrPadding=False*)
Class for reading TIF markers. This will automatically determine if TIF markers are present and automatically correct ill-formed little-endian TIF markers.

theStream - the file stream.

allowPrPadding - If True this will consume spurious padding bytes after the Physical Record tail i.e. the TIF markers determine the Physical Record structure rather than the Physical Record Headers.

__init__(*theStream*, *allowPrPadding=False*)
Constructor, initialises internals. allowPrPadding - If true this allows padding bytes after the PRT.

property hasPrevious
True if a Physical Record has been read, cleared on reset().

reset()
Calling reset() means that the caller is probably randomly accessing the file so we can not error check the previous marker in the same way that we can if we are reading the file linearly.

read(*theStream*)
Read TIF markers from a RawStream object. Returns the stream tell() or None of the start of the TIF marker. This is not necessarily the same as the stream tell() seen by the caller as we might consume PR padding.

class TotalDepth.LIS.core.TifMarker.**TifMarkerWrite**
Class for writing TIF markers.

__init__()
Constructor, initialises internals.

write(*theStream*, *theLen*)
Write TIF markers to a RawStream object. theLen must be the length of the Physical Record including the PRH and PRT.

close(*theStream*)
Write TIF EOF markers.

TotalDepth.LIS.core.Type01Plan (Binary Frame Data Random Access)

Given a DFSR the FrameSetPlan gives offsets to any part of the Logical Record for any frame and channel.

exception TotalDepth.LIS.core.Type01Plan.**ExceptionFrameSetPlan**
Specialisation of exception for FrameSetPlan.

exception TotalDepth.LIS.core.Type01Plan.**ExceptionFrameSetPlanNegLen**
Specialisation of exception for negative/too small length arguments to FrameSetPlan.

exception TotalDepth.LIS.core.Type01Plan.**ExceptionFrameSetPlanOverrun**
Exception channel number greater than available.

TotalDepth.LIS.core.Type01Plan.**EVENT_READ = 'read'**
Read event

TotalDepth.LIS.core.Type01Plan.**EVENT_SKIP = 'skip'**
Skip event

TotalDepth.LIS.core.Type01Plan.**EVENT_EXTRAPOLATE = 'extrapolate'**
Extrapolate event

class TotalDepth.LIS.core.Type01Plan.**FrameSetPlan**(*dfsr*)
Given a DFSR the FrameSetPlan gives offsets to any part of the frame set within a Logical Record.

NOTE: All offsets, lengths etc. are relative to the end of the LRH i.e. add LR_HEADER_LENGTH to get absolute Logical Data position.

__init__(*dfsr*)
Initialize self. See help(type(self)) for accurate signature.

__str__()
Return str(self).

property indirectSize
The size, in bytes of the indirect X axis value. 0 for explicit X axis.

property frameSize
Frame size in bytes.

property numChannels
Number of channels.

channelSize(*i*)
The size of the given channel.

numFrames(*recLen*)
Returns the number of frames that will fit into the record length. May raise a ExceptionFrameSetPlan. May raise ExceptionFrameSetPlanNegLen if arguments are negative.

NOTE: record length should not include size of LRH.

chOffset(*frame*, *ch*)
Returns the offset into the LR (after LRH) to the start of a particular channel and frame.

Will raise ExceptionFrameSetPlanNegLen if arguments are negative or an IndexError if ch out of range.

skipToEndOfFrame(*ch*)
Returns the skip distance into the LR to the end of frame after a particular channel.

genOffsets(*theChIndexS*)
Yields an indefinite set of (frame, channel, offset) values for the set of channels. Will raise ExceptionFrameSetPlanNegLen if any channel negative. Will raise KeyError if any channel index is out of range.

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

genEvents(*theFSlice*, *theChIndexS*)
For a single Logical Record type 0/1 this yields an set of events.

theFSlice is a frame slice object (default start=0, step=1), step 0 is interpreted as step 1.

theChIndexS is as list of indexes of channels that need to be read.

The events are 5 member tuples: (event_type, size, frame, channel_start, channel_stop)

event_type is 'skip' or 'read' or 'extrapolate':

skip: ('skip', size, frame_number, channel_start, channel_stop)

read: ('read', size, frame_number, channel_start, channel_stop)

extrapolate: ('extrapolate', frames, frame_number, None, None)

NOTE: No 'seek', 0, ... events are generated.

size is the number of bytes to read or skip or, for 'extrapolate' the number of frames to extrapolate the implied X axis.

frame_number is the frame number in the predicted Logical Record.

In the case of 'skip' or 'extrapolate' its is the frame number in the Logical Record being moved to. The frame number can be None if there is no prediction, for example when reading an indirect X at the beginning of an LR.

channel_start, channel_stop are the channel numbers (inclusive) in the DFSR DSB block None being the indirect X axis.

Thus: ('read', size, frame_number, None, 3) Means read size bytes and interpret them as channels: None, 0, 1, 2, 3. 'extrapolate' events mean project the Xaxis forward by the specified number of frames (this will not be bestrided with two seek events but preceded or followed by a single seek event (if any). 'read' and 'seek' events are synchronous as are 'read' and 'extrapolate'. However 'seek' and 'extrapolate' are asynchronous i.e. 'seek'-'extrapolate' is equivalent to 'extrapolate'-'seek'.

Will raise ExceptionFrameSetPlanNegLen if any channel negative or fStop < fStart. Will raise IndexError if any channel index is out of range.

Be aware: This code is tricky.

TotalDepth.LIS.core.Units (Unit Conversion)

Provides unit conversion for LIS79.

The __RAW_UNIT_MAP is the master map from which all other data is derived. Its format is as follows:

key - The unit category as four bytes representing uppercase ASCII characters. value - A tuple of three fields:

- [0] - Descriptive string of the unit category.
- [1] - The base unit name as four bytes representing uppercase ASCII characters.
- [2] - A tuple the contents of which is a four or five item tuple:
  - **If four members:**
    - [2][0] - The unit name as four bytes representing uppercase ASCII characters.
    - [2][1] - The multiplier as a float.
    - [2][2] - Descriptive string of the units.
    - **[2][3] - The unit name that this is an alternate for as four bytes** representing uppercase ASCII characters, or four spaces.
  - **If five members:**
    - [2][0] - The unit name as four bytes representing uppercase ASCII characters.
    - [2][1] - The multiplier as a float.
    - [2][2] - The offset as a float.
    - [2][3] - Descriptive string of the units.
    - **[2][4] - The unit name that this is an alternate for as four bytes** representing uppercase ASCII characters or four spaces.

The unit name should also be unique.

TODO: Clean up units by making reciprocal e.g. 1/6.0 rather than 0.166666...

TODO: Check each unit for errors.

exception TotalDepth.LIS.core.Units.**ExceptionUnits**
Specialisation of exception for Unit conversion.

exception TotalDepth.LIS.core.Units.**ExceptionUnitsUnknownUnit**
When a unit does not exist.

exception TotalDepth.LIS.core.Units.**ExceptionUnitsUnknownCategory**
When a unit category does not exist.

exception TotalDepth.LIS.core.Units.**ExceptionUnitsNoUnitInCategory**
When a unit does not exist in a category.

exception TotalDepth.LIS.core.Units.**ExceptionUnitsMissmatchedCategory**
When a two units do not exist in the same category.

class TotalDepth.LIS.core.Units.**UnitConvertCategory**(*theCat*, *theDesc*, *theBaseUnitName*, *theUnitS*)
Internal module data structure that represents a category of units such as linear length.

theCat is the unit category.

theDesc is the description of that category.

theBaseUnitName is the name of the base units for the category. For example for linear lenght this is b'M '.

theUnitS is a list of unit names.

__init__(*theCat*, *theDesc*, *theBaseUnitName*, *theUnitS*)
Initialize self. See help(type(self)) for accurate signature.

units()
Reuturns a list of unit names for this category.

unitConvertor(*u*)
Returns a UnitConvert object corresponding to the name u. Will raise a ExceptionUnitsNoUnitInCategory if not found.

convert(*v*, *u_1*, *u_2*)
Returns a value converted from one units to another. e.g. convert(1.2, "FEET", "INCH")

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

class TotalDepth.LIS.core.Units.**UnitConvert**(*tup*)
Internal data structure for this module that representas a particular unit of measure. Takes a 4 or 5 member tuple from __RAW_UNIT_MAP.

__init__(*tup*)
Initialize self. See help(type(self)) for accurate signature.

convert(*val*, *other*)
Convert a value from me to the other where other is a UnitConvert object.

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

TotalDepth.LIS.core.Units.**unitCategories**()
Returns a list of the unit categories.

TotalDepth.LIS.core.Units.**hasUnitCategory**(*c*)
Returns True if I have that unit category e.g. b"TIME".

TotalDepth.LIS.core.Units.**hasUnit**(*u*)
Returns True if I have that unit e,g, b"FEET".

TotalDepth.LIS.core.Units.**category**(*unit*)
Returns the category of the unit. May raise a ExceptionUnitsUnknownUnit.

TotalDepth.LIS.core.Units.**categoryDescription**(*theCat*)
Returns the description of a unit category.

TotalDepth.LIS.core.Units.**units**(*theCat=None*)
Returns an unordered list of unit names. If category is None all unit names are returned, otherwise the unit

names for a particular category are returned. This may raise a ExceptionUnitsUnknownCategory if the category does not exist.

TotalDepth.LIS.core.Units.**retUnitConvertCategory**(*c*)
Returns a UnitConvertCategory object for the category. May raise a ExceptionUnits or descendent.

TotalDepth.LIS.core.Units.**retUnitConvert**(*u*)
Returns a UnitConvert object for the unit. May raise a ExceptionUnits or descendent.

TotalDepth.LIS.core.Units.**unitDescription**(*u*)
Returns the description of the unit. e.g. Given ".1IN" returns "Tenth-inches". May raise a ExceptionUnits or descendent.

TotalDepth.LIS.core.Units.**realUnitName**(*u*)
Returns the real unit name or None if u is the 'real' unit e.g. the 'real' unit name for b"FT " is b"FEET". May raise a ExceptionUnits or descendent.

TotalDepth.LIS.core.Units.**convert**(*v*, *u_1*, *u_2*)
Returns a value converted from one units to another. e.g. convert(1.2, b"FEET", b"INCH").

Will raise an ExceptionUnitsUnknownUnit if either unit is unknown.

Will raise an ExceptionUnitsMissmatchedCategory is both units doe not belong is the same unit category.

TotalDepth.LIS.core.Units.**opticalUnits**(*u*)
If possible returns the 'optical' units i.e. user friendly units. For example the 'optical' units of b'.1IN' are b'FEET'. Failure returns the argument.

Examples

Converting bytes objects:

```
from TotalDepth.LIS.core import Units

v = Units.convert(1.0, b"M   ", b"FEET")
# v is now 3.281
```

Testing

The unit tests are in `test/TestUnits.py`.

TotalDepth.LIS.DumpFrameSet

Reads a LIS file and writes out separated values of each frame.

Created on 25 Mar 2011

TotalDepth.LIS.DumpFrameSet.**dump_frame_sets**(*fp*, *keep_going*, *output_frames: bool*, *output_summary: bool*, *channels*, *seperator: str = ' '*)
Dump the frame values to stdout. channels is a set of Mnems, if non-empty then only these channels, if present, are written out.

TotalDepth.LIS.Index

Indexes LIS files and reports performance.

Created on 24 Feb 2011

@author: p2ross

Indexing errors on LIS files:

[34] TotalDepth.LIS.core.TifMarker.ExceptionTifMarker: TIF read() expected 0x50, got tell: 0x4A, Shortfall: 0x6 Fixed.

[24] TotalDepth.LIS.core.LogiRec.ExceptionEntryBlock: EntryBlockSet.setEntryBlock(): type 10 excluded from EntryBlockSet Fixed.

[2] TotalDepth.LIS.core.Type01Plan.ExceptionFrameSetPlan: Can not fit integer number of frames length 120 into LR length 824, modulo 104 [indirect size 0]. Two file have different problems:

13576.S1

W:openLISsrcTotalDepth.LIS>python Index.py -rk -l40 ..... pLogicTestDataLIS]76.S1 ... TotalDepth.LIS.core.Type01Plan.ExceptionFrameSetPlan: Can not fit integer number of frames length 120 into LR length 824, modulo 104 [indirect size 0].

Looks like the last PR is truncated: ... TIF True >: 0x 0 0x 19006 0x 197b6 PR: 0x 193de 972 0x9600 962 0x006c 0x0001 0xa2e1 0x00 0x00 [962] TIF True >: 0x 0 0x 193de 0x 19b06 PR: 0x 197b6 836 0x9600 826 0x006d 0x0001 0x0304 0x00 0x00 [826] Missing 962-826 bytes 136 bytes.

13610.S1

W:openLISsrcTotalDepth.LIS>python Index.py -rk -l40 ..... pLogicTestDataLIS^10.S1 ... TotalDepth.LIS.core.Type01Plan.ExceptionFrameSetPlan: Can not fit integer number of frames length 7176 into LR length 13354, modulo 6178 [indirect size 0].

This looks like a bad PR header at 0x3a986 that has set a successor bit: W:openLISsrcTotalDepth.LIS>python ScanPhysRec.py ..... pLogicTestDataLIS^10.S1 Cmd: ScanPhysRec.py ..... pLogicTestDataLIS^10.S1 TIF ? : Type Back Next PR: tell() Length Attr LD_len RecNum FilNum ChkSum LR Attr [Total LD] TIF True >: 0x 0 0x 0 0x 4a PR: 0x 0 62 0x8000 58 —— —— —— 0x80 0x00 [58] TIF True >: 0x 0 0x 0 0x 3ac PR: 0x 4a 854 0x8000 850 —— —— —— 0x40 0x00 [850] ... TIF True >: 0x 0 0x 390f4 0x 395ae PR: 0x 394ec 182 0x8001 178 —— —— —— 0x00 0x00 TIF True >: 0x 0 0x 394ec 0x 399a6 PR: 0x 395ae 1004 0x8003 1000 —— —— —— + —- —- TIF True >: 0x 0 0x 395ae 0x 39d9e PR: 0x 399a6 1004 0x8003 1000 —— —— —— + —- —- TIF True >: 0x 0 0x 399a6 0x 3a196 PR: 0x 39d9e 1004 0x8003 1000 —— —— —— + —- —- TIF True >: 0x 0 0x 39d9e 0x 3a58e PR: 0x 3a196 1004 0x8003 1000 —— —— —— + —- —- TIF True >: 0x 0 0x 3a196 0x 3a986 PR: 0x 3a58e 1004 0x8003 1000 —— —— —— + —- —- TIF True >: 0x 0 0x 3a58e 0x 3ad7e PR: 0x 3a986 1004 0x8003 1000 —— —— —— + —- —- 2011-03-09 19:50:13,710 WARNING Physical record at 0x3AD7E is successor but has no predecessor bit set.

TIF True >: 0x 0 0x 3a986 0x 3ae40 PR: 0x 3ad7e 182 0x8001 178 —— —— —— + —- —- TIF True >: 0x 0 0x 3ad7e 0x 3b238 PR: 0x 3ae40 1004 0x8003 1000 —— —— —— + —- —- TIF True >: 0x 0 0x 3ae40 0x 3b630 PR: 0x 3b238 1004 0x8003 1000 —— —— —— + —- —- TIF True >: 0x 0 0x 3b238 0x 3ba28 PR: 0x 3b630 1004 0x8003 1000 —— —— —— + —- —- TIF True >: 0x 0 0x 3b630 0x 3be20 PR: 0x 3ba28 1004 0x8003 1000 —— —— —— + —- —- TIF True >: 0x 0 0x 3ba28 0x 3c218 PR: 0x 3be20 1004 0x8003 1000 —— —— —— + —- —- TIF True >: 0x 0 0x 3be20 0x 3c610 PR: 0x 3c218 1004 0x8003 1000 —— —— —— + —- —- TIF True >: 0x 0 0x 3c218 0x 3ca08 PR: 0x 3c610 1004 0x8002 1000 —— —— —— + —- —- [13356]

[1] TotalDepth.LIS.core.pRepCode.ExceptionRepCodeUnknown: Unknown representation code: 0 Fixed by being a bit more cautious about dealing with DSB blocks that are 'null'.

TotalDepth.LIS.Index.**index_dir_single_process**(*d*, *r*, *t*, *v*, *k*) → Dict[str, TotalDepth.LIS.Index.IndexTimer]
Recursively process a directory using a single process.

TotalDepth.LIS.Index.**generate_file_paths**(*d*, *r*)
Generates file paths, recursive if necessary.

TotalDepth.LIS.LisToHtml

Created on Jun 14, 2011

@author: paulross

class TotalDepth.LIS.LisToHtml.**FileInfo**(*pathIn*, *pathOut*, *lisSize*, *numLr*, *cpuTime*, *exception*)

property pathIn
Alias for field number 0

property pathOut
Alias for field number 1

property lisSize
Alias for field number 2

property numLr
Alias for field number 3

property cpuTime
Alias for field number 4

property exception
Alias for field number 5

__str__()
Return str(self).

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *pathIn: str*, *pathOut: str*, *lisSize: int*, *numLr: int*, *cpuTime: float*, *exception: bool*)
Create new instance of FileInfo(pathIn, pathOut, lisSize, numLr, cpuTime, exception)

__repr__()
Return a nicely formatted representation string

class TotalDepth.LIS.LisToHtml.**LisToHtml**(*fpIn*, *fpOut*, *recursive*, *keepGoing*, *accCh=True*)
Takes an input path, output path and generates HTML file(s) form LIS.

__init__(*fpIn*, *fpOut*, *recursive*, *keepGoing*, *accCh=True*)
Write an HTML page about a LIS file. If accChan is True a summary table of the data is written.

TotalDepth.LIS.LisToHtml.**processFile**(*fpIn*, *fpOut*, *keepGoing*) → TotalDepth.LIS.LisToHtml.IndexSummary
Used by the multiprocessing code.

TotalDepth.LIS.PlotLogPasses

Created on May 23, 2011

@author: p2ross

class TotalDepth.LIS.PlotLogPasses.**IndexTableValue**(*scale*, *evFirst*, *evLast*, *curves*, *numPoints*, *outPath*)

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *scale*, *evFirst*, *evLast*, *curves*, *numPoints*, *outPath*)
Create new instance of IndexTableValue(scale, evFirst, evLast, curves, numPoints, outPath)

__repr__()
Return a nicely formatted representation string

property curves
Alias for field number 3

property evFirst
Alias for field number 1

property evLast
Alias for field number 2

property numPoints
Alias for field number 4

property outPath
Alias for field number 5

property scale
Alias for field number 0

class TotalDepth.LIS.PlotLogPasses.**PlotLogInfo**
Class that collates information about the results of plotting log passes. This can, for example, write out an index.html page with links to SVG pages.

__init__()
Initialize self. See help(type(self)) for accurate signature.

__str__()
Return str(self).

addPlotResult(*theInPath*, *theOutPath*, *theLpIdx*, *theFilmID*, *theScale*, *theEvFirst*, *theEvLast*, *theCurveS*, *ptsPlotted*)
Adds a successful plot. theInPath - The file path to the input file. theOutPath - The file path to the output file. theLpIdx - Integer index of the LogPass in the input file. theFilmID - The FILM ID as a Mnem. theScale - Plot scale as an number. theEvFirst - The first X axis as an EngVal. theEvLast - The last X axis as an EngVal. theCurveS - A list of Mnem of the curves plotted. ptsPlotted - Number of points plotted.

writeHTML(*theFilePath*, *theDesc*)
Write the index.html table.

retRelPath(*d*, *f*)
Given directory d and file path of f this returns relative path to f from d.

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

class TotalDepth.LIS.PlotLogPasses.**PlotLogPasses**(*fpIn*, *fpOut*, *recursive=False*, *keepGoing=True*, *lgFormatS=None*, *apiHeader=False*)

Takes an input path, output path and generates SVG file(s) from LIS.

__init__(*fpIn*, *fpOut*, *recursive=False*, *keepGoing=True*, *lgFormatS=None*, *apiHeader=False*)

Constructor.

fpIn and fpOut are file or directory paths. fpOut will be created if necessary.

recursive is a flag to control directory recursion.

keepGoing is a flag passed to the LIS File.FileRead object.

lgFormatS is a list of strings the correspond to the LgFormat UniqueId XML attribute. If absent the LIS file FILM/PRES etc. tables are used.

apiHeader is a flag to control whether a API header is extracted from CONS tables is to be plotted on the top of the log.

__weakref__

list of weak references to the object (if defined)

TotalDepth.LIS.PlotLogPasses.**plotLogPassesMP**(*dIn*, *dOut*, *fnMatch*, *recursive*, *keepGoing*, *lgFormatS*, *apiHeader*, *jobs*)

Multiprocessing code to plot log passes. Returns a PlotLogInfo object.

TotalDepth.LIS.ProcLISPath

Created on Jun 14, 2011

@author: paulross

exception TotalDepth.LIS.ProcLISPath.**ExceptionProcLisPath**

class TotalDepth.LIS.ProcLISPath.**ProcLISPathBase**(*fpIn*, *fpOut*, *recursive*, *keepGoing*)

Takes an input path, output path and processes LIS files.

__init__(*fpIn*, *fpOut*, *recursive*, *keepGoing*)

Initialize self. See help(type(self)) for accurate signature.

__weakref__

list of weak references to the object (if defined)

TotalDepth.LIS.ProcLISPath.**procLISPath**(*dIn*, *dOut*, *fnMatch*, *recursive*, *keepGoing*, *jobs*, *fileFn*, *resultObj=None*)

Multiprocessing code to process LIS files. dIn, dOut are directories.

fnMatch is a glob string.

recursive is a boolean to control recursion.

keepGoing is passed to fileFn

jobs is number of jobs; -1 single process, 0 number of available CPUs

fileFn is the operational function that will take a tuple of: (fIn, fOut, keepGoing) and return a result that can be added to the resultObj or None. This should not raise.

resultObj is accumulation of the results of fileFn or None, this it returned.

TotalDepth.LIS.ProcLISPath.**procLISPathMP**(*dIn*, *dOut*, *fnMatch*, *recursive*, *keepGoing*, *jobs*, *fileFn*, *resultObj=None*)

Multiprocessing code to process LIS files.

dIn, dOut are directories.

fnMatch is a glob string.

recursive is a boolean to control recursion.

keepGoing is passed to fileFn

fileFn is the operational function that will take a tuple of: (fIn, fOut, keepGoing) and return a result that can be added to the resultObj or None. This should not raise.

resultObj is accumulation of the results of fileFn or None, this it returned.

TotalDepth.LIS.RandomFrameSetRead

Created on 25 Feb 2011

@author: p2ross

`TotalDepth.LIS.RandomFrameSetRead.`**`regexStdout`**(*fp*)
Opens a file on path fp and regexes it for data.

TotalDepth.LIS.ScanLogiData

Created on 10 Nov 2010

@author: p2ross

`TotalDepth.LIS.ScanLogiData.`**`ret_int_dump_list`**(*arg_string: str*) → List[int]
Splits a string and returns a list of integers from hex/dec.

TotalDepth.LIS.ScanLogiRec

Created on 10 Nov 2010

@author: p2ross

TotalDepth.LIS.ScanPhysRec

Created on 10 Nov 2010

@author: p2ross

class `TotalDepth.LIS.ScanPhysRec.`**`PhysRecScanResult`**(*path*, *pr_count*, *error*)

property path
Alias for field number 0

property pr_count
Alias for field number 1

property error
Alias for field number 2

__str__()
Return str(self).

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *path: str*, *pr_count: int*, *error: bool*)
Create new instance of PhysRecScanResult(path, pr_count, error)

__repr__()
Return a nicely formatted representation string

class TotalDepth.LIS.ScanPhysRec.**PhysRecScanResultWithPad**(*result*, *pad_modulo*, *pad_non_null*)

property result
Alias for field number 0

property pad_modulo
Alias for field number 1

property pad_non_null
Alias for field number 2

__str__()
Return str(self).

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *result: TotalDepth.LIS.ScanPhysRec.PhysRecScanResult*, *pad_modulo: int*, *pad_non_null: bool*)
Create new instance of PhysRecScanResultWithPad(result, pad_modulo, pad_non_null)

__repr__()
Return a nicely formatted representation string

TotalDepth.LIS.TableHistogram

Provides a count of elements in LIS tables.

Created on May 24, 2011

@author: paulross

class TotalDepth.LIS.TableHistogram.**TableMatcher**
Tests if a table entry matches.

property lrType
Alias for field number 0

property nameTable
Alias for field number 1

property nameRow
Alias for field number 2

property nameCol
Alias for field number 3

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *lrType: int*, *nameTable: bytes*, *nameRow: bytes*, *nameCol: bytes*)
Create new instance of TableMatcher(lrType, nameTable, nameRow, nameCol)

__repr__()
Return a nicely formatted representation string

TotalDepth.LIS.ToLAS

Reads a LIS file and writes out LAS files.

Created on 25 Mar 2011

class TotalDepth.LIS.ToLAS.**LisLogicalFile**
Contains a representation of a LIS Logical File which is a series of indexes followed by a Log Pass

__init__()
Initialize self. See help(type(self)) for accurate signature.

add_index(*index_entry: TotalDepth.LIS.core.FileIndexer.IndexObjBase*) → None
Adds an index or Log Pass.

is_end() → bool
Can be added to.

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

TotalDepth.LIS.ToLAS.**stringify**(*value: Any*, *float_format: str*) → str
Convert an object to a string respecting the requested floating point format.

TotalDepth.LIS.ToLAS.**write_las_file**(*path_in: str, array_reduction: str, path_out: str, frame_slice: Union[TotalDepth.common.Slice.Slice, TotalDepth.common.Slice.Sample], channels: Set[str], field_width: int, float_format: str, lis_file: TotalDepth.LIS.core.File.FileRead, logical_file_index: int, lis_logical_file: TotalDepth.LIS.ToLAS.LisLogicalFile*) → int
Writes a single LAS file corresponding ot a LIS logical file (set of CONS tables and a LogPass).

TotalDepth.LIS.lis_cmn_cmd_opts

Common command line options for LIS tools.

1.9.6 TotalDepth.RP66V1 API Reference

Contents:

TotalDepth.RP66V1.core.AbsentValue

Absent Value - a value that represents a missing value.

Unlike LIS79 which specifies an absent value in the DFSR, RP66V1 has no means of explicitly specifying an absent value. Observation of real-world files shows the -999.25 is used for floats and -999 for integer representation codes.

TotalDepth.RP66V1.core.AbsentValue.**absent_value_from_rep_code**(*rep_code: int*) → Union[float, int, None]
Returns the absent value depending on the Representation Code.

TotalDepth.RP66V1.core.File

Handles low level RP66V1 operations.

TODO: Replace this with the C/C++ implementation.

exception TotalDepth.RP66V1.core.pFile.**ExceptionFile**

exception TotalDepth.RP66V1.core.pFile.**ExceptionEOF**
Premature EOF.

exception TotalDepth.RP66V1.core.pFile.**ExceptionTIF**

exception TotalDepth.RP66V1.core.pFile.**ExceptionTIFEOF**

exception TotalDepth.RP66V1.core.pFile.**ExceptionVisibleRecord**

exception TotalDepth.RP66V1.core.pFile.**ExceptionVisibleRecordEOF**

exception TotalDepth.RP66V1.core.pFile.**ExceptionLogicalRecordSegmentHeaderAttributes**

exception TotalDepth.RP66V1.core.pFile.**ExceptionLogicalRecordSegmentHeader**

exception TotalDepth.RP66V1.core.pFile.**ExceptionLogicalRecordSegmentHeaderEOF**

exception TotalDepth.RP66V1.core.pFile.**ExceptionLogicalRecordSegmentHeaderSequence**

exception TotalDepth.RP66V1.core.pFile.**ExceptionFileRead**

exception TotalDepth.RP66V1.core.pFile.**ExceptionFileReadEOF**

exception TotalDepth.RP66V1.core.pFile.**ExceptionFileReadPositionsInconsistent**

exception TotalDepth.RP66V1.core.pFile.**ExceptionStorageUnitLabel**
Exception specialisation for this module.

TotalDepth.RP66V1.core.pFile.**TIF_FILE_PREFIX = b'\x00\x00\x00\x00\x00\x00\x00\x00x5c\x00\x0**
The first one must be StorageUnitLabel.SIZE + TIF_SIZE = 92 or 0x5c little endian 0000 0000 0000 0000 5c00 0000 This is here just to raise and exception for TIF marked RP66V1 files.

class TotalDepth.RP66V1.core.pFile.**StorageUnitLabel**(*by: bytes*)
The Storage Unit Label that must be at the beginning of a file. See [RP66V1 Section 2.3.2 Storage Unit Label (SUL)].

Unique Storage Unit Labels seen in practice:

```
$ grep -rIh StorageUnitLabel <XML indexes> | sort | uniq -c
  16   <StorageUnitLabel dlis_version="V1.00" maximum_record_length="16384"␣
↪sequence_number="1"
       storage_set_identifier="CUSTOMER                                    ␣
↪          "
       storage_unit_structure="RECORD"/>
   2   <StorageUnitLabel dlis_version="V1.00" maximum_record_length="16384"␣
↪sequence_number="1"
       storage_set_identifier="PRODUCER                                    ␣
↪          "
       storage_unit_structure="RECORD"/>
   1   <StorageUnitLabel dlis_version="V1.00" maximum_record_length="8192"␣
↪sequence_number="1"
       storage_set_identifier="                                            ␣
↪          "
       storage_unit_structure="RECORD"/>
  34   <StorageUnitLabel dlis_version="V1.00" maximum_record_length="8192"␣
↪sequence_number="1"
```

(continues on next page)

(continued from previous page)

```
        storage_set_identifier="DLIS ATLAS 1
↪           "
        storage_unit_structure="RECORD"/>
 537   <StorageUnitLabel dlis_version="V1.00" maximum_record_length="8192"
↪sequence_number="1"
        storage_set_identifier="Default Storage Set
↪           "
        storage_unit_structure="RECORD"/>
```

__init__ (*by: bytes*)
Initialize self. See help(type(self)) for accurate signature.

as_bytes () → bytes
Returns the bytes that encode this Storage Unit Label.

__str__ () → str
Return str(self).

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

TotalDepth.RP66V1.core.pFile.**create_storage_unit_label** (*storage_unit_sequence_number: int, dlis_version: bytes, maximum_record_length: int, storage_set_identifier: bytes*) → TotalDepth.RP66V1.core.pFile.StorageUnitLabel
Create a StorageUnitLabel from the given values.

class TotalDepth.RP66V1.core.pFile.**VisibleRecord** (*fobj: Optional[BinaryIO]*)
RP66V1 visible records. See [RP66V1 Section 2.3.6] (sic) - Place marker for error in the standard in this case

__init__ (*fobj: Optional[BinaryIO]*)
Initialize self. See help(type(self)) for accurate signature.

as_bytes () → bytes
The Visible Record represented in raw bytes.

read (*fobj: BinaryIO*) → None
Read a new Visible Record and check it. This may throw a ExceptionVisibleRecord.

read_next (*fobj: BinaryIO*) → None
Move to next Visible Record and read it. This may throw a ExceptionVisibleRecord.

__format__ (*format_spec*) → str
Default object formatter.

__eq__ (*other*)
Return self==value.

__str__ () → str
Return str(self).

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

class TotalDepth.RP66V1.core.pFile.**LogicalRecordSegmentHeader** (*fobj: BinaryIO*)
RP66V1 Logical Record Segment Header. See See [RP66V1 2.2.2.1]

__init__ (*fobj: BinaryIO*)
Constructor. position: The file position of the start of the LRSH.

length: The *Logical Record Segment Length* is a two-byte, unsigned integer (Representation Code UNORM) that specifies the length, in bytes, of the Logical Record Segment. The Logical Record Segment Length is required to be even. The even length ensures that 2-byte checksums can be computed, when present, and permits some operating systems to handle DLIS data more efficiently without degrading performance with other systems. There is no limitation on a Logical Record length. Logical Record Segments must contain at least sixteen (16) bytes. This requirement facilitates mapping the Logical Format to those Physical Formats that require a minimum physical record length.

attributes: The *Logical Record Segment Attributes* consist of a one-byte bit string that specifies the Attributes of the Logical Record Segment. Its structure is defined in Figure 2-3. Since its structure is defined explicitly in Figure 2-3, no Representation Code is assigned to it.

record_type: The *Logical Record Type* is a one-byte, unsigned integer (Representation Code USHORT) that specifies the Type of the Logical Record. Its value indicates the general semantic content of the Logical Record. The same value must be used in all Segments of a Logical Record. Logical Record Types are specified in Appendix A.

IFLRs: Numeric codes 0-127 are reserved for Public IFLRs. Codes 128-255 are reserved for Private IFLRs. 0 is Frame Data, 1 is unformatted data.

EFLRs: Numeric codes 0-127 are reserved for Public EFLRs. Codes 128-255 are reserved for Private EFLRs. 0 is FILE-HEADER, 1 is ORIGIN and so on.

read(*fobj: BinaryIO*) → None
Read a new Logical Record Segment Header. This may throw a ExceptionVisibleRecord or ExceptionLogicalRecordSegmentHeaderEOF.

as_bytes() → bytes
The LRSH represented in raw bytes.

__str__() → str
Return str(self).

property next_position
File position of the start of the next Logical Record Segment Header.

property logical_data_position
File position of the start of the Logical Data.

property logical_data_length
Returns the length of the logical data, including padding but excluding the tail.

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

class TotalDepth.RP66V1.core.pFile.**LogicalRecordPosition**(*vr: TotalDepth.RP66V1.core.pFile.VisibleRecord*, *lrsh: TotalDepth.RP66V1.core.pFile.LogicalRecordSegme*
Class that contains the file position of the Logical Record Segment Header and the immediately prior Visible Record.

__init__(*vr: TotalDepth.RP66V1.core.pFile.VisibleRecord*, *lrsh: TotalDepth.RP66V1.core.pFile.LogicalRecordSegmentHeader*)
Initialize self. See help(type(self)) for accurate signature.

__str__()
Return str(self).

__eq__(*other*)
Return self==value.

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

class TotalDepth.RP66V1.core.pFile.**LogicalDataDescription**
At this level this describes the raw Logical Data that can be converted into a Logical Record.

property attributes
Alias for field number 0

property lr_type
Alias for field number 1

property ld_length
Alias for field number 2

__str__() → str
Return str(self).

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *attributes: TotalDepth.RP66V1.core.pFile.LogicalRecordSegmentHeaderAttributes*, *lr_type: int*, *ld_length: int*)
Create new instance of LogicalDataDescription(attributes, lr_type, ld_length)

__repr__()
Return a nicely formatted representation string

class TotalDepth.RP66V1.core.pFile.**LRPosDesc**
This contains the position and description of a Logical Record suitable for an indexer.

It contains:

- **LogicalRecordPosition: This is the absolute file position of the Visible Record and LRSH.** This will be of interest to indexers that mean to `use get_file_logical_data()` as this is a required argument.
- **LogicalDataDescription: This provides some basic information about the Logical Data such as attributes** Logical Record type and the Logical Data length. This will be of interest to indexers to offer up to their callers.

property position
Alias for field number 0

property description
Alias for field number 1

__str__() → str
Return str(self).

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *position: TotalDepth.RP66V1.core.pFile.LogicalRecordPosition*, *description: TotalDepth.RP66V1.core.pFile.LogicalDataDescription*)
Create new instance of LRPosDesc(position, description)

__repr__()
Return a nicely formatted representation string

class TotalDepth.RP66V1.core.pFile.**LogicalData**(*by: bytes*)
Class that holds data bytes and can successively read them maintaining an index of what has been read.

__init__(*by: bytes*)
Initialize self. See help(type(self)) for accurate signature.

peek() → int
Return the next bytes without incrementing the index. May raise an IndexError if there is no data left.

read() → int
Return the next byte and increment the index. May raise an IndexError if there is no data left.

seek(*length: int*) → None
Increments the index. There is no error checking.

view_remaining(*length: int*) → bytes
Read only method to return a slice of length from the current index. Usage `ld.view_remaining(ld.remain)` to see all the remaining data.

chunk(*length: int*) → bytes
Return the next length bytes and increment the index. May raise an IndexError if there is not enough data.

property remain
The number of bytes remaining.

property sha1
Lazy SHA1 evaluation of the complete binary data.

rewind() → None
Reset the index to 0.

__bool__()
True if there is some data remaining.

__len__()
Total length of the binary data.

__getitem__(*index*)
Return a byte and the given index.

__str__() → str
String representation.

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

class TotalDepth.RP66V1.core.pFile.**FileLogicalData**(*vr: TotalDepth.RP66V1.core.pFile.VisibleRecord*, *lrsh: TotalDepth.RP66V1.core.pFile.LogicalRecordSegmentHeade*
Class that contains information about a Logical Record within a physical file. This is lazily evaluated with only the VisibleRecord and LogicalRecordSegmentHeader provided to the constructor. Eager evaluation is done with one or more add()'s followed by a seal().

__init__(*vr: TotalDepth.RP66V1.core.pFile.VisibleRecord*, *lrsh: TotalDepth.RP66V1.core.pFile.LogicalRecordSegmentHeader*)
Initialize self. See help(type(self)) for accurate signature.

add_bytes(*by: bytes*) → None
Add some raw data that is part of aa Logical Record.

seal() → None
All of the Logical Record has been read into this class so seal it to prevent any more data being added. This also creates a LogicalData object that encapsulates the logical data.

is_sealed() → bool
Returns True if this is sealed so no more bytes can be added.

__len__() → int
Number of bytes of data whether sealed or unsealed.

__str__() → str
Return str(self).

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

class TotalDepth.RP66V1.core.pFile.**FileRead**(*path_or_file: Union[str, BinaryIO]*)
RP66V1 file reader.

__init__(*path_or_file: Union[str, BinaryIO]*)
Initialize self. See help(type(self)) for accurate signature.

iter_visible_records() → Sequence[TotalDepth.RP66V1.core.pFile.VisibleRecord]
Iterate across the file yielding the Visible Records as VisibleRecord objects. The iteration cann be further divided by calling iter_LRSHs_for_VR()

iter_LRSHs_for_visible_record(*vr_given: TotalDepth.RP66V1.core.pFile.VisibleRecord*) → Sequence[TotalDepth.RP66V1.core.pFile.LogicalRecordSegmentHeader]
Iterate across the Visible Record yielding the Logical Record Segments as LogicalRecordSegmentHeader objects. This leaves the file positioned at the next Visible Record or EOF.

iter_LRSHs_for_visible_record_and_logical_data_fragment(*vr_given: TotalDepth.RP66V1.core.pFile.VisibleRecord*) → Sequence[Tuple[TotalDepth.RP66V1.core.pFile bytes]]
Iterate across the Visible Record yielding the Logical Record Segments and the Logical Data fragment as (LogicalRecordSegmentHeader, bytes) objects. This leaves the file positioned at the next Visible Record or EOF. TODO: Drop this from pFile/cFile as it is only used in one rare case in Scan.py?

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

iter_logical_records() → Sequence[TotalDepth.RP66V1.core.pFile.FileLogicalData]
Iterate across the file from the beginning yielding FileLogicalData objects.

iter_logical_record_positions() → Sequence[TotalDepth.RP66V1.core.pFile.LRPosDesc]
Iterate across the file from the beginning yielding a LRPosDesc which contains:

- **LogicalRecordPosition: This is the absolute file position of the Visible Record and LRSH.** This will be of interest to indexers that mean to `use get_file_logical_data()` as this is a required argument.
- **LogicalDataDescription: This provides some basic information about the Logical Data such as attributes** Logical Record type and the Logical Data length. This will be of interest to indexers to offer up to their callers.

get_file_logical_data(*position: TotalDepth.RP66V1.core.pFile.LogicalRecordPosition*, *offset: int = 0*, *length: int = -1*) → TotalDepth.RP66V1.core.pFile.FileLogicalData
Returns a FileLogicalData object from the Logic Record position (Visible Record Position and Logical Record Segment Header position). This allows random access to the file to an index that has the Logical Record Positions. This will leave the file at EOFF or at the beginning of the next Visible Record or LRSH.

Param position A LogicalRecordPosition that specifies the visible record and LRSH position of the first LRSH for the Logical Record data.

Param offset An integer offset into the Logical Record data, default 0.

Param length An integer length the required Logical Record data, default of -1 is all.

validate_positions() → None
Iterate through the Visible Records and Logical Record Segment Headers and raise a ExceptionFileReadPositionsInconsistent on the first inconsistent position.

TotalDepth.RP66V1.core.Index

RP66V1 file indexer at the level of Visible Records and the first Logical Record Segment Header.

This allows random access of Logical Record data.

In the taxonomy of indexes this is a 'mid level' index as it indexes:

- Below: Internally it discovers and records Visible Records and Logical Record Segment Headers (where `is_first()` is `True`).
- Above: Externally it provides an API to a sequence of Logical Records and the data that makes up those Logical Records.

TODO: Replace this with the C/C++ implementation.

exception TotalDepth.RP66V1.core.pIndex.**ExceptionIndex**
Base class for exceptions in this module.

class TotalDepth.RP66V1.core.pIndex.**LogicalRecordIndex**(*path_or_file: Union[str, _io.BytesIO]*)
This maintains an index of visible record and Logical Record Segment Header (LRSH) positions where the LRSH is the first in the Logical Record.

The index is a list of File.LRPosDesc objects that contain:

- **.position A LogicalRecordPosition which has the absolute file position of the Visible Record and LRSH.** This will be of interest to indexers that mean to `use get_file_logical_data()` as this is a required argument.
- **.description A LogicalDataDescription which provides some basic information about the Logical Data such as** the LRSH attributes, Logical Record type and the Logical Data length. This will be of interest to indexers to offer up to their callers.

__init__(*path_or_file: Union[str, _io.BytesIO]*)
Initialize self. See help(type(self)) for accurate signature.

property sul
The file's Storage Unit Label.

property visible_record_positions
A list of Visible Record positions. This is used by the XML index for example.

get_file_logical_data(*index: int*, *offset: int = 0*, *length: int = -1*) → TotalDepth.RP66V1.core.pFile.FileLogicalData
Returns a FileLogicalData object from the Logical Record position in the index (Visible Record Position and Logical Record Segment Header position).

This allows random access to any Logical Record in the file. The caller can construct a more sophisticated index such as a sequence of Logical Files which contain Logical Records that can be EFLRs or IFLRs and interpreted accordingly.

If offset or length are use then the result will be the partial data from that offset and length.

Param index The index of the Logical Record.

Param offset An integer offset into the Logical Record data, default 0.

Param length An integer length the Logical Record data, default of -1 is all.

get_file_logical_data_at_position(*position: TotalDepth.RP66V1.core.pFile.LogicalRecordPosition*, *offset: int = 0*, *length: int = -1*) → TotalDepth.RP66V1.core.pFile.FileLogicalData

Returns a FileLogicalData object from the Logical Record position.

This allows random access to any Logical Record in the file. The caller can construct a more sophisticated index such as a sequence of Logical Files which contain Logical Records that can be EFLRs or IFLRs and interpreted accordingly.

If offset or length are use then the result will be the partial data from that offset and length.

Param position The Logical Record position in the file.

Param offset An integer offset into the Logical Record data, default 0.

Param length An integer length the Logical Record data, default of -1 is all.

validate()

Perform validation checks.

__weakref__

list of weak references to the object (if defined)

TotalDepth.RP66V1.core.LogPass

This provides a representation of the structure of recorded data.

exception TotalDepth.RP66V1.core.LogPass.**ExceptionLogPass**

exception TotalDepth.RP66V1.core.LogPass.**ExceptionFrameChannel**

exception TotalDepth.RP66V1.core.LogPass.**ExceptionFrameArray**

exception TotalDepth.RP66V1.core.LogPass.**ExceptionFrameArrayInit**

exception TotalDepth.RP66V1.core.LogPass.**ExceptionLogPassInit**

exception TotalDepth.RP66V1.core.LogPass.**ExceptionLogPassProcessIFLR**

class TotalDepth.RP66V1.core.LogPass.**RP66V1FrameChannel**(*ident: Hashable*, *long_name: bytes*, *units: bytes*, *dimensions: List[int]*, *np_dtype: numpy.dtype*, *rep_code: int*)

This represents a single channel in a frame. It is file format independent and can be used depending on the source of the information: LIS/LAS/RP66V1 file, XML index, Postgres database etc.

__init__(*ident: Hashable*, *long_name: bytes*, *units: bytes*, *dimensions: List[int]*, *np_dtype: numpy.dtype*, *rep_code: int*)

Constructor.

Parameters

- **ident** – Some hashable identity.
- **long_name** – A description of the channel
- **units** – Units of Measure.

- **dimensions** – A list of dimensions of each value. [1] is a single value per frame. [4, 1024] is a 4 * 1024 matrix such as sonic waveform of 1024 samples with 4 waveforms per frame.
- **np_dtype** – The numpy dtype to use.
- **rep_code** – Integer representation code that this channel is encoded in.

property ident
Overload base class.

property len_input_bytes
The number of RP66V1 bytes to read for one frame of this channel.

read(*ld: TotalDepth.RP66V1.core.pFile.LogicalData*, *frame_number: int*) → None
Reads the Logical Data into the numpy frame at the specified frame number.

This is currently RP66V1 specific. In future designs this can be sub-classed by format (LAS, LIS, RP66V1 etc.)

seek(*ld: TotalDepth.RP66V1.core.pFile.LogicalData*) → None
Increments the logical data without reading any values into the array.

class TotalDepth.RP66V1.core.LogPass.**RP66V1FrameArray**(*ident: Hashable, description: Union[str, bytes]*)
In the olden days we would record this on a single chunk of continuous film.

read(*ld: TotalDepth.RP66V1.core.pFile.LogicalData*, *frame_number: int*) → None
Reads the Logical Data into the numpy frame.

read_partial(*ld: TotalDepth.RP66V1.core.pFile.LogicalData, frame_number: int, channels: Set[Hashable]*) → None
Reads the Logical Data into the numpy frame for the nominated channels.

property x_axis_len_input_bytes
The number of RP66V1 bytes to read for one frame of the X axis. This can be useful for partial reads of an IFLR from file if only the X axis is interesting, for example for indexing.

Will raise an ExceptionFrameChannel if the X axis is not represented by a fixed length Representation Code.

init_x_axis_array(*number_of_frames: int*) → None
Initialises an empty Numpy array for the first channel suitable to fill with <frames> number of frame data.

read_x_axis(*ld: TotalDepth.RP66V1.core.pFile.LogicalData*, *frame_number: int*) → None
Reads the first channel of the Logical Data into the numpy frame.

TotalDepth.RP66V1.core.LogPass.**frame_channel_from_RP66V1**(*channel_object: TotalDepth.RP66V1.core.LogicalRecord.EFLR.Obje* → TotalDepth.RP66V1.core.LogPass.RP66V1FrameCh
Create a file format agnostic FrameChannel from an EFLR.Object (row) in a *CHANNEL* EFLR.

TotalDepth.RP66V1.core.LogPass.**frame_array_from_RP66V1**(*frame_object: TotalDepth.RP66V1.core.LogicalRecord.EFLR.Object, channel_eflr: TotalDepth.RP66V1.core.LogicalRecord.EFLR.Explicit* → TotalDepth.RP66V1.core.LogPass.RP66V1FrameArray
Create a file format agnostic FrameArray from an EFLR.Object (row) in a *FRAME* EFLR and a *CHANNEL* EFLR.

TotalDepth.RP66V1.core.LogPass.**log_pass_from_RP66V1**(*frame: TotalDepth.RP66V1.core.LogicalRecord.EFLR.ExplicitlyFo*
channels: TotalDepth.RP66V1.core.LogicalRecord.EFLR.ExplicitlyFo
→ TotalDepth.common.LogPass.LogPass

Create a file format agnostic FrameArray from a *FRAME* type EFLR and a *CHANNEL* type EFLR.

TotalDepth.RP66V1.core.LogicalFile

Represents a RP66V1 file as a 'logical' level.

exception TotalDepth.RP66V1.core.LogicalFile.**ExceptionLogicalFile**

exception TotalDepth.RP66V1.core.LogicalFile.**ExceptionLogicalFileCtor**

exception TotalDepth.RP66V1.core.LogicalFile.**ExceptionLogicalFileAdd**

exception TotalDepth.RP66V1.core.LogicalFile.**ExceptionLogicalFileMissingData**

exception TotalDepth.RP66V1.core.LogicalFile.**ExceptionLogicalIndex**

exception TotalDepth.RP66V1.core.LogicalFile.**ExceptionLogicalIndexCtor**

class TotalDepth.RP66V1.core.LogicalFile.**PositionEFLR**

POD class that represents the Logical Record Segment Header position in the file of the Explicitly Formatted Logical Record and the EFLR itself.

property lrsh_position

Alias for field number 0

property eflr

Alias for field number 1

__getnewargs__()

Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *lrsh_position: TotalDepth.RP66V1.core.pFile.LogicalRecordPosition*, *eflr: TotalDepth.RP66V1.core.LogicalRecord.EFLR.ExplicitlyFormattedLogicalRecord*)

Create new instance of PositionEFLR(lrsh_position, eflr)

__repr__()

Return a nicely formatted representation string

class TotalDepth.RP66V1.core.LogicalFile.**LogicalFile**(*logical_record_index: TotalDepth.RP66V1.core.pIndex.LogicalRecordIndex*, *file_logical_data: TotalDepth.RP66V1.core.pFile.FileLogicalData*, *fhlr: TotalDepth.RP66V1.core.LogicalRecord.EFLR.ExplicitlyF*

This represents a RP66V1 Logical File.

From the standard [RP66V1 Definitions]:

Logical File A sequence of two or more contiguous Logical Records in a Storage Set that begins with a File Header Logical Record and contains no other File Header Logical Records. A Logical File must have at least one OLR (Origin) Logical Record immediately following the File Header Logical Record. A Logical File supports user-level organization of data.

For the File Header Logical Record see [RP66V1 Section 5.1 File Header Logical Record (FHLR)] For the Origin Logical Record see [RP66V1 Section 5.2 Origin Logical Record (OLR)]

This is actually two/multi stage construction with the FHLR first. The OLR is extracted from the first add().

TODO: Check the FHLR references the ORIGIN record. [RP66V1 Section 5.1 File Header Logical Record (FHLR)]: "The Origin Subfield of the Name of the File-Header Object must reference the Defining Origin (see §5.2.1)." Is this done in practice?

TODO: Handle multiple ORIGIN records. This is actually allowed in the standard: "A Logical File must have at least one OLR (Origin) Logical Record immediately following the File Header Logical Record".

TODO: Handle multiple CHANNEL records in a Logical File.

classmethod DUPE_EFLR_CHANNEL_LOGGER(*msg*, **args*, ***kwargs*)
Log 'msg % args' with severity 'WARNING'.

To pass exception information, use the keyword argument exc_info with a true value, e.g.

logger.warning("Houston, we have a %s", "bit of a problem", exc_info=1)

__init__(*logical_record_index: TotalDepth.RP66V1.core.pIndex.LogicalRecordIndex*, *file_logical_data: TotalDepth.RP66V1.core.pFile.FileLogicalData*, *fhlr: TotalDepth.RP66V1.core.LogicalRecord.EFLR.ExplicitlyFormattedLogicalRecord*)
Initialize self. See help(type(self)) for accurate signature.

property file_header_logical_record
Returns the FILE-HEADER EFLR see [RP66V1 Section 5.1 File Header Logical Record (FHLR)].

property origin_logical_record
Returns the ORIGIN EFLR see [RP66V1 Section 5.2 Origin Logical Record (OLR)].

property defining_origin
Returns the Defining Origin of the Logical File. This is the first row of the ORIGIN Logical Record. From [RP66V1 Section 5.2.1 Origin Objects]: "The first Object in the first ORIGIN Set is the Defining Origin for the Logical File in which it is contained, and the corresponding Logical File is called the Origin's Parent File. It is intended that no two Logical Files will ever have Defining Origins with all Attribute Values identical."

property has_log_pass
Return True if a log pass has been created from a CHANNEL and a FRAME record and some relevant IFLRs.

static is_next(*eflr: TotalDepth.RP66V1.core.LogicalRecord.EFLR.ExplicitlyFormattedLogicalRecord*) → bool
Returns True if the given EFLR belongs to the next Logical Record.

add_eflr(*file_logical_data: TotalDepth.RP66V1.core.pFile.FileLogicalData*, *eflr: TotalDepth.RP66V1.core.LogicalRecord.EFLR.ExplicitlyFormattedLogicalRecord*) → None
Adds an EFLR in sequence from the file.

Will raise a `ExceptionLogicalFileAdd` if the EFLR is a `FILE-HEADER` as that signals the next Logical File.

add_iflr(*file_logical_data: TotalDepth.RP66V1.core.pFile.FileLogicalData*, *iflr: TotalDepth.RP66V1.core.LogicalRecord.IFLR.IndirectlyFormattedLogicalRecord*) → None
Adds a IFLR entry to the index. The IFLR just contains the object name and frame number. This extracts the X axis from the first value in the IFLR free data and appends this to the iflr_position_map.

num_frames(*frame_array: TotalDepth.RP66V1.core.LogPass.RP66V1FrameArray*) → int
Return the number of frames in the FrameArray

populate_frame_array(*frame_array: TotalDepth.RP66V1.core.LogPass.RP66V1FrameArray*, *frame_slice: Union[TotalDepth.common.Slice.Slice, TotalDepth.common.Slice.Sample, None] = None*, *channels: Optional[Set[Hashable]] = None*) → int

Populates a FrameArray with channel values.

frame_array must be a member of the LogPass in thisLogicalFile.

frame_slice Allows partial population in the X axis.

channels Allows partial population of specific channels.

The FrameArray will be populated and this returns the number of frames populated.

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

class TotalDepth.RP66V1.core.LogicalFile.**LogicalIndex**(*path_or_file: Union[str, BinaryIO]*)

This takes a RP66V1 file and indexes it into a sequence of Logical Files.

__init__(*path_or_file: Union[str, BinaryIO]*)
Initialize self. See help(type(self)) for accurate signature.

__len__() → int
Returns the number of Logical Files.

__getitem__(*item*) → TotalDepth.RP66V1.core.LogicalFile.LogicalFile
Returns the Logical Files at position item.

property storage_unit_label
The Storage Unit Label. This comes from the LogicalRecordIndex.

property visible_record_positions
A list of Visible Record positions. This is used by the XML index for example.

__enter__()
Context manager support.

__exit__(*exc_type*, *exc_val*, *exc_tb*)
Context manager support.

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

TotalDepth.RP66V1.core.RepCode

TotalDepth.RP66V1.core.Units

Handles RP66V1 units.

There are two approaches:

Standard Based Parser

Formal but slow and does not appreciate that there are a load of non-standard implementations out there.

References: [RP66V1 Appendix b, B.27 Code UNITS: Units Expression] ["Energistics Unit Symbol Grammar Specification" Section 2.2 Unit Symbol Construction Grammar]

```
UnitSymbol ::= [ Multiplier ' ' ] FactorExpression .

FactorExpression ::= OneOrMore |
    ( '1' | OneOrMore | Division ) '/' (
```

(continues on next page)

(continued from previous page)

```
Division ::= '(' OneOrMore '/' Divisor ')'

OneOrMore ::= Factor | Factors .

Divisor ::= Factor | '(' Factors ')' .

Factors ::= Factor '.' Factor { '.' Factor

Factor ::= UnitComponent [ Exponent ] .

UnitComponent ::= PrefixedAtom | Atom | SpecialAtom [ Qualifier ] .

PrefixedAtom ::= ( SIPrefix | BinaryPrefix ) Atom .

Atom ::= Letter { Letter } [ Qualifier ] .

SpecialAtom ::= '%' | 'inH2O' | 'cmH2O' .

Qualifier ::= '[' [ AT ] QualPart { COMMA QualPart } ']' .

AT ::= '@' .

COMMA ::= ',' .

QualPart ::= LetterOrDigit { LetterOrDigit

LetterOrDigit ::= Letter | Digit .

Letter ::= E | LTTR .

LTTR ::= 'A'|'B'|'C' | 'D' | 'L' | 'M' | 'N' | 'O' | 'W' | 'X' | 'Y' | 'Z' | 'a' | 'b
↪' | 'c' | 'd' | 'l' | 'm' | 'n'
    | 'o' | 'w' | 'x' | 'y' | 'z' . 'P' | 'e' | 'p' | } .
    } .
    'F'|'G'|'H' 'Q' | 'R' | 'S'
    'f' | 'g' | 'h'
    'q' | 'r' | 's'
    | 'I' | 'J' | 'K' |
    | 'T' | 'U' | 'V' |
    | 'i' | 'j' | 'k' |
    | 't' | 'u' | 'v' |

Exponent ::= GtOneDigit | '(' ( NonZeroInt '.' FractionalPart | '0' '.'␣
↪FractionalPart ) ')' .

Multiplier ::= '1' E PowerOfTen [ '/' GtOneInt ] | '1' '/' GtOneInt |
    Number [ E PowerOfTen ][ '/' GtOneInt ] .

E ::= 'E' .

PowerOfTen ::= [ '-' ] GtOneInt .

Number ::= GtOneInt |
           NonZeroInt '.' FractionalPart | '0' '.' FractionalPart .

GtOneInt ::= GtOneDigit | NonZeroDigit Digit { Digit } .
```

(continues on next page)

(continued from previous page)

```
FractionalPart ::= { Digit } NonZeroDigit .

NonZeroInt ::= NonZeroDigit { Digit } .

Digit ::= '0' | NonZeroDigit .

GtOneDigit ::= '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' .

NonZeroDigit ::= '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' .

SIPrefix ::= 'y' | 'z' | 'a' | 'f' | 'p' | 'n' | 'u' | 'm' | 'c' | 'd' |
'da' | 'h' | 'k' | 'M' | 'G' | 'T' | 'P' | 'E' | 'Z' | 'Y' .

BinaryPrefix ::= 'Ki' | 'Mi' | 'Gi' | 'Ti' | 'Pi' | 'Ei' | 'Zi' | 'Yi' .
```

Something like this:

```
import re
import typing


# Token = collections.namedtuple('Token', ['type', 'value', 'line', 'column'])
class Token(typing.NamedTuple):
    type: str
    value: str
    line: int
    column:int


def tokenise_units(code: str) -> typing.Sequence[Token]:
    # See also [RP66V1 Appendix B, B.27 Code UNITS: Units Expression]
    token_specification = [
        ('BinaryPrefix',   r'Ki|Mi|Gi|Ti|Pi|Ei|Zi|Yi'),
        ('SiPrefix',   r'y|z|a|f|p|n|u|m|c|d|da|h|k|M|G|T|P|E|Z|Y'),
        ('NonZeroDigit', r'1|2|3|4|5|6|7|8|9'),
        ('Digit', r'0|1|2|3|4|5|6|7|8|9'),
        ('GtOneDigit', r'2|3|4|5|6|7|8|9'),
        ('E', r'E'),
        ('Letter', r'[A-Za-z]'),
        ('COMMA', r','),
        ('AT', r'\@'),
        ('SpecialAtom', r'%|inH2O|cmH2O'),
        ('PARENLEFT', r'\('),
        ('PARENRIGHT', r'\)'),
        ('MULTIPLY', r'\*'),
        ('DIVIDE', r'/'),
        ('BLANK', r' '),
        ('DOT', r'.'),
        ('HYPHEN', r'-'),
        ('NEWLINE',  r'\n'),
        ('ID',       r'[A-Za-z]+'),
        ('MISSMATCH', r'.'),
    ]
    tok_regex = '|'.join('(?P<%s>%s)' % pair for pair in token_specification)
    line_num = 1
    line_start = 0
```

(continues on next page)

(continued from previous page)

```
    for mo in re.finditer(tok_regex, code):
        kind = mo.lastgroup
        value = mo.group()
        column = mo.start() - line_start
        if kind == 'NUMBER':
            value = float(value) if '.' in value else int(value)
        # elif kind == 'ID' and value in keywords:
        #     kind = value
        # elif kind == 'NEWLINE':
        #     line_start = mo.end()
        #     line_num += 1
        #     continue
        # elif kind == 'SKIP':
        #     continue
        elif kind == 'MISMATCH':
            raise RuntimeError(f'{value!r} unexpected on line {line_num}')
        yield Token(kind, value, line_num, column)

for token in tokenise_units('0.1m/s2'):
for token in tokenise_units('627264E5/15499969 m2'):
    print(token)
```

This is likely to be slow, some pre-processing may help.

Lookup

This uses online or offline data structures. The primary source is Schlumberger's Oilfield Services Data Dictionary (OSDD): https://www.apps.slb.com/cmd/units.aspx It is quick and largely respects existing implementations.

Other data providers (by PRODUCER-CODE) may have alternate mappings that can be put into PRODUCER_CODE_MAPPING_OF_UNIT_CODE.

See src/TotalDepth/RP66V1/util/XMLReadUnits.py for some analysis.

exception TotalDepth.RP66V1.core.Units.**ExceptionRP66V1Units**
Base class exception for this module.

TotalDepth.RP66V1.core.Units.**PRODUCER_CODE_MAPPING_OF_UNIT_CODE = {280: {b'gapi': b'GAPI**
This permits different producer codes to map into Schlumberger's Oilfield Services Data Dictionary (OSDD). This is just an example, see *src/TotalDepth/RP66V1/util/XMLReadUnits.py* for some analysis of test files.

TotalDepth.RP66V1.core.Units.**convert**(*value: float*, *unit_from: bytes*, *unit_to: bytes*, *producer_code: int = 0*) → float
Converts a value from one unit to another. This uses TotalDepth.common.units with an additional producer code mapping.

Examples:

```
Code    Name                Standard Form   Dimension   Scale               Offset
DEGC    'degree celsius'    degC            Temperature 1                   -273.
↪15
DEGF    'degree fahrenheit' degF            Temperature 0.555555555555556   -459.
↪67
```

So conversion from, say DEGC to DEGF is:

```
((value - DEGC.offset) * DEGC.scale) / DEGF.scale + DEGF.offset

((0.0 - -273.15) * 1.0) / 0.555555555555556 + -459.67 == 32.0
```

TotalDepth.RP66V1.core.Units.**convert_function**(*unit_from: TotalDepth.common.units.Unit*, *unit_to: TotalDepth.common.units.Unit*, *producer_code: int = 0*) → Callable

Return a partial function to convert from one RP66V1 units to another.

TotalDepth.RP66V1.core.XAxis

Provides analysis and navigation along the X axis of RP66V1 logs.

class TotalDepth.RP66V1.core.XAxis.**XAxisSpacingCounts**(*norm*, *dupe*, *skip*, *back*)

property norm
Alias for field number 0

property dupe
Alias for field number 1

property skip
Alias for field number 2

property back
Alias for field number 3

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *norm: int*, *dupe: int*, *skip: int*, *back: int*)
Create new instance of XAxisSpacingCounts(norm, dupe, skip, back)

__repr__()
Return a nicely formatted representation string

class TotalDepth.RP66V1.core.XAxis.**XAxisSpacingSummary**(*min*, *max*, *mean*, *median*, *std*, *counts*, *histogram*)

property min
Alias for field number 0

property max
Alias for field number 1

property mean
Alias for field number 2

property median
Alias for field number 3

property std
Alias for field number 4

property counts
Alias for field number 5

property histogram
Alias for field number 6

__eq__(*other*)
Return self==value.

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls, min: float, max: float, mean: float, median: float, std: float, counts: TotalDepth.RP66V1.core.XAxis.XAxisSpacingCounts, histogram: Tuple[numpy.ndarray, numpy.ndarray]*)
Create new instance of XAxisSpacingSummary(min, max, mean, median, std, counts, histogram)

__repr__()
Return a nicely formatted representation string

TotalDepth.RP66V1.core.XAxis.**compute_spacing**(*x_array: numpy.ndarray*) → Optional[TotalDepth.RP66V1.core.XAxis.XAxisSpacingSummary]
Given an array this computes the summary of the first differential of the array or None if there are less than two values in the array.

Given a median of the first differential, median, a subsequent differential, dx, is considered: 'backward' if dx < -0.5 median 'duplicate' if -0.5 median <= dx < 0.5 median 'normal' if 0.5 median <= dx < 1.5 median 'skipped' if dx >= 1.5 median

class TotalDepth.RP66V1.core.XAxis.**XAxisSummary**(*min*, *max*, *count*, *spacing*)

property min
Alias for field number 0

property max
Alias for field number 1

property count
Alias for field number 2

property spacing
Alias for field number 3

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls, min: float, max: float, count: int, spacing: Optional[TotalDepth.RP66V1.core.XAxis.XAxisSpacingSummary]*)
Create new instance of XAxisSummary(min, max, count, spacing)

__repr__()
Return a nicely formatted representation string

class TotalDepth.RP66V1.core.XAxis.**IFLRReference**
POD class that represents the position of the IFLR in the file.

property logical_record_position
Alias for field number 0

property frame_number
Alias for field number 1

property x_axis
Alias for field number 2

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls, logical_record_position: TotalDepth.RP66V1.core.pFile.LogicalRecordPosition, frame_number: int, x_axis: Union[int, float]*)
Create new instance of IFLRReference(logical_record_position, frame_number, x_axis)

__repr__()
Return a nicely formatted representation string

class TotalDepth.RP66V1.core.XAxis.**XAxis**(*ident: bytes*, *long_name: bytes*, *units: bytes*)
This represents an X axis of a log pass for a particular object in that log pass. It has an ident, long name and units. It accumulates, for every IFLR in the set, the VR position LRSH position, frame number and X axis value.

__init__(*ident: bytes*, *long_name: bytes*, *units: bytes*)
Initialize self. See help(type(self)) for accurate signature.

append(*position: TotalDepth.RP66V1.core.pFile.LogicalRecordPosition, frame_number: int, x_axis: Union[int, float]*) → None
Add a IFLRReference to the XAxis.

__getitem__(*item*) → TotalDepth.RP66V1.core.XAxis.IFLRReference
Return the IFLRReference for the index.

__len__() → int
Return the number of IFLRs.

property summary
Lazily compute the summary.

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

TotalDepth.RP66V1.core.stringify

TotalDepth.RP66V1.core.stringify.**stringify_object_by_type**(*obj: Any*) → str
Convert objects to strings for HTML or text presentation.

TotalDepth.RP66V1.core.LogicalRecord.ComponentDescriptor

Implements the Component Descriptor [RP66V1 Section 3.2.2.1 Component Descriptor]

References:

RP66V1: http://w3.energistics.org/rp66/v1/rp66v1.html

Specifically section 3:

http://w3.energistics.org/rp66/v1/rp66v1_sec3.html

exception TotalDepth.RP66V1.core.LogicalRecord.ComponentDescriptor.**ExceptionComponentDescr**
General Exception class for Component Descriptor errors.

exception TotalDepth.RP66V1.core.LogicalRecord.ComponentDescriptor.**ExceptionComponentDescr**
Exception class for Component Descriptor __init__ errors.

exception TotalDepth.RP66V1.core.LogicalRecord.ComponentDescriptor.**ExceptionComponentDescr**
Exception class for Component Descriptor data access errors.

class TotalDepth.RP66V1.core.LogicalRecord.ComponentDescriptor.**RoleType**
Contains the role and type such as `('ABSATR', 'Absent Attribute')`

property role
Alias for field number 0

property type
Alias for field number 1

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *role: str*, *type: str*)
Create new instance of RoleType(role, type)

__repr__()
Return a nicely formatted representation string

class TotalDepth.RP66V1.core.LogicalRecord.ComponentDescriptor.**CharacteristicRepCodeGlobalI**
Contains the characteristic, rep_code and global_default such as `('Name', 'IDENT', b'')`

property characteristic
Alias for field number 0

property rep_code
Alias for field number 1

property global_default
Alias for field number 2

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls, characteristic: str, rep_code: Optional[str], global_default: Any*)
Create new instance of CharacteristicRepCodeGlobalDefault(characteristic, rep_code, global_default)

__repr__()
Return a nicely formatted representation string

class TotalDepth.RP66V1.core.LogicalRecord.ComponentDescriptor.**ComponentDescriptor**(*descriptor: int*)
Component Descriptor that extracts the necessary attributes from an integer.

See: [RP66V1 Section 3.2.2.1 Component Descriptor]

__init__(*descriptor: int*)
Initialize self. See help(type(self)) for accurate signature.

__eq__(*other*)
Return self==value.

__str__()
Return str(self).

property role
The Component Descriptor role.

property type
The Component Descriptor type.

property is_attribute_group
Is an attribute, absent attribute or invariant attribute.

property is_set_group
Is a set, redundant set or replacement set.

property is_absent_attribute
Is an absent attribute.

property is_attribute
Is an attribute.

property is_invariant_attribute
Is an invariant attribute.

property is_object
Is an object.

property is_redundant_set
Is a redundant set.

property is_replacement_set
Is a replacement set.

property is_set
Is a set.

property has_set_T
Has a type, must be in the set group otherwise an ExceptionComponentDescriptorAccessError will be raised.

property has_set_N
Has a name, must be in the set group otherwise an ExceptionComponentDescriptorAccessError will be raised.

property has_object_N
Has a name, must be an object otherwise an ExceptionComponentDescriptorAccessError will be raised.

property has_attribute_L
Has a label, must be in the attribute group otherwise an ExceptionComponentDescriptorAccessError will be raised.

property has_attribute_C
Has a count, must be in the attribute group otherwise an ExceptionComponentDescriptorAccessError will be raised.

property has_attribute_R
Has a Representation Code, must be in the attribute group otherwise an ExceptionComponentDescriptorAccessError will be raised.

property has_attribute_U
Has units, must be in the attribute group otherwise an ExceptionComponentDescriptorAccessError will be raised.

property has_attribute_V
Has a value, must be in the attribute group otherwise an ExceptionComponentDescriptorAccessError will be raised.

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

TotalDepth.RP66V1.core.LogicalRecord.EFLR

Implements the Explicitly Formatted Logical Record (EFFLR) [RP66V1 Section 3 Logical Record Syntax]

References: RP66V1: http://w3.energistics.org/rp66/v1/rp66v1.html

Specifically section 3: http://w3.energistics.org/rp66/v1/rp66v1_sec3.html

exception TotalDepth.RP66V1.core.LogicalRecord.EFLR.**ExceptionEFLR**
General Exception class for EFLR errors.

exception TotalDepth.RP66V1.core.LogicalRecord.EFLR.**ExceptionEFLRSet**
Exception class for EFLR Set errors.

exception TotalDepth.RP66V1.core.LogicalRecord.EFLR.**ExceptionEFLRSetDuplicateObjectNames**
Exception class for EFLR Set with duplicate object names.

exception TotalDepth.RP66V1.core.LogicalRecord.EFLR.**ExceptionEFLRAttribute**
Exception class for EFLR Attribute errors.

exception TotalDepth.RP66V1.core.LogicalRecord.EFLR.**ExceptionEFLRTemplate**
Exception class for EFLR Template errors.

exception TotalDepth.RP66V1.core.LogicalRecord.EFLR.**ExceptionEFLRTemplateDuplicateLabel**
Exception class for EFLR Template with duplicate object names.

exception TotalDepth.RP66V1.core.LogicalRecord.EFLR.**ExceptionEFLRObject**
Exception class for EFLR Object errors.

exception TotalDepth.RP66V1.core.LogicalRecord.EFLR.**ExceptionEFLRObjectDuplicateLabel**
Exception class for EFLR Object with duplicate labels.

class TotalDepth.RP66V1.core.LogicalRecord.EFLR.**Set**(*ld: TotalDepth.RP66V1.core.pFile.LogicalData*)
Class that represents a component set. See [RP66V1 3.2.2.1 Component Descriptor]

__init__(*ld: TotalDepth.RP66V1.core.pFile.LogicalData*)
Initialize self. See help(type(self)) for accurate signature.

__str__() → str
String representation.

__eq__(*other*) → bool
Equality operator.

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

class TotalDepth.RP66V1.core.LogicalRecord.EFLR.**AttributeBase**(*component_descriptor: TotalDepth.RP66V1.core.LogicalRecord.Com*
Class that represents a component attribute. See [RP66V1 3.2.2.1 Component Descriptor]

__init__(*component_descriptor: TotalDepth.RP66V1.core.LogicalRecord.ComponentDescriptor.ComponentDescriptor*)
Initialize self. See help(type(self)) for accurate signature.

__eq__(*other*)
Equality operator.

__str__() → str
String representation.

stringify_value(*stringify_function: Callable*) → str
Return the value as a string.

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

class TotalDepth.RP66V1.core.LogicalRecord.EFLR.**TemplateAttribute**(*component_descriptor: TotalDepth.RP66V1.core.LogicalRecor ld: TotalDepth.RP66V1.core.pFile.Logical*
Class that represents a component template. See [RP66V1 3.2.2.1 Component Descriptor]

__init__(*component_descriptor: TotalDepth.RP66V1.core.LogicalRecord.ComponentDescriptor.ComponentDescriptor*, *ld: TotalDepth.RP66V1.core.pFile.LogicalData*)
Initialize self. See help(type(self)) for accurate signature.

class TotalDepth.RP66V1.core.LogicalRecord.EFLR.**Attribute**(*component_descriptor: TotalDepth.RP66V1.core.LogicalRecord.Componen ld: TotalDepth.RP66V1.core.pFile.LogicalData, template_attribute: TotalDepth.RP66V1.core.LogicalRecord.EFLR.Tem*
Class that represents a component attribute. See [RP66V1 3.2.2.1 Component Descriptor]

__init__(*component_descriptor: TotalDepth.RP66V1.core.LogicalRecord.ComponentDescriptor.ComponentDescriptor*, *ld: TotalDepth.RP66V1.core.pFile.LogicalData*, *template_attribute: TotalDepth.RP66V1.core.LogicalRecord.EFLR.TemplateAttribute*)
Initialize self. See help(type(self)) for accurate signature.

class TotalDepth.RP66V1.core.LogicalRecord.EFLR.**Template**
Class that represents a component template. See [RP66V1 3.2.2.1 Component Descriptor]

__init__()
Initialize self. See help(type(self)) for accurate signature.

read(*ld: TotalDepth.RP66V1.core.pFile.LogicalData*)
Populate the template with the Logical Data.

__len__() → int
Return the number of columns described by this Template.

__getitem__(*item*) → TotalDepth.RP66V1.core.LogicalRecord.EFLR.TemplateAttribute
Get a TemplateAttribute by name or integer index.

__eq__(*other*) → bool
Equality operator.

__str__() → str
String representation.

header_as_strings(*stringify_function: Callable*) → List[str]
Return the TemplateAttributes as strings.

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

class TotalDepth.RP66V1.core.LogicalRecord.EFLR.**Object**(*ld: TotalDepth.RP66V1.core.pFile.LogicalData, template: TotalDepth.RP66V1.core.LogicalRecord.EFLR.Templat*
Class that represents a component object. See [RP66V1 3.2.2.1 Component Descriptor]. Essentially this is one row in the table as a list of Atributes.

__init__(*ld: TotalDepth.RP66V1.core.pFile.LogicalData*, *template: TotalDepth.RP66V1.core.LogicalRecord.EFLR.Template*)
Initialize self. See help(type(self)) for accurate signature.

__len__() → int
Return the number of attributes (columns) for this row.

__getitem__(*item*) → Optional[TotalDepth.RP66V1.core.LogicalRecord.EFLR.AttributeBase]
Get an Attribute (column) by name or integer index.

__eq__(*other*) → bool
Equality operator.

__str__() → str
String representation.

values_as_strings(*stringify_function: Callable*) → List[str]
Return the Attribute values as strings.

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

class TotalDepth.RP66V1.core.LogicalRecord.EFLR.**PublicEFLRType**
From [RP66V1 Appendix A: Logical Record Types] Figure A-2. Numeric Codes for Public EFLR Types.

property code
Alias for field number 0

property type
Alias for field number 1

property description
Alias for field number 2

property allowable_set_types
Alias for field number 3

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls, code: int, type: bytes, description: str, allowable_set_types: Set[bytes]*)
Create new instance of PublicEFLRType(code, type, description, allowable_set_types)

__repr__()
Return a nicely formatted representation string

TotalDepth.RP66V1.core.LogicalRecord.EFLR.**PUBLIC_EFLR_TYPES = {0: PublicEFLRType(code=0, t**
From [RP66V1 Appendix A: Logical Record Types] Figure A-2. Numeric Codes for Public EFLR Types

class TotalDepth.RP66V1.core.LogicalRecord.EFLR.**ExplicitlyFormattedLogicalRecord**(*lr_type: int*, *ld: TotalDepth.RP6*
Represents a RP66V1 Explicitly Formatted Logical Record (EFLR). Effectively this is a table containing a list of rows, each row is represented by an Object.

DUPE_OBJECT_STRATEGY = 3
The strategy for dealing with duplicate objects.

classmethod DUPE_OBJECT_LOGGER(*msg*, **args*, ***kwargs*)
What level to log duplicate object operations.

__init__ (*lr_type: int*, *ld: TotalDepth.RP66V1.core.pFile.LogicalData*)
Initialize self. See help(type(self)) for accurate signature.

__len__ () → int
Returns the number of rows in the table.

__getitem__ (*item*) → TotalDepth.RP66V1.core.LogicalRecord.EFLR.Object
Get an Object (row) by name or integer index.

__str__ () → str
Short string representation.

__eq__ (*other*) → bool
Equality operator.

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

str_long () → str
Returns a long string representing the table.

table_as_strings (*stringify_function: Callable*, *sort: bool*) → List[List[str]]
Returns a list of strings representing the table.

is_key_value () → bool
True if this is a key/value table.

key_values (*stringify_function: Callable*, *sort: bool*) → List[List[str]]
Returns a list of stringified key values. Will raise ExceptionEFLR if not a key/value table.

property shape
Shape as (rows, columns)

TotalDepth.RP66V1.core.LogicalRecord.EFLR.**reduced_object_map** (*eflr: TotalDepth.RP66V1.core.LogicalRecord.EFLR*) → Dict[bytes, int]
This returns a reduced lookup map that refers to the latest object by count. Key is the object IDENT, value is the ordinal into self.

TotalDepth.RP66V1.core.LogicalRecord.Encryption

Handles encryption in Logical Record Segments [RP66V1 Section 2.2.2.1], which is to say we do very little.

References:

RP66V1: http://w3.energistics.org/rp66/v1/rp66v1.html

TODO: Ensure this is consistent across the storage unit?

TotalDepth.RP66V1.core.LogicalRecord.IFLR

Indirectly Formatted Logical Records

exception TotalDepth.RP66V1.core.LogicalRecord.IFLR.**ExceptionIFLR**

class TotalDepth.RP66V1.core.LogicalRecord.IFLR.**IndirectlyFormattedLogicalRecord** (*lr_type: int*, *ld: TotalDepth.RP66*
Indirectly Formatted Logical Record has an OBNAME as its identity, a UVARI as the frame number then free

data. This just reads the OBNAME and UVARI but not the free data.

Reference: [RP66V1 Section 3.3 Indirectly Formatted Logical Record]

Reference: [RP66V1 Section 5.6.1 Frames] "The Frame Number is an integer (Representation Code UVARI) specifying the numerical order of the Frame in the Frame Type, counting sequentially from one."

__init__(*lr_type: int*, *ld: TotalDepth.RP66V1.core.pFile.LogicalData*)
Initialize self. See help(type(self)) for accurate signature.

__str__()
Return str(self).

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

TotalDepth.RP66V1.core.LogicalRecord.Semantics

The DLIS Semantics of Logical Records.

References:

[RP66V1 Chapter 4: Semantic Terminology and Rules] [RP66V1 Chapter 5: Semantics: Static and Frame Data]

Links:

http://w3.energistics.org/rp66/v1/rp66v1_sec4.html http://w3.energistics.org/rp66/v1/rp66v1_sec5.html

TODO: Use this for validating EFLRs.

class TotalDepth.RP66V1.core.LogicalRecord.Semantics.**Restrictions**(***kwargs*)
Imposes semantic restrictions.

__init__(***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

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

TotalDepth.RP66V1.core.LogicalRecord.Semantics.**FREQUENTLY_USED_ATTRIBUTES = {b'AXIS': <Tota**
[RP66V1 Section 4.4]

TotalDepth.RP66V1.core.LogicalRecord.Semantics.**SEMANTICS = {b'FRAME': {b'CHANNELS': <TotalI**
[RP66V1 Section 5]

TotalDepth.RP66V1.core.LogicalRecord.Types

Implements the Logical Record Syntax.

References: [RP66V1 Section 3 Logical Record Syntax]

In particular: [RP66V1 Appendix A logical Record Types]

TotalDepth.RP66V1.core.LogicalRecord.Types.**TypeDescriptionDataDescriptorObjectReferenceType**
alias of *TotalDepth.RP66V1.core.LogicalRecord.Types.TypeDescriptionAllowableSetTypes*

class TotalDepth.RP66V1.core.LogicalRecord.Types.**TypeDescriptionAllowableSetTypes**(*type*, *description*, *allowable_set_typ*

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *type*, *description*, *allowable_set_types*)
Create new instance of TypeDescriptionAllowableSetTypes(type, description, allowable_set_types)

__repr__()
Return a nicely formatted representation string

property allowable_set_types
Alias for field number 2

property description
Alias for field number 1

property type
Alias for field number 0

TotalDepth.RP66V1.IndexPickle

Read RP66V1 files and saves the index a s pickle file.

class TotalDepth.RP66V1.IndexPickle.**IndexResult**(*path_in*, *size_input*, *size_index*, *time_index*, *time_write*, *time_read*, *exception*, *ignored*)

property path_in
Alias for field number 0

property size_input
Alias for field number 1

property size_index
Alias for field number 2

property time_index
Alias for field number 3

property time_write
Alias for field number 4

property time_read
Alias for field number 5

property exception
Alias for field number 6

property ignored
Alias for field number 7

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *path_in: str*, *size_input: int*, *size_index: int*, *time_index: float*, *time_write: float*, *time_read: float*, *exception: bool*, *ignored: bool*)
Create new instance of IndexResult(path_in, size_input, size_index, time_index, time_write, time_read, exception, ignored)

__repr__()
Return a nicely formatted representation string

TotalDepth.RP66V1.IndexPickle.**unpickle**(*path: str*) → TotalDepth.RP66V1.core.LogicalFile.LogicalIndex
Un-pickles a Logical Index from the given path.

TotalDepth.RP66V1.IndexPickle.**index_dir_multiprocessing**(*dir_in: str*, *dir_out: str*, *jobs: int*, *recurse: bool*, *read_back: bool*) → Dict[str, TotalDepth.RP66V1.IndexPickle.IndexResult]
Multiprocessing code to plot log passes. Returns a dict of {path_in : IndexResult, ... }

TotalDepth.RP66V1.IndexXML

exception TotalDepth.RP66V1.IndexXML.**ExceptionRP66V1IndexXMLRead**

exception TotalDepth.RP66V1.IndexXML.**ExceptionIndexXML**

exception TotalDepth.RP66V1.IndexXML.**ExceptionIndexXMLRead**

exception TotalDepth.RP66V1.IndexXML.**ExceptionLogPassXML**

TotalDepth.RP66V1.IndexXML.**xml_write_value**(*xml_stream: TotalDepth.util.XmlWrite.XmlStream*, *value: Any*) → None
Write a value to the XML stream with specific type as an attribute. This writes either a <Value> or an <ObjectName> element.

TotalDepth.RP66V1.IndexXML.**frame_channel_to_XML**(*channel: TotalDepth.RP66V1.core.LogPass.RP66V1FrameChannel*, *xml_stream: TotalDepth.util.XmlWrite.XmlStream*) → None
Writes a XML Channel node suitable for RP66V1.

Example:

```
<Channel C="0" I="DEPTH" O="35" count="1" dimensions="1" long_name="Depth Channel
↪" rep_code="7" units="m"/>
```

TotalDepth.RP66V1.IndexXML.**frame_array_to_XML**(*frame_array: TotalDepth.RP66V1.core.LogPass.RP66V1FrameArray*, *iflr_data: Sequence[TotalDepth.RP66V1.core.XAxis.IFLRReference]*, *xml_stream: TotalDepth.util.XmlWrite.XmlStream*) → None
Writes a XML FrameArray node suitable for RP66V1.

Example:

```
<FrameArray C="0" I="0B" O="11" description="">
  <Channels channel_count="9">
    <Channel C="0" I="DEPT" O="11" count="1" dimensions="1" long_name="MWD Tool␣
↪Measurement Depth" rep_code="2" units="0.1 in"/>
    <Channel C="0" I="INC" O="11" count="1" dimensions="1" long_name="Inclination
↪" rep_code="2" units="deg"/>
    <Channel C="0" I="AZI" O="11" count="1" dimensions="1" long_name="Azimuth"␣
↪rep_code="2" units="deg"/>
    ...
  </Channels>
  <IFLR count="83">
      <FrameNumbers count="83" rle_len="1">
        <RLE datum="1" repeat="82" stride="1"/>
      </FrameNumbers>
      <LRSH count="83" rle_len="2">
        <RLE datum="0x14ac" repeat="61" stride="0x30"/>
        <RLE datum="0x2050" repeat="20" stride="0x30"/>
      </LRSH>
      <Xaxis count="83" rle_len="42">
        <RLE datum="0.0" repeat="1" stride="75197.0"/>
        <RLE datum="154724.0" repeat="1" stride="79882.0"/>
      </Xaxis>
  </IFLR>
</FrameArray>
```

TotalDepth.RP66V1.IndexXML.**log_pass_to_XML**(*log_pass: <module 'TotalDepth.common.LogPass' from '/Users/engun/Documents/workspace/TotalDepth/src/TotalDepth/comm*
iflr_data_map: Dict[Hashable, Sequence[TotalDepth.RP66V1.core.XAxis.IFLRReference]], *xml_stream: TotalDepth.util.XmlWrite.XmlStream*) → None

Writes a XML LogPass node suitable for RP66V1. Example:

```
<LogPass count="4">
    <FrameArray C="0" I="600T" O="44" description="">
        ...
    <FrameArray>
    ...
</LogPass>
```

TotalDepth.RP66V1.IndexXML.**write_logical_file_sequence_to_xml**(*logical_index: TotalDepth.RP66V1.core.LogicalFile.Logica*
output_stream: TextIO, *private: bool*) → None

Takes a LogicalIndex and writes the index to an XML stream.

class TotalDepth.RP66V1.IndexXML.**IndexResult**(*path_input*, *size_input*, *size_index*, *time*, *exception*, *ignored*)

property path_input
Alias for field number 0

property size_input
Alias for field number 1

property size_index
Alias for field number 2

property time
Alias for field number 3

property exception
Alias for field number 4

property ignored
Alias for field number 5

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *path_input: str*, *size_input: int*, *size_index: int*, *time: float*, *exception: bool*, *ignored: bool*)
Create new instance of IndexResult(path_input, size_input, size_index, time, exception, ignored)

__repr__()
Return a nicely formatted representation string

TotalDepth.RP66V1.IndexXML.**index_dir_multiprocessing**(*dir_in: str*, *dir_out: str*, *private: bool*, *jobs: int*) → Dict[str, TotalDepth.RP66V1.IndexXML.IndexResult]
Multiprocessing code to index in XML. Returns a dict of {path_in : IndexResult, ...}

TotalDepth.RP66V1.LogRecIndex

Exercises the LogicalRecordIndex on real files.

TotalDepth.RP66V1.LogRecIndex.**unpickle**(*path: str*) → TotalDepth.RP66V1.core.pIndex.LogicalRecordIndex
Unpickle a file and return an Index.LogicalRecordIndex.

class TotalDepth.RP66V1.LogRecIndex.**IndexResult**
Result of indexing a RP66V1 file.

property path_in
Alias for field number 0

property size_input
Alias for field number 1

property size_index
Alias for field number 2

property index_time
Alias for field number 3

property time_write
Alias for field number 4

property time_read_back
Alias for field number 5

property exception
Alias for field number 6

property ignored
Alias for field number 7

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *path_in: str*, *size_input: int*, *size_index: int*, *index_time: float*, *time_write: float*, *time_read_back: float*, *exception: bool*, *ignored: bool*)
Create new instance of IndexResult(path_in, size_input, size_index, index_time, time_write, time_read_back, exception, ignored)

__repr__()
Return a nicely formatted representation string

TotalDepth.RP66V1.LogRecIndex.**index_dir_multiprocessing**(*dir_in: str*, *dir_out: str*, *jobs: int*, *recurse: bool*, *read_back: bool*, *validate: bool*) → Dict[str, TotalDepth.RP66V1.LogRecIndex.IndexResult]
Multiprocessing code to plot log passes. Returns a dict of {path_in : IndexResult, ... }

TotalDepth.RP66V1.LogRecIndex.**index_a_single_file**(*path_in: str*, *path_out: str*, *read_back: bool*, *validate: bool*) → TotalDepth.RP66V1.LogRecIndex.IndexResult
Read a single file and return an IndexResult.

TotalDepth.RP66V1.LogRecIndex.**index_dir_or_file**(*path_in: str*, *path_out: str*, *recurse: bool*, *read_back: bool*, *validate: bool*) → Dict[str, TotalDepth.RP66V1.LogRecIndex.IndexResult]
Index a directory or file and return the results.

TotalDepth.RP66V1.LogRecIndex.**plot_gnuplot**(*data: Dict[str, TotalDepth.RP66V1.LogRecIndex.IndexResult]*, *gnuplot_dir: str*) → None
Plot the results as a Gnuplot graph.

TotalDepth.RP66V1.LogRecIndex.**main**() → int
Main entry point.

TotalDepth.RP66V1.Scan

Scans a RP66V1 file an prints out the summary.

This produces text output at various levels of encapsulation:

–VR ~ Visible Records only. –LRSH ~ Logical Record segments. –LD ~ Logical data i.e. all Logical Record segments concatenated for each Logical Record. –EFLR ~ Explicitly Formatted Logical Records. –IFLR ~ Implicitly Formatted Logical Records. –LR ~ All data, including frame data from all Logical Records.

TotalDepth.RP66V1.Scan.**scan_RP66V1_file_visible_records**(*fobj: BinaryIO*, *fout: TextIO*, ***kwargs*) → None
Scans the file reporting Visible Records, optionally Logical Record Segments as well.

class TotalDepth.RP66V1.Scan.**LRSHSummary**(*vr_position*, *lrsh_position*, *lrsh_attributes*, *lrsh_record_type*)

property vr_position
Alias for field number 0

property lrsh_position
Alias for field number 1

property lrsh_attributes
Alias for field number 2

property lrsh_record_type
Alias for field number 3

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *vr_position: int*, *lrsh_position: int*, *lrsh_attributes: TotalDepth.RP66V1.core.pFile.LogicalRecordSegmentHeaderAttributes*, *lrsh_record_type: int*)
Create new instance of LRSHSummary(vr_position, lrsh_position, lrsh_attributes, lrsh_record_type)

__repr__()
Return a nicely formatted representation string

TotalDepth.RP66V1.Scan.**scan_RP66V1_LRSH_consistency**(*fobj: BinaryIO*, *fout: TextIO*, ***kwargs*) → None
Look at the consistency of the sequence of LRSH.

TotalDepth.RP66V1.Scan.**scan_RP66V1_file_logical_data**(*fobj: BinaryIO*, *fout: TextIO*, ***kwargs*) → None
Scans the file reporting the raw Logical Data.

TotalDepth.RP66V1.Scan.**scan_RP66V1_file_EFLR_IFLR**(*fobj: BinaryIO*, *fout: TextIO*, ***kwargs*) → None
Scans the file reporting the individual EFLR and IFLR.

TotalDepth.RP66V1.Scan.**scan_RP66V1_file_data_content**(*fobj: BinaryIO*, *fout: TextIO*, ***, *rp66v1_path: str*, *frame_slice: TotalDepth.common.Slice.Slice*, *eflr_as_table: bool*) → None
Scans all of every EFLR and IFLR in the file using a ScanFile object.

TotalDepth.RP66V1.Scan.**dump_RP66V1_test_data**(*fobj: BinaryIO*, *fout: TextIO*, ***kwargs*) → None
Scans the file reporting Visible Records, optionally Logical Record Segments as well.

class TotalDepth.RP66V1.Scan.**IndexResult**(*size_input*, *size_output*, *time*, *exception*, *ignored*)

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *size_input*, *size_output*, *time*, *exception*, *ignored*)
Create new instance of IndexResult(size_input, size_output, time, exception, ignored)

__repr__()
Return a nicely formatted representation string

property exception
Alias for field number 3

property ignored
Alias for field number 4

property size_input
Alias for field number 0

property size_output
Alias for field number 1

property time
Alias for field number 2

TotalDepth.RP66V1.ScanHTML

Scans a RP66V1 file an writes out the summary in HTML.

TotalDepth.RP66V1.ScanHTML.**html_write_table_of_contents**(*logical_file_sequence: TotalDepth.RP66V1.core.LogicalFile.LogicalIndex*, *xhtml_stream: TotalDepth.util.XmlWrite.XhtmlStream*) → None

Write out the table of contents.

TotalDepth.RP66V1.ScanHTML.**html_write_body**(*logical_file_sequence: TotalDepth.RP66V1.core.LogicalFile.LogicalIndex*, *frame_slice: TotalDepth.common.Slice.Slice*, *xhtml_stream: TotalDepth.util.XmlWrite.XhtmlStream*, *sort_eflr: bool*) → TotalDepth.common.ToHTML.HTMLBodySummary

Write out the <body> of the document.

TotalDepth.RP66V1.ScanHTML.**html_scan_RP66V1_file_data_content**(*path_in: str*, *fout: TextIO*, *label_process: bool*, *frame_slice: TotalDepth.common.Slice.Slice*, *sort_eflr: bool*) → TotalDepth.common.ToHTML.HTMLBodyS

Scans all of every EFLR and IFLR in the file and writes to HTML. Similar to TotalDepth.RP66V1.core.Scan.scan_RP66V1_file_data_content Returns the text to use as a link.

TotalDepth.RP66V1.ScanHTML.**scan_a_single_file**(*path_in: str, path_out: str, label_process: bool, frame_slice: Union[TotalDepth.common.Slice.Slice, TotalDepth.common.Slice.Sample], sort_eflr: bool*) → TotalDepth.common.ToHTML.HTMLResult

Scan a single file and write out an HTML summary.

class TotalDepth.RP66V1.ScanHTML.**FileNameLinkHref**(*file_name*, *link_text*, *href*)

property file_name
Alias for field number 0

property link_text
Alias for field number 1

property href
Alias for field number 2

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *file_name: str*, *link_text: str*, *href: str*)
Create new instance of FileNameLinkHref(file_name, link_text, href)

__repr__()
Return a nicely formatted representation string

TotalDepth.RP66V1.ScanHTML.**scan_dir_multiprocessing**(*dir_in*, *dir_out*, *jobs*, *frame_slice: Union[TotalDepth.common.Slice.Slice, TotalDepth.common.Slice.Sample]*, *sort_eflr: bool*) → Dict[str, TotalDepth.common.ToHTML.HTMLResult]
Multiprocessing code to plot log passes. Returns a dict of {path_in : HTMLResult, ... }

TotalDepth.RP66V1.ScanHTML.**scan_dir_or_file**(*path_in: str*, *path_out: str*, *recursive: bool*, *label_process: bool*, *frame_slice: Union[TotalDepth.common.Slice.Slice, TotalDepth.common.Slice.Sample]*, *sort_eflr: bool*) → Dict[str, TotalDepth.common.ToHTML.HTMLResult]
Scans a directory or file putting the results in path_out. Returns a dict of {path_in : HTMLResult, ... }

TotalDepth.RP66V1.SearchFF01

TotalDepth.RP66V1.SearchFF01.**main**() → int
Searches for likely visible record headers which have 0xff01 as bytes two and three.

TotalDepth.RP66V1.ToLAS

Converts RP66V1 files to LAS files.

References:

- General: http://www.cwls.org/las/
- LAS Version 2: http://www.cwls.org/wp-content/uploads/2017/02/Las2_Update_Feb2017.pdf

Example reference to the LAS documentation in this source code:

```
[LAS2.0 Las2_Update_Feb2017.pdf Section 5.3 ~V (Version Information)]
```

TotalDepth.RP66V1.ToLAS.**write_las_header**(*input_file: str*, *logical_file: TotalDepth.RP66V1.core.LogicalFile.LogicalFile*, *logical_file_number: int*, *frame_array_ident: str*, *output_stream: TextIO*) → None
Writes the LAS opening such as:

```
~Version Information Section
VERS.           2.0                          : CWLS Log ASCII Standard - VERSION␣
↪2.0
WRAP.           NO                           : One Line per depth step
PROD.           TotalDepth                   : LAS Producer
PROG.           TotalDepth.RP66V1.ToLAS 0.1.1 : LAS Program name and version
CREA.           2012-11-14 10:50             : LAS Creation date [YYYY-MMM-DD␣
↪hh:mm]
DLIS_CREA.      2012-11-10 22:06             : DLIS Creation date and time [YYYY-
↪MMM-DD hhmm]
```

(continues on next page)

(continued from previous page)

```
SOURCE.          SOME-FILE-NAME.dlis           : DLIS File Name
FILE-ID.         SOME-FILE-ID                  : File Identification from the FILE-
↪HEADER Logical Record
LOGICAL-FILE.    3                             : Logical File number in the DLIS␣
↪file
FRAME-ARRAY.     60B                           : Identity of the Frame Array in␣
↪the Logical File
```

Reference: [LAS2.0 Las2_Update_Feb2017.pdf Section 5.3 ~V (Version Information)]

TotalDepth.RP66V1.ToLAS.**DLIS_TO_WELL_INFORMATION_LAS_EFLR_MAPPING = {b'PARAMETER': {b'APIN**

Mapping of DLIS EFLR Type and Object Name to LAS WELL INFORMATION section and Mnemonic. The Object LONG-NAME is used as the LAS description. We prefer to take data from the ORIGIN EFLR as it is more clearly specified [RP66V1 5.2 Origin Logical Record (OLR)] whereas the PARAMETER EFLR tables are fairly free form. See also [RP66V1 Section 5.8.2 Parameter Objects]

TotalDepth.RP66V1.ToLAS.**WELL_INFORMATION_FROM_ORIGIN = {b'COMPANY': 'COMP', b'CREATION-TIM**

[RP66V1 Section 5.2 Origin Logical Record (OLR)]

TotalDepth.RP66V1.ToLAS.**extract_well_information_from_origin**(*logical_file: TotalDepth.RP66V1.core.LogicalFile.LogicalF*) → Dict[str, TotalDepth.LAS.core.WriteLAS.UnitValueDes

Extracts partial well information from the ORIGIN record. Example:

```
Objects [1]:
OBNAME: O: 11 C: 0 I: b'DLIS_DEFINING_ORIGIN'
  CD: 001 00001 L: b'FILE-ID' C: 1 R: ASCII U: b'' V: [b'auto_las_survey']
  CD: 001 00001 L: b'FILE-SET-NAME' C: 1 R: IDENT U: b'' V: [b'']
  CD: 001 00001 L: b'FILE-SET-NUMBER' C: 1 R: UVARI U: b'' V: [1]
  CD: 001 00001 L: b'FILE-NUMBER' C: 1 R: UVARI U: b'' V: [1]
  CD: 001 00001 L: b'FILE-TYPE' C: 1 R: IDENT U: b'' V: [b'DEPTH-LOG']
  CD: 001 00001 L: b'PRODUCT' C: 1 R: ASCII U: b'' V: [b'RX6']
  CD: 001 00001 L: b'VERSION' C: 1 R: ASCII U: b'' V: [b'v0.0']
  CD: 000 00000 L: b'PROGRAMS' C: 1 R: ASCII U: b'' V: None
  CD: 001 00001 L: b'CREATION-TIME' C: 1 R: DTIME U: b'' V: [<<class 'TotalDepth.
↪RP66V1.core.RepCode.DateTime'> 2015-08-16 04:57:12.000 STD>]
  CD: 001 00001 L: b'ORDER-NUMBER' C: 1 R: ASCII U: b'' V: [b'0000']
  CD: 000 00000 L: b'DESCENT-NUMBER' C: 1 R: ULONG U: b'' V: None
  CD: 001 00001 L: b'RUN-NUMBER' C: 1 R: ULONG U: b'' V: [1]
  CD: 000 00000 L: b'WELL-ID' C: 1 R: IDENT U: b'' V: None
  CD: 001 00001 L: b'WELL-NAME' C: 1 R: ASCII U: b'' V: [b'PRASLIN 1                ␣
↪            ']
  CD: 001 00001 L: b'FIELD-NAME' C: 1 R: ASCII U: b'' V: [b'PRASLIN PROSPECT       ␣
↪             ']
  CD: 001 00001 L: b'PRODUCER-CODE' C: 1 R: UNORM U: b'' V: [440]
  CD: 001 00001 L: b'PRODUCER-NAME' C: 1 R: ASCII U: b'' V: [b'PathFinder']
  CD: 001 00001 L: b'COMPANY' C: 1 R: ASCII U: b'' V: [b'BURU ENERGY               ␣
↪           ']
  CD: 001 00001 L: b'NAME-SPACE-NAME' C: 1 R: IDENT U: b'' V: [b'PF']
  CD: 000 00000 L: b'NAME-SPACE-VERSION' C: 1 R: UVARI U: b'' V: None
```

TotalDepth.RP66V1.ToLAS.**write_well_information_to_las**(*logical_file: TotalDepth.RP66V1.core.LogicalFile.LogicalFile, frame_array: Optional[TotalDepth.RP66V1.core.LogPass.RP66V1Fran frame_slice: Union[TotalDepth.common.Slice.Slice, TotalDepth.common.Slice.Sample], ostream: TextIO*) → None

Writes the well information section.

Reference: [LAS2.0 Las2_Update_Feb2017.pdf Section 5.4 ~W (Well Information)]

TotalDepth.RP66V1.ToLAS.**write_parameter_section_to_las**(*logical_file: TotalDepth.RP66V1.core.LogicalFile.LogicalFile, ostream: TextIO*) → None

Write the PARAMETER tables to LAS.

TotalDepth.RP66V1.ToLAS.**las_file_name**(*path_out: str, logicial_file_index: int, frame_array_ident: bytes*) → str

Returns the output file name.

TotalDepth.RP66V1.ToLAS.**write_logical_index_to_las**(*logical_index: TotalDepth.RP66V1.core.LogicalFile.LogicalIndex, array_reduction: str, path_out: str, frame_slice: Union[TotalDepth.common.Slice.Slice, TotalDepth.common.Slice.Sample], channels: Set[str], field_width: int, float_format: str*) → List[str]

Take a Logical Index for a Logical File within a RP66V1 file and write out a set of LAS 2.0 files.

TotalDepth.RP66V1.ToLAS.**single_rp66v1_file_to_las**(*path_in: str, array_reduction: str, path_out: str, frame_slice: TotalDepth.common.Slice.Slice, channels: Set[str], field_width: int, float_format: str*) → TotalDepth.LAS.core.WriteLAS.LASWriteResult

Convert a single RP66V1 file to a set of LAS files.

TotalDepth.RP66V1.ToLAS.**main**() → int

Main entry point.

TotalDepth.RP66V1.util API Reference

Contents:

TotalDepth.RP66V1.util.IndexXMLRead

Example of reading the index XML files and mining them for data.

class TotalDepth.RP66V1.util.IndexXMLRead.**IndexResult**(*size_input*, *size_index*, *time*, *exception*, *ignored*, *analysis*)

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *size_input*, *size_index*, *time*, *exception*, *ignored*, *analysis*)
Create new instance of IndexResult(size_input, size_index, time, exception, ignored, analysis)

__repr__()
Return a nicely formatted representation string

property analysis
Alias for field number 5

property exception
Alias for field number 3

property ignored
Alias for field number 4

property size_index
Alias for field number 1

property size_input
Alias for field number 0

property time
Alias for field number 2

TotalDepth.RP66V1.util.IndexXMLRead.**read_a_single_index**(*xml_path_in: str*, *eflr_set_type: List[str]*) → TotalDepth.RP66V1.util.IndexXMLRead.IndexResult

Reads a single XML index and analyses it.

TotalDepth.RP66V1.util.XMLReadUnits

Reads XML files generated from RP66V1 files and explores them for the units that they use.

Example output of 1885 files:

Producer code names:

```
 -1 TERRASCIENCES, Inc.      :                1
  0                          :                6
 15 INTEQ                    :                2
126 CROCKER DATA PROCESSING  :                3
126 WEATHERFORD              :               13
150 ATLAS                    :              256
```

(continues on next page)

(continued from previous page)

```
280 Halliburton             :              273
440 PathFinder              :                2
440 Schlumberger            :            2,733
Producer code channel units:
 -1                         :                4
 -1 %                       :                8
 -1 GAPI                    :                2
 -1 UNITS                   :                1
 -1 brne                    :                1
 -1 deg                     :                4
 -1 degC                    :                2
 -1 frac                    :               15
 -1 g/cc                    :                2
 -1 in                      :                1
 -1 m                       :               14
 -1 m/hr                    :                1
 -1 ohmm                    :                7
 -1 ppm                     :                7
 -1 us/f                    :                4
 -1 v/v                     :                2
  0                         :               22
  0 F3                      :                2
  0 MMHO                    :               54
  0 MS10                    :                3
  0 b/e                     :                1
  0 deg                     :               16
  0 degC                    :                4
  0 flg                     :                4
  0 g                       :               18
  0 g/cm3                   :                4
  0 gAPI                    :                5
  0 gauss                   :                3
  0 in                      :               47
  0 lb                      :                1
  0 lbf                     :                6
  0 m                       :                6
  0 m/min                   :                3
  0 m/s                     :                4
  0 mV                      :                1
  0 ms                      :               10
  0 ohm.m                   :                5
  0 unitless                :               22
  0 us                      :               16
  0 us/ft                   :                5
  0 v/v                     :                8
 15 deg                     :                8
 15 m                       :                2
 15 ohm.m                   :                3
126                         :               25
126 CCPS                    :                4
126 DC/S                    :                1
126 G                       :                3
126 GAUSS                   :                3
126 MMHO                    :              106
126 MPM                     :                2
126 b/e                     :                2
126 b/elec                  :                3
```

(continues on next page)

(continued from previous page)

```
126 cps          :                1
126 deg          :               20
126 degC         :               14
126 ft/s         :                6
126 g            :               20
126 g/cc         :               10
126 g/cm3        :               44
126 gAPI         :               28
126 gapi         :                3
126 gauss        :                5
126 in           :              100
126 inch         :                5
126 lb           :                1
126 lbf          :               11
126 m            :               13
126 m/h          :                2
126 m/min        :                6
126 m3           :                4
126 mA           :                1
126 mD           :                1
126 mV           :               26
126 microseconds :               10
126 milliVolts   :                6
126 min          :                2
126 mm           :                3
126 mmho/m       :               30
126 ms           :                9
126 ohm.m        :               54
126 psi          :                1
126 pu           :               63
126 t            :                2
126 unitless     :                2
126 us           :               19
126 us/ft        :               11
126 v/v          :               38
150              :            8,033
150 %            :               96
150 1/psi        :               70
150 1/s          :               51
150 1E-2 T/m     :               18
150 1E-5 Oe      :              130
150 K            :                5
150 L            :              423
150 V            :              322
150 b/e          :                9
150 bbl          :                2
150 cP           :               35
150 cm           :               98
150 cm3          :              624
150 cm3/s        :              196
150 cu           :                3
150 dAPI         :               70
150 dB           :                6
150 dB/ft        :              261
150 deg          :              715
150 degC         :            1,083
150 degC/min     :              124
```

(continues on next page)

(continued from previous page)

```
150 degF          :              556
150 degF/min      :               34
150 ft            :               32
150 ft/s          :               70
150 ft3           :               18
150 ft3/bbl       :               70
150 g/cm3         :               67
150 gAPI          :              424
150 h             :              893
150 in            :            1,637
150 kHz           :              107
150 lbf           :            1,036
150 m             :            1,267
150 m/min         :              300
150 m3            :               90
150 mA            :              103
150 mD            :               28
150 mG            :              131
150 mS/m          :               79
150 mV            :              835
150 min           :              328
150 mm            :               67
150 ms            :              538
150 ohm.m         :              346
150 pF            :               35
150 ppm           :              127
150 psi           :            2,252
150 psi/min       :              347
150 pu            :              800
150 s             :            1,147
150 uV            :              175
150 us            :              342
150 us/ft         :              535
280               :            1,646
280 %             :               66
280 0.00          :                2
280 0.001/ohm     :                7
280 0.01          :                1
280 0.01 L        :               66
280 0.1           :                1
280 0.1 L/S       :               96
280 0.1 in        :                2
280 08.3          :                4
280 1.0/          :                8
280 1.0/S         :               24
280 1/PS          :               32
280 100           :               18
280 100 pu        :               12
280 C/C           :                4
280 DECP          :                9
280 G             :               14
280 G/CC          :                9
280 GOR           :               48
280 IN_2          :                1
280 Kv/K          :               35
280 MILS          :                2
280 NESW          :                1
```

(continues on next page)

(continued from previous page)

```
280 NONE         :             2,003
280 QLTY         :                 2
280 RPM          :                57
280 S            :               406
280 SEC          :                 3
280 V            :                49
280 api          :               133
280 b/e          :                28
280 cP           :                14
280 cc           :                45
280 ccps         :                90
280 cnts         :                 1
280 cps          :                32
280 dB           :                 1
280 decp         :               616
280 deg          :               212
280 degC         :               272
280 degF         :               123
280 dist         :                29
280 fph          :                 4
280 fpm          :                11
280 ft           :                11
280 ft3          :                20
280 g            :                28
280 g/c3         :                22
280 g/cc         :               152
280 gAPI         :               175
280 gapi         :                 1
280 gm/c         :                29
280 in           :               288
280 kHz          :                10
280 kg/m3        :                 8
280 klbf         :                 5
280 lb           :                 3
280 lbm          :                77
280 lbs          :               140
280 ltrs         :                11
280 m            :               316
280 m/mi         :                 4
280 m/min        :                44
280 m3           :                60
280 mA           :                29
280 mD           :                23
280 mG           :                 4
280 mPsec        :                 2
280 mS           :                49
280 mSEC         :                 2
280 mV           :                51
280 md/cp        :                96
280 min          :                66
280 mm           :               148
280 mmho         :                 3
280 mmho/m       :                48
280 mmo/         :                 1
280 mmo/m        :                76
280 mpm          :                38
280 ms           :               167
```

(continues on next page)

(continued from previous page)

```
280 nT           :              22
280 no.          :              29
280 ohm.         :               7
280 ohm.m        :              58
280 ohmm         :             390
280 pF           :              36
280 ppm          :              75
280 pres         :              29
280 psi          :             597
280 psia         :             965
280 psig         :              31
280 pu           :             127
280 sec          :               8
280 serv         :              29
280 time         :              29
280 uS           :               8
280 uS/f         :               6
280 uS/ft        :               3
280 ucts         :               7
280 us           :              78
280 us/m         :               5
280 uspf         :             101
280 v/v          :               5
280 vol          :              29
280 x8.3ms       :               6
280 z1x1         :             174
280 z1x2         :             174
280 z1y1         :             174
280 z1y2         :             174
280 z2x1         :              29
280 z2x2         :              29
280 z2y1         :              29
280 z2y2         :              29
280 zoom         :             203
440              :         327,687
440 %            :          16,349
440 ----         :             587
440 0.1 deg/m    :               1
440 0.1 in       :          29,587
440 0.5 ms       :               1
440 1/30 deg/m   :               6
440 1/min        :             100
440 1/s          :          11,907
440 10 ms        :               9
440 1000 ft.lbf  :              65
440 1000 kPa.s/m :              23
440 1000 kgf     :              53
440 1000 lbf     :             100
440 96.487 C/g   :               4
440 A            :           7,696
440 A/m          :           1,948
440 BAD_UNIT-?   :              14
440 CPS          :               1
440 GPa          :              27
440 Hz           :          15,966
440 L/min        :               5
440 MPa          :               1
```

(continues on next page)

(continued from previous page)

```
440 Mrayl        :              703
440 N            :            3,073
440 N.m          :                4
440 Oe           :               67
440 Pa           :               20
440 Pa.s         :                6
440 S            :                4
440 V            :           79,974
440 W            :               26
440 b/cm3        :                2
440 b/e          :                2
440 bar          :                8
440 c/min        :            5,748
440 c/s          :               43
440 cP           :              561
440 cm3          :           13,512
440 cm3/s        :            6,155
440 cu           :              214
440 dB           :            4,245
440 dB.mW        :               48
440 dB/m         :            1,461
440 deg          :            8,578
440 deg/100ft    :                2
440 deg/30m      :               88
440 deg/ft       :                6
440 deg/m        :              334
440 degC         :           25,640
440 degF         :            1,753
440 ft           :              719
440 ft/h         :            4,875
440 ft/min       :              721
440 ft/s2        :              368
440 ft2          :                6
440 ft3          :               28
440 ft3/bbl      :            1,014
440 ft3/ft3      :              136
440 g            :               44
440 g/cm3        :            8,827
440 gAPI         :           14,847
440 gal/min      :               61
440 gn           :               60
440 h            :            1,730
440 in           :           18,022
440 in2          :               87
440 kN.m         :                1
440 kPa          :           42,487
440 kPa/h        :              244
440 kg/m3        :               37
440 kgf/kgf      :              320
440 km.daN       :                2
440 lbf          :            4,957
440 lbm/gal      :               94
440 m            :           13,969
440 m/h          :              595
440 m/min        :            1,792
440 m/s          :              313
440 m/s2         :            5,142
```

(continues on next page)

(continued from previous page)

```
440 m2           :              885
440 m3           :              728
440 m3/m         :              126
440 m3/m3        :           20,861
440 mA           :           11,418
440 mD           :            1,024
440 mD.ft        :                6
440 mD.m         :              216
440 mPa.s        :              222
440 mS/m         :            3,407
440 mSv/h        :                5
440 mT           :              775
440 mV           :            5,218
440 mgn          :               11
440 min          :            5,312
440 min/ft       :               21
440 min/m        :              102
440 mm           :                1
440 mm2          :                7
440 ms           :           33,095
440 nT           :                7
440 nW           :                6
440 nm           :              135
440 ohm          :              223
440 ohm.m        :           19,449
440 ppk          :               42
440 ppm          :              790
440 psi          :          165,969
440 pu           :            1,522
440 rad          :               18
440 s            :            8,699
440 uA           :              864
440 uV           :            2,420
440 unitless     :                6
440 us           :            9,722
440 us/ft        :            8,995
440 us/m         :               28
Producer code attribute units:
 -1              :            1,458
 -1 GAPI         :                3
 -1 dCpm         :                1
 -1 deg          :                3
 -1 degC         :                4
 -1 frac         :                3
 -1 g/cc         :                3
 -1 in           :                1
 -1 lb/g         :                1
 -1 m            :               33
 -1 ohmm         :                3
 -1 ppm          :                2
 -1 us/f         :                4
  0              :            4,665
  0 DEG          :                2
  0 Hrs          :                1
  0 M            :               17
  0 Metres       :                8
  0 SEC/QT       :                1
```

(continues on next page)

(continued from previous page)

```
  0 deg C        :                8
  0 degC         :               20
  0 g/c3         :                1
  0 g/cm3        :                4
  0 in           :                8
  0 inches       :                6
  0 kg/m         :                4
  0 lb/USg       :                1
  0 lb/ft        :                4
  0 m            :               70
  0 mL/30min     :                4
  0 metres       :               19
  0 ml/30Min     :                2
  0 ohm-m        :                6
  0 ohm.m        :               16
  0 sec/qt       :                5
  0 us           :               22
  0 us/ft        :                4
 15              :              800
 15 Hrs          :                2
 15 M            :                3
 15 m            :               17
126              :           13,775
126 CP           :               14
126 Celsius      :               18
126 DEG C        :                2
126 FT           :                3
126 IN           :               13
126 INCHES       :                2
126 M            :               11
126 METRES       :                4
126 OHM-M        :                4
126 degC         :               97
126 g/cc         :               14
126 g/cm3        :                3
126 grams/cc     :                3
126 in           :              126
126 lbf/ft       :               51
126 m            :              328
126 metres       :               35
126 ml/30Min     :               15
126 mm           :                3
126 ohm.m        :               43
126 ohmm         :                6
126 pounds/ft    :                5
126 sec/qt       :                6
126 us           :                8
150              :       15,502,799
150 %            :              921
150 Hz           :              188
150 cP           :               98
150 cm           :              383
150 cm3          :              131
150 cm3/s        :                7
150 dB/ft        :              300
150 deg          :            1,119
150 degC         :              538
```

(continues on next page)

(continued from previous page)

```
150 degC/min     :                7
150 degF         :               50
150 ft           :               44
150 g/cm3        :              147
150 gAPI         :              181
150 in           :            1,255
150 kHz          :              792
150 lbf          :              186
150 lbm          :              905
150 lbm/ft       :              172
150 lbm/gal      :              140
150 m            :           10,387
150 m/min        :               23
150 mD/cP        :               14
150 mS/m         :               30
150 mV           :               87
150 mm           :              301
150 ms           :              214
150 ohm.m        :              357
150 ppm          :               14
150 psi          :              203
150 psi/min      :               87
150 pu           :                7
150 s            :           19,681
150 us           :            6,689
150 us/ft        :              258
280              :          260,292
280 $/da         :               32
280 %            :              133
280 0.1 in       :                4
280 Hz           :               46
280 N            :               19
280 cP           :               18
280 cps          :               12
280 cptm         :               77
280 deg          :            1,093
280 degC         :            1,484
280 degF         :              402
280 f-p          :                6
280 fph          :               12
280 ft           :              190
280 g            :              211
280 g/cc         :              221
280 gpm          :                6
280 hr           :               79
280 in           :              714
280 kHz          :              145
280 kPaa         :               19
280 kg/m3        :               22
280 kgm3         :               18
280 klb          :                6
280 lbpf         :              159
280 lbs          :              930
280 m            :            5,789
280 m/hr         :               25
280 mV           :               12
280 mm           :              981
```

(continues on next page)

(continued from previous page)

```
280 mmo/m        :              15
280 mpm          :             753
280 mptm         :              12
280 ms           :             289
280 nT           :             209
280 ohmm         :             504
280 pH           :              92
280 pa           :               2
280 ppg          :             120
280 ppm          :             273
280 psi          :             119
280 psia         :             867
280 psig         :              10
280 rpm          :               6
280 s            :             342
280 s/qt         :              78
280 sec          :             144
280 sg           :               6
280 spf          :               8
280 spl          :               2
280 spqt         :              12
280 ucts         :             220
280 us           :             385
280 us/m         :              31
280 uspf         :             215
440              :      35,769,005
440 %            :          34,429
440 0.01 degF/ft :              26
440 0.1 in       :           7,110
440 0.5 ms       :          11,097
440 1/s          :         147,530
440 1000 1/s     :             143
440 1000 ft.lbf  :               1
440 1000 kPa.s/m :              15
440 1000 kgf     :               1
440 1000 lbf     :               1
440 1000 lbm     :             834
440 1E-4 cm2/s   :              83
440 1E-5 Oe      :               3
440 1E-6 1/Pa    :             214
440 1E-6 1/psi   :              13
440 96.487 C/g   :               2
440 A            :           2,614
440 A/m          :             581
440 Hz           :           7,413
440 K            :             180
440 Mrayl        :             453
440 N            :          11,282
440 Oe           :             192
440 V            :         316,781
440 bbl          :             999
440 c/min        :           4,829
440 cP           :         611,762
440 cm           :           1,441
440 cm3          :       1,771,845
440 cm3/h        :               5
440 cm3/min      :           1,316
```

(continues on next page)

(continued from previous page)

```
440 cm3/s        :              124
440 cu           :              342
440 d            :                6
440 dAPI         :              763
440 dB           :            2,300
440 dB/m         :              283
440 deg          :          167,842
440 degC         :          209,611
440 degC/km      :              428
440 degC/m       :            2,166
440 degF         :            8,410
440 degF/ft      :               15
440 ft           :        2,043,783
440 ft/h         :            3,556
440 ft/s2        :                9
440 ft3          :              841
440 ft3/bbl      :               13
440 ft3/ft3      :               13
440 g            :               24
440 g/cm3        :           34,157
440 gAPI         :           74,124
440 gn           :               48
440 h            :            1,493
440 in           :          913,413
440 kHz          :               34
440 kN.m         :                1
440 kPa          :          860,887
440 kPa/h        :              118
440 keV          :            4,726
440 kg           :           69,794
440 kg/m         :              511
440 kg/m3        :               85
440 kgf/kgf      :              212
440 kohm         :               88
440 lbf          :            9,681
440 lbf/lbf      :               23
440 lbm          :              635
440 lbm/ft       :            2,997
440 lbm/gal      :            1,608
440 lbm/min      :               32
440 m            :          398,845
440 m/h          :               58
440 m/min        :               19
440 m/s          :               14
440 m/s2         :            5,330
440 m2/g         :               64
440 m3           :          275,811
440 m3/m3        :            1,132
440 mA           :            4,238
440 mD           :        1,223,325
440 mD.ft        :                1
440 mD/cP        :        1,633,855
440 mPa.s        :            1,442
440 mS/m         :            1,884
440 mT           :            5,997
440 mV           :            8,523
440 mV/m         :              111
```

(continues on next page)

(continued from previous page)

```
440 mbar          :               224
440 mgn           :                52
440 min           :           164,455
440 mm            :               544
440 mm2           :                96
440 mol/kg        :                96
440 ms            :             3,007
440 nT            :               974
440 nW            :                 4
440 nm            :               459
440 ohm           :             1,017
440 ohm.m         :            14,967
440 ppk           :                45
440 ppm           :            85,551
440 psi           :         4,190,937
440 psi/h         :                 4
440 pu            :             1,668
440 rad/ft3       :                13
440 rad/m3        :                52
440 s             :         1,764,018
440 s/m3          :                 5
440 uA            :             3,719
440 uV            :             7,845
440 us            :            12,236
440 us/ft         :             7,500
440 us/m          :                20


2020-08-31 11:28:31,102 - units.py         -  152 -  1801 - (MainThread) - INFO     -␣
↪Loading all unit standard forms into the cache

Channels:
Producer code: -1
  Found code: ['', '%', 'GAPI', 'degC', 'g/cc', 'in', 'm', 'ohmm', 'ppm']
Missing code: ['UNITS', 'brne', 'deg', 'frac', 'm/hr', 'us/f', 'v/v']
  Found standard form: ['deg', 'degC', 'in', 'm', 'ppm']
Missing standard form: ['', '%', 'GAPI', 'UNITS', 'brne', 'frac', 'g/cc', 'm/hr',
↪'ohmm', 'us/f', 'v/v']

Producer code: 0
  Found code: ['', 'F3', 'MMHO', 'b/e', 'degC', 'g', 'g/cm3', 'gAPI', 'gauss', 'in',
↪'lbf', 'm', 'm/min', 'm/s', 'mV', 'ms', 'ohm.m', 'us', 'us/ft']
Missing code: ['MS10', 'deg', 'flg', 'lb', 'unitless', 'v/v']
  Found standard form: ['deg', 'degC', 'g', 'g/cm3', 'gAPI', 'in', 'lbf', 'm', 'm/min
↪', 'm/s', 'ms', 'ohm.m', 'us', 'us/ft']
Missing standard form: ['', 'F3', 'MMHO', 'MS10', 'b/e', 'flg', 'gauss', 'lb', 'mV',
↪'unitless', 'v/v']

Producer code: 15
  Found code: ['m', 'ohm.m']
Missing code: ['deg']
  Found standard form: ['deg', 'm', 'ohm.m']
Missing standard form: []

Producer code: 126
  Found code: ['', 'G', 'MMHO', 'b/e', 'cps', 'degC', 'ft/s', 'g', 'g/cc', 'g/cm3',
↪'gAPI', 'gauss', 'in', 'lbf', 'm', 'm/h', 'm/min', 'm3', 'mA', 'mD', 'mV', 'min',
↪'mm', 'mmho/m', 'ms', 'ohm.m', 'psi', 'pu', 't', 'us', 'us/ft']
```

(continues on next page)

(continued from previous page)

```
Missing code: ['CCPS', 'DC/S', 'GAUSS', 'MPM', 'b/elec', 'deg', 'gapi', 'inch', 'lb',
↪'microseconds', 'milliVolts', 'unitless', 'v/v']
  Found standard form: ['deg', 'degC', 'ft/s', 'g', 'g/cm3', 'gAPI', 'in', 'lbf', 'm',
↪ 'm/h', 'm/min', 'm3', 'mD', 'min', 'mm', 'ms', 'ohm.m', 'psi', 'pu', 't', 'us',
↪'us/ft']
Missing standard form: ['', 'CCPS', 'DC/S', 'G', 'GAUSS', 'MMHO', 'MPM', 'b/e', 'b/
↪elec', 'cps', 'g/cc', 'gapi', 'gauss', 'inch', 'lb', 'mA', 'mV', 'microseconds',
↪'milliVolts', 'mmho/m', 'unitless', 'v/v']

Producer code: 150
  Found code: ['', '%', '1/psi', '1/s', '1E-5 Oe', 'K', 'L', 'V', 'b/e', 'bbl', 'cP',
↪'cm', 'cm3', 'cm3/s', 'cu', 'dAPI', 'dB', 'dB/ft', 'degC', 'degC/min', 'degF',
↪'degF/min', 'ft', 'ft/s', 'ft3', 'ft3/bbl', 'g/cm3', 'gAPI', 'h', 'in', 'kHz', 'lbf
↪', 'm', 'm/min', 'm3', 'mA', 'mD', 'mS/m', 'mV', 'min', 'mm', 'ms', 'ohm.m', 'pF',
↪'ppm', 'psi', 'psi/min', 'pu', 's', 'uV', 'us', 'us/ft']
Missing code: ['1E-2 T/m', 'deg', 'mG']
  Found standard form: ['1/psi', '1/s', '1E-5 Oe', 'K', 'L', 'V', 'bbl', 'cP', 'cm',
↪'cm3', 'cm3/s', 'cu', 'dAPI', 'dB', 'dB/ft', 'deg', 'degC', 'degC/min', 'degF',
↪'degF/min', 'ft', 'ft/s', 'ft3', 'ft3/bbl', 'g/cm3', 'gAPI', 'h', 'in', 'kHz', 'lbf
↪', 'm', 'm/min', 'm3', 'mD', 'mS/m', 'min', 'mm', 'ms', 'ohm.m', 'ppm', 'psi', 'psi/
↪min', 'pu', 's', 'uV', 'us', 'us/ft']
Missing standard form: ['', '%', '1E-2 T/m', 'b/e', 'mA', 'mG', 'mV', 'pF']

Producer code: 280
  Found code: ['', '%', '0.1 in', 'C/C', 'DECP', 'G', 'G/CC', 'NONE', 'RPM', 'S', 'SEC
↪', 'V', 'b/e', 'cP', 'cc', 'cps', 'dB', 'degC', 'degF', 'ft', 'ft3', 'g', 'g/cc',
↪'gAPI', 'in', 'kHz', 'kg/m3', 'lbm', 'm', 'm/mi', 'm/min', 'm3', 'mA', 'mD', 'mS',
↪'mV', 'min', 'mm', 'mmho', 'mmho/m', 'ms', 'nT', 'ohm.m', 'ohmm', 'pF', 'ppm', 'psi
↪', 'psig', 'pu', 'uS', 'us', 'us/m']
Missing code: ['0.00', '0.001/ohm', '0.01', '0.01 L', '0.1', '0.1 L/S', '08.3', '1.0/
↪', '1.0/S', '1/PS', '100', '100 pu', 'GOR', 'IN_2', 'Kv/K', 'MILS', 'NESW', 'QLTY',
↪'api', 'ccps', 'cnts', 'decp', 'deg', 'dist', 'fph', 'fpm', 'g/c3', 'gapi', 'gm/c',
↪'klbf', 'lb', 'lbs', 'ltrs', 'mG', 'mPsec', 'mSEC', 'md/cp', 'mmo/', 'mmo/m', 'mpm',
↪ 'no.', 'ohm.', 'pres', 'psia', 'sec', 'serv', 'time', 'uS/f', 'uS/ft', 'ucts',
↪'uspf', 'v/v', 'vol', 'x8.3ms', 'z1x1', 'z1x2', 'z1y1', 'z1y2', 'z2x1', 'z2x2',
↪'z2y1', 'z2y2', 'zoom']
  Found standard form: ['0.01', '0.1 in', 'S', 'V', 'cP', 'dB', 'deg', 'degC', 'degF',
↪ 'ft', 'ft3', 'g', 'gAPI', 'in', 'kHz', 'kg/m3', 'lbm', 'm', 'm/min', 'm3', 'mD',
↪'mS', 'min', 'mm', 'ms', 'nT', 'ohm.m', 'ppm', 'psi', 'pu', 'uS', 'us', 'us/m']
Missing standard form: ['', '%', '0.00', '0.001/ohm', '0.01 L', '0.1', '0.1 L/S', '08.
↪3', '1.0/', '1.0/S', '1/PS', '100', '100 pu', 'C/C', 'DECP', 'G', 'G/CC', 'GOR',
↪'IN_2', 'Kv/K', 'MILS', 'NESW', 'NONE', 'QLTY', 'RPM', 'SEC', 'api', 'b/e', 'cc',
↪'ccps', 'cnts', 'cps', 'decp', 'dist', 'fph', 'fpm', 'g/c3', 'g/cc', 'gapi', 'gm/c',
↪ 'klbf', 'lb', 'lbs', 'ltrs', 'm/mi', 'mA', 'mG', 'mPsec', 'mSEC', 'mV', 'md/cp',
↪'mmho', 'mmho/m', 'mmo/', 'mmo/m', 'mpm', 'no.', 'ohm.', 'ohmm', 'pF', 'pres', 'psia
↪', 'psig', 'sec', 'serv', 'time', 'uS/f', 'uS/ft', 'ucts', 'uspf', 'v/v', 'vol',
↪'x8.3ms', 'z1x1', 'z1x2', 'z1y1', 'z1y2', 'z2x1', 'z2x2', 'z2y1', 'z2y2', 'zoom']

Producer code: 440
  Found code: ['', '%', '----', '0.1 deg/m', '0.1 in', '0.5 ms', '1/min', '1/s', '10␣
↪ms', '1000 ft.lbf', '1000 kPa.s/m', '1000 kgf', '1000 lbf', 'A', 'A/m', 'CPS', 'GPa
↪', 'Hz', 'L/min', 'MPa', 'Mrayl', 'N', 'N.m', 'Oe', 'Pa', 'Pa.s', 'S', 'V', 'W', 'b/
↪cm3', 'b/e', 'bar', 'c/min', 'c/s', 'cP', 'cm3', 'cm3/s', 'cu', 'dB', 'dB.mW', 'dB/m
↪', 'deg/100ft', 'deg/30m', 'deg/ft', 'deg/m', 'degC', 'degF', 'ft', 'ft/h', 'ft/min
↪', 'ft/s2', 'ft2', 'ft3', 'ft3/bbl', 'ft3/ft3', 'g', 'g/cm3', 'gAPI', 'gal/min', 'gn
↪', 'h', 'in', 'in2', 'kN.m', 'kPa', 'kPa/h', 'kg/m3', 'kgf/kgf', 'km.daN', 'lbf',
↪'lbm/gal', 'm', 'm/h', 'm/min', 'm/s', 'm/s2', 'm2', 'm3', 'm3/m', 'm3/m3', 'mA',
↪'mD', 'mD.ft', 'mD.m', 'mPa.s', 'mS/m', 'mSv/h', 'mT', 'mV', 'mgn', 'min', 'min/ft',
↪ 'min/m', 'mm', 'mm2', 'ms', 'nT', 'nW', 'nm', 'ohm', 'ohm.m', 'ppk', 'ppm', 'psi',
↪'pu', 'rad', 's', 'uA', 'uV', 'us', 'us/ft', 'us/m']
```

(continues on next page)

(continued from previous page)

```
Missing code: ['1/30 deg/m', '96.487 C/g', 'BAD_UNIT-?', 'deg', 'unitless']
  Found standard form: ['0.1 deg/m', '0.1 in', '0.5 ms', '1/30 deg/m', '1/min', '1/s',
↪ '10 ms', '1000 ft.lbf', '1000 kgf', '1000 lbf', 'A', 'A/m', 'GPa', 'Hz', 'L/min',
↪'MPa', 'N', 'N.m', 'Oe', 'Pa', 'Pa.s', 'S', 'V', 'b/cm3', 'bar', 'c/min', 'c/s', 'cP
↪', 'cm3', 'cm3/s', 'cu', 'dB', 'dB.mW', 'dB/m', 'deg', 'deg/ft', 'deg/m', 'degC',
↪'degF', 'ft', 'ft/h', 'ft/min', 'ft/s2', 'ft2', 'ft3', 'ft3/bbl', 'ft3/ft3', 'g',
↪'g/cm3', 'gAPI', 'gal/min', 'h', 'in', 'in2', 'kN.m', 'kPa', 'kPa/h', 'kg/m3', 'kgf/
↪kgf', 'km.daN', 'lbf', 'lbm/gal', 'm', 'm/h', 'm/min', 'm/s', 'm/s2', 'm2', 'm3',
↪'m3/m', 'm3/m3', 'mD', 'mD.ft', 'mD.m', 'mPa.s', 'mS/m', 'mSv/h', 'mT', 'min', 'min/
↪ft', 'min/m', 'mm', 'mm2', 'ms', 'nT', 'nW', 'nm', 'ohm', 'ohm.m', 'ppk', 'ppm',
↪'psi', 'pu', 'rad', 's', 'uA', 'uV', 'us', 'us/ft', 'us/m']
Missing standard form: ['', '%', '----', '1000 kPa.s/m', '96.487 C/g', 'BAD_UNIT-?',
↪'CPS', 'Mrayl', 'W', 'b/e', 'deg/100ft', 'deg/30m', 'gn', 'mA', 'mV', 'mgn',
↪'unitless']


Attributes:
Producer code: -1
  Found code: ['', 'GAPI', 'degC', 'g/cc', 'in', 'm', 'ohmm', 'ppm']
Missing code: ['dCpm', 'deg', 'frac', 'lb/g', 'us/f']
  Found standard form: ['deg', 'degC', 'in', 'm', 'ppm']
Missing standard form: ['', 'GAPI', 'dCpm', 'frac', 'g/cc', 'lb/g', 'ohmm', 'us/f']

Producer code: 0
  Found code: ['', 'DEG', 'M', 'deg C', 'degC', 'g/cm3', 'in', 'kg/m', 'lb/ft', 'm',
↪'mL/30min', 'ohm.m', 'us', 'us/ft']
Missing code: ['Hrs', 'Metres', 'SEC/QT', 'g/c3', 'inches', 'lb/USg', 'metres', 'ml/
↪30Min', 'ohm-m', 'sec/qt']
  Found standard form: ['degC', 'g/cm3', 'in', 'kg/m', 'm', 'ohm.m', 'us', 'us/ft']
Missing standard form: ['', 'DEG', 'Hrs', 'M', 'Metres', 'SEC/QT', 'deg C', 'g/c3',
↪'inches', 'lb/USg', 'lb/ft', 'mL/30min', 'metres', 'ml/30Min', 'ohm-m', 'sec/qt']
Producer code: 15
  Found code: ['', 'M', 'm']
Missing code: ['Hrs']
  Found standard form: ['m']
Missing standard form: ['', 'Hrs', 'M']

Producer code: 126
  Found code: ['', 'CP', 'FT', 'IN', 'INCHES', 'M', 'METRES', 'degC', 'g/cc', 'g/cm3',
↪ 'in', 'm', 'mm', 'ohm.m', 'ohmm', 'us']
Missing code: ['Celsius', 'DEG C', 'OHM-M', 'grams/cc', 'lbf/ft', 'metres', 'ml/30Min
↪', 'pounds/ft', 'sec/qt']
  Found standard form: ['degC', 'g/cm3', 'in', 'm', 'mm', 'ohm.m', 'us']
Missing standard form: ['', 'CP', 'Celsius', 'DEG C', 'FT', 'IN', 'INCHES', 'M',
↪'METRES', 'OHM-M', 'g/cc', 'grams/cc', 'lbf/ft', 'metres', 'ml/30Min', 'ohmm',
↪'pounds/ft', 'sec/qt']

Producer code: 150
  Found code: ['', '%', 'Hz', 'cP', 'cm', 'cm3', 'cm3/s', 'dB/ft', 'degC', 'degC/min',
↪ 'degF', 'ft', 'g/cm3', 'gAPI', 'in', 'kHz', 'lbf', 'lbm', 'lbm/ft', 'lbm/gal', 'm',
↪ 'm/min', 'mD/cP', 'mS/m', 'mV', 'mm', 'ms', 'ohm.m', 'ppm', 'psi', 'psi/min', 'pu',
↪ 's', 'us', 'us/ft']
Missing code: ['deg']
  Found standard form: ['Hz', 'cP', 'cm', 'cm3', 'cm3/s', 'dB/ft', 'deg', 'degC',
↪'degC/min', 'degF', 'ft', 'g/cm3', 'gAPI', 'in', 'kHz', 'lbf', 'lbm', 'lbm/ft',
↪'lbm/gal', 'm', 'm/min', 'mD/cP', 'mS/m', 'mm', 'ms', 'ohm.m', 'ppm', 'psi', 'psi/
↪min', 'pu', 's', 'us', 'us/ft']
```

(continues on next page)

(continued from previous page)

```
Missing standard form: ['', '%', 'mV']

Producer code: 280
  Found code: ['', '%', '0.1 in', 'Hz', 'N', 'cP', 'cps', 'degC', 'degF', 'ft', 'g',
↪'g/cc', 'gpm', 'in', 'kHz', 'kPaa', 'kg/m3', 'm', 'mV', 'mm', 'ms', 'nT', 'ohmm',
↪'ppm', 'psi', 'psig', 'rpm', 's', 'us', 'us/m']
Missing code: ['$/da', 'cptm', 'deg', 'f-p', 'fph', 'hr', 'kgm3', 'klb', 'lbpf', 'lbs
↪', 'm/hr', 'mmo/m', 'mpm', 'mptm', 'pH', 'pa', 'ppg', 'psia', 's/qt', 'sec', 'sg',
↪'spf', 'spl', 'spqt', 'ucts', 'uspf']
  Found standard form: ['0.1 in', 'Hz', 'N', 'cP', 'deg', 'degC', 'degF', 'ft', 'g',
↪'in', 'kHz', 'kg/m3', 'm', 'mm', 'ms', 'nT', 'ppm', 'psi', 's', 'us', 'us/m']
Missing standard form: ['', '$/da', '%', 'cps', 'cptm', 'f-p', 'fph', 'g/cc', 'gpm',
↪'hr', 'kPaa', 'kgm3', 'klb', 'lbpf', 'lbs', 'm/hr', 'mV', 'mmo/m', 'mpm', 'mptm',
↪'ohmm', 'pH', 'pa', 'ppg', 'psia', 'psig', 'rpm', 's/qt', 'sec', 'sg', 'spf', 'spl',
↪ 'spqt', 'ucts', 'uspf']

Producer code: 440
  Found code: ['', '%', '0.01 degF/ft', '0.1 in', '0.5 ms', '1/s', '1000 1/s', '1000␣
↪ft.lbf', '1000 kPa.s/m', '1000 kgf', '1000 lbf', '1000 lbm', '1E-4 cm2/s', '1E-5 Oe
↪', '1E-6 1/Pa', '1E-6 1/psi', 'A', 'A/m', 'Hz', 'K', 'Mrayl', 'N', 'Oe', 'V', 'bbl',
↪ 'c/min', 'cP', 'cm', 'cm3', 'cm3/h', 'cm3/min', 'cm3/s', 'cu', 'd', 'dAPI', 'dB',
↪'dB/m', 'degC', 'degC/km', 'degC/m', 'degF', 'degF/ft', 'ft', 'ft/h', 'ft/s2', 'ft3
↪', 'ft3/bbl', 'ft3/ft3', 'g', 'g/cm3', 'gAPI', 'gn', 'h', 'in', 'kHz', 'kN.m', 'kPa
↪', 'kPa/h', 'keV', 'kg', 'kg/m', 'kg/m3', 'kgf/kgf', 'kohm', 'lbf', 'lbf/lbf', 'lbm
↪', 'lbm/ft', 'lbm/gal', 'lbm/min', 'm', 'm/h', 'm/min', 'm/s', 'm/s2', 'm2/g', 'm3',
↪ 'm3/m3', 'mA', 'mD', 'mD.ft', 'mD/cP', 'mPa.s', 'mS/m', 'mT', 'mV', 'mV/m', 'mbar',
↪ 'mgn', 'min', 'mm', 'mm2', 'mol/kg', 'ms', 'nT', 'nW', 'nm', 'ohm', 'ohm.m', 'ppk',
↪ 'ppm', 'psi', 'psi/h', 'pu', 'rad/ft3', 'rad/m3', 's', 's/m3', 'uA', 'uV', 'us',
↪'us/ft', 'us/m']
Missing code: ['96.487 C/g', 'deg']
  Found standard form: ['0.01 degF/ft', '0.1 in', '0.5 ms', '1/s', '1000 1/s', '1000␣
↪ft.lbf', '1000 kgf', '1000 lbf', '1000 lbm', '1E-4 cm2/s', '1E-5 Oe', '1E-6 1/Pa',
↪'1E-6 1/psi', 'A', 'A/m', 'Hz', 'K', 'N', 'Oe', 'V', 'bbl', 'c/min', 'cP', 'cm',
↪'cm3', 'cm3/h', 'cm3/min', 'cm3/s', 'cu', 'd', 'dAPI', 'dB', 'dB/m', 'deg', 'degC',
↪'degC/km', 'degC/m', 'degF', 'degF/ft', 'ft', 'ft/h', 'ft/s2', 'ft3', 'ft3/bbl',
↪'ft3/ft3', 'g', 'g/cm3', 'gAPI', 'h', 'in', 'kHz', 'kN.m', 'kPa', 'kPa/h', 'keV',
↪'kg', 'kg/m', 'kg/m3', 'kgf/kgf', 'kohm', 'lbf', 'lbf/lbf', 'lbm', 'lbm/ft', 'lbm/
↪gal', 'lbm/min', 'm', 'm/h', 'm/min', 'm/s', 'm/s2', 'm2/g', 'm3', 'm3/m3', 'mD',
↪'mD.ft', 'mD/cP', 'mPa.s', 'mS/m', 'mT', 'mV/m', 'min', 'mm', 'mm2', 'mol/kg', 'ms',
↪ 'nT', 'nW', 'nm', 'ohm', 'ohm.m', 'ppk', 'ppm', 'psi', 'psi/h', 'pu', 'rad/ft3',
↪'rad/m3', 's', 's/m3', 'uA', 'uV', 'us', 'us/ft', 'us/m']
Missing standard form: ['', '%', '1000 kPa.s/m', '96.487 C/g', 'Mrayl', 'gn', 'mA',
↪'mV', 'mbar', 'mgn']
```

TotalDepth.RP66V1.util.XMLReadUnits.**FileResult**
alias of TotalDepth.RP66V1.util.XMLReadUnits.IndexResult

TotalDepth.RP66V1.util.XMLReadUnits.**read_a_single_file**(*xml_path_in: str*, *accumulator: TotalDepth.RP66V1.util.XMLReadUnits.AggregateCou*
→ TotalDepth.RP66V1.util.XMLReadUnits.IndexResult

Reads a single XML index and analyses it.

TotalDepth.RP66V1.util.XMLReadUnits.**check_against_osdd_units**(*acc: TotalDepth.RP66V1.util.XMLReadUnits.Aggre*
→ None

Checks which units are in our version of the OSDD.

1.9.7 TotalDepth.DAT.DAT_parser

DAT Files

Parses DAT files. DAT is an informal specification (i.e. undefined) with loads of poor implementations.

Here described:

Section 1: Channel Declaration. A set of lines of the form: A B C, whitespace separated, where A is an uppercase word, B is free text, C is a word. There is no guarantee that channels declared here have any data in the subsequent sections. Order of this section is ignored.

Section 2: Channel Header A single line with space separated uppercase words. All words must appear as channel declarations from section 1 but some declarations from section 1 may be missing. The order of the words in the channel header is used to interpret the order of the subsequent values in section 3.

So deciding if we are in section 2 can be done with some (dubious?) heuristics:

- List matches list of section 1. Seems sensible but does not work when channels are declared but not defined.
- Some subset of the declared channels.
- All uppercase?
- Lots of words?
- Starts with 'UTIM DATE TIME . . . '

In this implementation we use the latter.

Section 3: Channel values Space separated values. Mostly numeric but date/time conversion can be inferred from section 1. The column order is defined by the order of the Channel Header.

Note many deficiencies here:

DATE Date ddmmyy but value 9Dec06, 09-Dec-06 etc.

TIME Time hhmmss but value 11-50-17

Example, <. . . > is continuation:

UTIM Unix Time sec DATE Date ddmmyy TIME Time hhmmss WAC Wits Activity Code unitless BDIA Bit Diameter inch <. . . > NPEN n-Pentane ppm EPEN Neo-Pentane ppm UTIM DATE TIME WAC BDIA <. . . > NPEN EPEN 1165665017 09Dec06 11-50-17 0 8.50 <. . . > 0 0

Performance

There is no particular effort made here for high performance. DAT files are small, typically <10Mb, so artful coding is not really required.

API

exception TotalDepth.DAT.DAT_parser.**ExceptionDAT**
General exception for problems with a DAT object.

exception TotalDepth.DAT.DAT_parser.**ExceptionDATRead**
Exception for reading a DAT file.

TotalDepth.DAT.DAT_parser.**RE_CHANNEL_DEFINITION = re.compile('^([A-Z0-9]+)\\s(.+?)\\s(\\S+)**
Matches 'UTIM Unix Time sec' Also need to process: "UTIM Unix Time sec"

TotalDepth.DAT.DAT_parser.**RE_DATA_HEADER_DEFINITION = re.compile('^UTIM\\s+DATE\\s+TIME\\s+**
Matches 'UTIM DATE TIME ...'

TotalDepth.DAT.DAT_parser.**RE_DATE_STYLE_A = re.compile('^(\\d+)(Jan|Feb|Mar|Apr|May|Jun|Jul**
Matches '12Oct20' and '5Oct20'

TotalDepth.DAT.DAT_parser.**RE_DATE_STYLE_B = re.compile('^(\\d+)-(Jan|Feb|Mar|Apr|May|Jun|Ju**
Matches '12-Oct-20' and '5O-Oct-20'

TotalDepth.DAT.DAT_parser.**NAME_UNITS_TYPE_MAP = {('DATE', 'ddmmyy'): <class 'object'>, ('**
Map of numpy dtype from the name/units.

TotalDepth.DAT.DAT_parser.**parse_file**(*file_object: TextIO*, *ident: str = ''*, *description: str = 'DAT File'*) → TotalDepth.common.LogPass.FrameArray
Parse the File object as a DAT file into a FrameArray. Will raise an ExceptionDAT on error.

Parameters

- **file_object** – The file to parse.
- **ident** – Identification of this DAT file.
- **description** – Description of this DAT file.

Returns

TotalDepth.DAT.DAT_parser.**parse_path**(*path: str*) → TotalDepth.common.LogPass.FrameArray
Parse the DAT file in the given path.

TotalDepth.DAT.DAT_parser.**can_parse_file**(*file_object: TextIO*) → bool
Tries to parse the file with just one row of data. On error returns False.

TotalDepth.DAT.DAT_parser.**can_parse_path**(*path: str*) → bool
Tries to parse the file at path with just one row of data. On error returns False.

TotalDepth.DAT.DAT_parser.**main**() → int
Read a file and dump the Log Pass.

1.9.8 TotalDepth.common - Common Package Reference

Contents:

TotalDepth.common.cmn_cmd_opts (Common Command Line Options)

Common command line options, this is to try and present some degree of interface consistency among command line applications.

TotalDepth.common.cmn_cmd_opts.**arg_parser**(*desc*, *prog=None*, *version=None*, ***kwargs*) → argparse.ArgumentParser
Return an command line parser with the standard pre-set options.

Standard options are `-h, --version` and:

`-k`: Flag to indicate that we should keep going as far as sensible.

`-l`: Log level.

`-v`: Verbosity.

TotalDepth.common.cmn_cmd_opts.**path_in**(**args*, ***kwargs*) → argparse.ArgumentParser
Return an command line parser with the standard pre-set options plus an input path as an argument.

TotalDepth.common.cmn_cmd_opts.**path_in_out**(**args*, ***kwargs*) → argparse.ArgumentParser
Return an command line parser with the standard pre-set options plus an input and output paths as an arguments.

TotalDepth.common.cmn_cmd_opts.**path_in_out_required**(**args*, ***kwargs*) → argparse.ArgumentParser
Return an command line parser with the standard pre-set options plus an input and output paths as an arguments.

TotalDepth.common.cmn_cmd_opts.**DEFAULT_OPT_LOG_LEVEL = 30**
Default log level

TotalDepth.common.cmn_cmd_opts.**DEFAULT_OPT_LOG_FORMAT = '%(asctime)s %(process)d %(levelnar**
Default log format (terse)

TotalDepth.common.cmn_cmd_opts.**DEFAULT_OPT_LOG_FORMAT_VERBOSE = '%(asctime)s - %(filename)-**
Default log format (verbose)

TotalDepth.common.cmn_cmd_opts.**add_log_level**(*parser: argparse.ArgumentParser*, *level: int = 30*) → None
Adds log level to the argument parser. The value can be either an integer of a string so 20 and 'INFO' are equivalent.

TotalDepth.common.cmn_cmd_opts.**set_log_level**(*parsed_args*, *format: str = '%(asctime)s - %(filename)-16s#%(lineno)-4d - %(process)5d - (%(threadName)-10s) - %(levelname)-8s - %(message)s'*) → int
Initialise logging.

TotalDepth.common.cmn_cmd_opts.**add_multiprocessing**(*parser: argparse.ArgumentParser*) → None
Adds log level to the argument parser as –jobs.

TotalDepth.common.cmn_cmd_opts.**multiprocessing_requested**(*parsed_args*) → bool
Returns True if the `--jobs=` option requires multiprocessing.

TotalDepth.common.cmn_cmd_opts.**number_multiprocessing_jobs**(*parsed_args*) → int
Returns the number of multiprocessing nodes interpreted from the``–jobs=`` option.

TotalDepth.common.colorama

Abstraction over colorama

TotalDepth.common.colorama.**section**(*title: str*, *fillchar: str*, *colour: colorama.ansi.AnsiFore = '\x1b[32m'*, *width: int = 75*, *out_stream: TextIO = <colorama.ansitowin32.StreamWrapper object>*)

Write a coloured header and trailer.

TotalDepth.common.data_table (Pretty Printing Tables)

Formats a table as a list of printable strings.

TODO: Support specific styles:

Sphinx style:

```
+------------------------+------------+----------+----------+
| Header row, column 1   | Header 2   | Header 3 | Header 4 |
| (header rows optional) |            |          |          |
+========================+============+==========+==========+
| body row 1, column 1   | column 2   | column 3 | column 4 |
+------------------------+------------+----------+----------+
| body row 2             | ...        | ...      |          |
+------------------------+------------+----------+----------+
```

Result:

Header row, column 1 (header rows optional)	Header 2	Header 3	Header 4
body row 1, column 1	column 2	column 3	column 4
body row 2	...	...	

Markdown style:

```
| Calculation | Video Time t (s) | Distance from start of Runway (m) | Distance from␣
↪start of Runway at t=0 (m) |
| --- | --: | --: | --: |
| Mid speed -10 knots | -30.1 | 537 | 1325 |
| Mid speed | -32.8 | 232 | 1182 |
| Mid speed +10 knots | -35.6 | -87 | 1039 |
| **Range and worst error** | **-32.8 ±2.8** | **232 ±319** | **1182 ±143** |
```

TotalDepth.common.data_table.**format_object**(*o: Any*) → str

Format a value as a string.

TotalDepth.common.data_table.**format_table**(*rows: Sequence[Sequence[Any]]*, *pad: str = ' '*, *heading_underline: str = ''*, *left_flush: bool = False*) → List[str]

Given a table of strings this formats them as a list of strings.

TotalDepth.common.data_table.**format_table_columns**(*rows: Sequence[Sequence[Any]]*, *column_formats: List[str]*, *pad: str = ' '*, *heading_underline: str = ''*) → List[str]

Given a list of objects this formats them as a list of strings.

TotalDepth.common.LogPass

This provides a file format agnostic representation of a LogPass.

- A LogPass consists of a set of FrameArray(s).
- A FrameArray consists of a set of FrameChannel(s).
- A FrameChannel consists of a set of values in a Numpy array of any shape from a single recorded channel.

exception TotalDepth.common.LogPass.**ExceptionLogPassBase**
General exception for problems with this module.

exception TotalDepth.common.LogPass.**ExceptionLogPass**
General exception for problems with a LogPass object.

exception TotalDepth.common.LogPass.**ExceptionFrameChannel**
General exception for problems with a FrameChannel object.

exception TotalDepth.common.LogPass.**ExceptionFrameArray**
General exception for problems with a FrameArray object.

class TotalDepth.common.LogPass.**FrameChannel**(*ident: Hashable, long_name: Union[str, bytes], units: Union[str, bytes], shape: Tuple[int, ...], np_dtype: numpy.dtype*)
This represents a single channel in a frame. It is file format independent and can be used depending on the source of the information: LIS/LAS/RP66V1 file, XML index, Postgres database etc.

__init__(*ident: Hashable, long_name: Union[str, bytes], units: Union[str, bytes], shape: Tuple[int, ...], np_dtype: numpy.dtype*)
Constructor.

Parameters

- **ident** – Some hashable identity.
- **long_name** – A description of the channel
- **units** – Units of Measure.
- **shape** – A list of dimensions of each value. [1] is a single value per frame. [4, 1024] is a 4 * 1024 matrix such as sonic waveform of 1024 samples with 4 waveforms per frame.
- **np_dtype** – The numpy dtype to use.

property ident
Overload this if necessary, for example RP66V1 has an OBNAME.

__str__() → str
Return str(self).

__getitem__(*key*)
Gets the value in the numpy array where key is a tuple of integers of length self.dimensions. For example this might be from self.numpy_indexes().

__setitem__(*key*, *value*)
Sets the value in the numpy array where key is a tuple of integers of length self.dimensions. For example this might be from self.numpy_indexes().

init_array(*number_of_frames: int*) → None
Initialises an empty Numpy array suitable to fill with <frames> number of frame data for this channel. If an array already exists of the correct length it is reused.

property array_size
The number of elements in the numpy array.

property sizeof_array
The size of each element of the current array as represented by numpy.

property sizeof_frame
The size of a single frame in bytes as represented by numpy.

numpy_indexes(*frame_number: int*) → itertools.product
Returns a generator of numpy indexes for a particular frame.

Example for a 2 x 3 array, given frame index 7:

```
>>> list(itertools.product([7], [0,1], [0,1,2]))
[(7, 0, 0), (7, 0, 1), (7, 0, 2), (7, 1, 0), (7, 1, 1), (7, 1, 2)]
>>> list(itertools.product([7], range(2), range(3)))
[(7, 0, 0), (7, 0, 1), (7, 0, 2), (7, 1, 0), (7, 1, 1), (7, 1, 2)]
```

Usage, where *function* is a conversion function on the data:

```
for dim in self.numpy_indexes(frame_number):
    # dim is a tuple of length self.rank + 1
    self.array[dim] = some_value
```

mask_array(*absent_value: Union[None, int, float]*) → None
Masks the absent values.

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

class TotalDepth.common.LogPass.**FrameArray**(*ident: Hashable, description: Union[str, bytes]*)
Represents a set of channels recorded simultaneously. In the olden days we would record this on a single piece of continuous film.

Subclass this depending on the source of the information: LIS/LAS/DLIS file, XML index etc.

__init__(*ident: Hashable, description: Union[str, bytes]*)
Initialize self. See help(type(self)) for accurate signature.

append(*channel: TotalDepth.common.LogPass.FrameChannel*) → None
Add a channel to the Array.

__str__() → str
Return str(self).

__len__() → int
The number of channels.

property sizeof_array
The total of the current frame array as represented by numpy.

property sizeof_frame
The size of the internal representation of a frame as represented by numpy.

property shape
The shape of the frame array.

init_arrays(*number_of_frames: int*) → None
Initialises empty Numpy arrays for each channel suitable to fill with <frames> number of frame data.

init_arrays_partial(*number_of_frames: int, channels: Set[Hashable]*) → None
Initialises empty Numpy arrays for each of the specified channels suitable to fill with <frames> number of frame data. The channels parameter limits the initialisation to only those channels. Unknown channels in that parameter are ignored.

mask_array(*absent_value: Union[int, float]*) → None
Mask the absent values in all but the index channels.

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

class TotalDepth.common.LogPass.**LogPass**
This represents the structure a single run of data acquisition such as 'Repeat Section' or 'Main Log'. These runs have one or more independent simultaneous recordings of different sensors at different depth/time resolutions. Each of these simultaneous recordings is represented as a FrameArray object.

- A LogPass consists of a set of FrameArray(s).
- A FrameArray consists of a set of FrameChannel(s).
- A FrameChannel consists of a set of values in a Numpy array of any shape from a single recorded channel (sensor).

This is a file format independent design. Different file formats use this in different ways:

- LIS79 - The standard allows 2 simultaneous FrameArrays, IFLR type 0, 1. Type 1 has never been seen in the wild.
- LAS (all versions) - The standard excludes simultaneous FrameArrays.
- RP66V1 - The standard allows for any number of simultaneous FrameArrays and this is common.
- DAT - No simultaneous FrameArrays.
- BIT - Custom and practice shows that there can be any number of simultaneous FrameArrays.

__init__()
Initialize self. See help(type(self)) for accurate signature.

has(*key: Hashable*) → bool
Returns True if the key is in the Frame Array Map.

keys() → Iterable
The identities of the Frame Arrays.

append(*frame_array: TotalDepth.common.LogPass.FrameArray*) → None
Add a channel to the Array.

__str__() → str
Return str(self).

__len__() → int
The number of Frame Arrays.

__getitem__(*item: Union[int, str, bytes]*) → TotalDepth.common.LogPass.FrameArray
The Frame Array by index or ID.

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

TotalDepth.common.lookup_mnemonic

Schlumberger Data

https://www.apps.slb.com/cmd/

"The Curve Mnemonic Dictionary is the publicly accessible version of the Oilfield Services Data Dictionary (OSDD)."

Mnemonics

URL is of the form:

```
https://www.apps.slb.com/cmd/<TYPE>.aspx?code=<NAME>
```

Where <TYPE> is:

Data channels: ChannelItem Parameters: ParameterItem Logging Tools: ToolItem Software Products: ProductItem

Examples:

```
https://www.apps.slb.com/cmd/ChannelItem.aspx?code=RHOB
https://www.apps.slb.com/cmd/ParameterItem.aspx?code=LATI
https://www.apps.slb.com/cmd/ToolItem.aspx?code=HDT
```

Anything that does not start with [A-Z] is in https://www.apps.slb.com/cmd/ChannelsList.aspx?start=na

Example:

```
https://www.apps.slb.com/cmd/ChannelItem.aspx?code=A0
```

Content contains the key/value table:

```
<table cellspacing="5" cellpadding="1" id="main_DetailsView1" style="width:492px;">
    <tr>
        <td style="font-weight:bold;">Channel</td>
        <td>A0</td>
    </tr>
    <tr>
        <td style="font-weight:bold;">Description</td>
        <td>Analog 0 (Regular)</td>
    </tr>
    <tr>
        <td style="font-weight:bold;">Unit quantity</td>
        <td>
            <a id="main_DetailsView1_HyperLink1" href="UOMDetail.aspx?
↪dim=ElectricPotential">ElectricPotential</a>
        </td>
    </tr>
    <tr>
        <td style="font-weight:bold;">Property</td>
        <td>
            <a id="main_DetailsView1_HyperLink4" href="PropertyItem.aspx?
↪code=Electric_Potential">Electric_Potential</a>
        </td>
    </tr>
</table>
```

Data Channels

Example:

```
https://www.apps.slb.com/cmd/ChannelItem.aspx?code=RHOB
```

Channels have the attributes: Channel, Description, Unit quantity, Property as key/value in a table 'main_DetailsView1'.

In the table 'main_GridView1' there are 'Related tools' as a list of key/value:

```
<th scope="col">Tool</th><th scope="col">Description</th>
```

In the table 'main_GridView2' there are 'Related products' as a list of key/value:

```
<th scope="col">Product</th><th scope="col">Description</th>
```

Parameters

Example:

```
https://www.apps.slb.com/cmd/ParameterItem.aspx?code=LATI
```

Parameters have the attributes: Code, Description, Unit quantity, Property in a key/value table 'main_DetailsView1'.

They also have 'Related products' in the 'main_GridView2' a table with a list of key/values:

```
<th scope="col">Product</th><th scope="col">Description</th>
```

Logging Tools

Example:

```
https://www.apps.slb.com/cmd/ToolItem.aspx?code=HDT
```

In the table 'main_DetailsView1' a ToolItem has the attributes: Code, Technology, Discipline, Method, Description as a key/value table.

Example:

```
Code        HDT
Technology  Dipmeter
Discipline  Geology
Method      WIRELINE
Description High Resolution Dipmeter Tool
```

They also have 'Related Channels' in the 'main_GridView1' a table a list of key/values:

```
<th scope="col">Channel</th><th scope="col">Description</th>
```

They also have 'Related Parameters' in the 'main_GridView2' a table a list of key/values:

```
<th scope="col">Parameter</th><th scope="col">Description</th>
```

Software Products

Example:

```
https://www.apps.slb.com/cmd/ProductItem.aspx?code=CALCULATE_TRAJECTORY
```

In the table 'main_DetailsView1' has the attributes: Code, Name, Discipline, Type, Description as a key/value table.

They also have 'Related Channels' in the 'main_GridView1' a table a list of key/values with links to ChannelItem:

```
<th scope="col">Channel</th><th scope="col">Description</th>
```

Properties

Example:

```
https://www.apps.slb.com/cmd/PropertyItem.aspx?code=Diameter
```

In the table 'main_DetailsView1' has the attributes: Code, Name, Parents, Description as a key/value table.

Parents are as a hierarchy, for example:

```
<td style="font-weight:bold;">Parents</td>
<td> 
    <span id="main_DetailsView1_labParents">
        <a href="PropertyItem.aspx?code=Property">Property</a>
    &gt;
        <a href="PropertyItem.aspx?code=Length">Length</a>
    </span>
</td>
```

They also have 'Related children' in the 'main_GridView1' a table a list of key/values with links to PropertyItem:

```
<th scope="col">Code</th><th scope="col">Name</th>
```

Units

See slb_units()

Unit assingnment:

https://www.apps.slb.com/cmd/unitassignment.aspx

Example:

```
<table cellspacing="0" cellpadding="4" id="main_GridView1" style="width:940px;border-
↪collapse:collapse;">
    <tr align="left" style="background-color:#E0E0E0;font-size:X-Small;">
        <th scope="col">Unit</th>
        <th scope="col">Unit System</th>
        <th scope="col">Unit Quantity</th>
        <th scope="col">Dimension</th>
    </tr><tr>
        <td>(bbl/d)/(rev/s)</td>
        <td>ProductionEnglish</td>
```

(continues on next page)

(continued from previous page)

```
        <td>FlowratePerRotationalVelocity</td>
        <td>VolumePerRotation</td>
    </tr><tr>
        <td>(rev/s)/(ft/min)</td>
        <td>ProductionEnglish</td>
        <td>RotationalVelocityPerVelocity</td>
        <td>RotationPerLength</td>
    </tr><tr>
        <td>gn</td>
        <td>Metric</td>
        <td>Gravity</td>
        <td>Acceleration</td>
    </tr>
```

exception TotalDepth.common.lookup_mnemonic.**ExceptionLookupMnemonic**

exception TotalDepth.common.lookup_mnemonic.**ExceptionLookupMnemonicReadURL**

exception TotalDepth.common.lookup_mnemonic.**ExceptionLookupMnemonicTable**

exception TotalDepth.common.lookup_mnemonic.**ExceptionLookupMnemonicReadTable**

TotalDepth.common.lookup_mnemonic.**decompose_table_by_header_row**(*parse_tree: bs4.BeautifulSoup*, *table_id: str*) → List[Dict[str, str]]

Return a list of rows from an HTML table of given ID.

class TotalDepth.common.lookup_mnemonic.**ProductDescription**(*product*, *description*)

property product
Alias for field number 0

property description
Alias for field number 1

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *product: str*, *description: str*)
Create new instance of ProductDescription(product, description)

__repr__()
Return a nicely formatted representation string

class TotalDepth.common.lookup_mnemonic.**ToolDescription**(*tool*, *description*)

property tool
Alias for field number 0

property description
Alias for field number 1

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *tool: str*, *description: str*)
Create new instance of ToolDescription(tool, description)

__repr__()
Return a nicely formatted representation string

class TotalDepth.common.lookup_mnemonic.**Channel**(*channel*, *description*, *unit_quantity*, *property*, *related_tools*, *related_products*)

property channel
Alias for field number 0

property description
Alias for field number 1

property unit_quantity
Alias for field number 2

property property
Alias for field number 3

property related_tools
Alias for field number 4

property related_products
Alias for field number 5

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls, channel: str, description: str, unit_quantity: str, property: str, related_tools: Tuple[TotalDepth.common.lookup_mnemonic.ToolDescription], related_products: Tuple[TotalDepth.common.lookup_mnemonic.ProductDescription]*)
Create new instance of Channel(channel, description, unit_quantity, property, related_tools, related_products)

__repr__()
Return a nicely formatted representation string

TotalDepth.common.lookup_mnemonic.**slb_data_channel**
Returns the Channel corresponding to the name. This is a cached live lookup.

class TotalDepth.common.lookup_mnemonic.**Parameter**(*code*, *description*, *unit_quantity*, *property*, *related_products*)

property code
Alias for field number 0

property description
Alias for field number 1

property unit_quantity
Alias for field number 2

property property
Alias for field number 3

property related_products
Alias for field number 4

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls, code: str, description: str, unit_quantity: str, property: str, related_products: Tuple[TotalDepth.common.lookup_mnemonic.ProductDescription]*)
Create new instance of Parameter(code, description, unit_quantity, property, related_products)

__repr__()
Return a nicely formatted representation string

TotalDepth.common.lookup_mnemonic.**slb_parameter**
Returns the Parameter corresponding to the name. This is a cached live lookup.

class TotalDepth.common.lookup_mnemonic.**ChannelDescription**(*channel*, *description*)

property channel
Alias for field number 0

property description
Alias for field number 1

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *channel: str*, *description: str*)
Create new instance of ChannelDescription(channel, description)

__repr__()
Return a nicely formatted representation string

class TotalDepth.common.lookup_mnemonic.**ParameterDescription**(*parameter*, *description*)

property parameter
Alias for field number 0

property description
Alias for field number 1

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *parameter: str*, *description: str*)
Create new instance of ParameterDescription(parameter, description)

__repr__()
Return a nicely formatted representation string

class TotalDepth.common.lookup_mnemonic.**LoggingTool**(*code*, *technology*, *discipline*, *method*, *description*, *related_channels*, *related_parameters*)

property code
Alias for field number 0

property technology
Alias for field number 1

property discipline
Alias for field number 2

property method
Alias for field number 3

property description
Alias for field number 4

property related_channels
Alias for field number 5

property related_parameters
Alias for field number 6

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls, code: str, technology: str, discipline: str, method: str, description: str, related_channels: Tuple[TotalDepth.common.lookup_mnemonic.ChannelDescription], related_parameters: Tuple[TotalDepth.common.lookup_mnemonic.ParameterDescription]*)
Create new instance of LoggingTool(code, technology, discipline, method, description, related_channels, related_parameters)

__repr__()
Return a nicely formatted representation string

TotalDepth.common.lookup_mnemonic.**slb_logging_tool**
Logging Tools

Example:

```
https://www.apps.slb.com/cmd/ToolItem.aspx?code=HDT
```

In the table 'main_DetailsView1' a ToolItem has the attributes: Code, Technology, Discipline, Method, Description as a key/value table.

Example:

```
Code        HDT
Technology  Dipmeter
Discipline  Geology
Method      WIRELINE
Description High Resolution Dipmeter Tool
```

They also have 'Related Channels' in the 'main_GridView1' a table a list of key/values:

```
<th scope="col">Channel</th><th scope="col">Description</th>
```

They also have 'Related Parameters' in the 'main_GridView2' a table a list of key/values:

```
<th scope="col">Parameter</th><th scope="col">Description</th>
```

TotalDepth.common.process (Monitoring a Process's CPU and Memory)

A HOWTO is here *Process Monitoring with TotalDepth.common.process*

Logs process information, such as memory usage, to a log as JSON. Example with ('memory_info', 'cpu_times'):

```
(Thread-7   ) ProcessLoggingThread JSON: {"memory_info": {"rss": 145448960, "vms":␣
↪4542902272, "pfaults": 37618, "pageins": 0}, "cpu_times": {"user": 0.28422032,
↪"system": 0.099182912, "children_user": 0.0, "children_system": 0.0}}
```

There are several DoF here:

- Logging interval in seconds. Or by poke()?
- Logging level, DEBUG, INFO etc.
- Logging verbosity, for example just memory? Or everything about the process (self._process.as_dict())

Also need to add a log parser to, well what?

TotalDepth.common.process.**LOGGER_PREFIX = 'ProcessLoggingThread-JSON'**
Unique string in the log line

TotalDepth.common.process.**RE_LOG_LINE = re.compile('^.+?ProcessLoggingThread-JSON(-START|-S**
Regex for the unique string in the log line

TotalDepth.common.process.**DATETIME_NOW_FORMAT = '%Y-%m-%d %H:%M:%S.%f'**
Regex for timestam, matches '2019-06-07 11:57:58.390921'

TotalDepth.common.process.**KEY_TIMESTAMP = 'timestamp'**
The JSON key that is the timestamp

TotalDepth.common.process.**KEY_ELAPSED_TIME = 'elapsed_time'**
The JSON key that is elapsed (wall clock) time in seconds. This is `time.time() - self._process.create_time()`

TotalDepth.common.process.**KEY_LABEL = 'label'**
The JSON key that is the label

TotalDepth.common.process.**KEY_PROCESS_ID = 'pid'**
The JSON key that is the process ID

TotalDepth.common.process.**PSUTIL_PROCESS_AS_DICT_KEYS = ['cmdline', 'connections', 'cpu_per**
psutil.Process().as_dict() has the following keys:

TotalDepth.common.process.**GNUPLOT_PLT = '\nset grid\nset title "Memory and CPU Usage." font**
Usage: GNUPLOT_PLT.format(name=dat_file_name)

TotalDepth.common.process.**parse_timestamp**(*s: str*) → datetime.datetime
Read a string such as '2019-06-07 11:57:58.390921' and return a datetime.

TotalDepth.common.process.**extract_json**(*istream: TextIO*) → List[Dict[str, Any]]
Reads a log file and returns the JSON as a list of dicts. Non-matching lines are ignored.

TotalDepth.common.process.**extract_labels_from_json**(*json_data: List[Dict[str, Any]]*) → List[Dict[str, Any]]
Returns a list of dicts of JSON data where 'label' is a key'.

TotalDepth.common.process.**extract_json_as_table**(*json_data: List[Dict[str, Any]]*) → Tuple[Dict[int, List[List[str]]], Dict[int, float], Dict[int, float], Dict[int, float], Dict[int, float]]
Create a table from JSON suitable for a Gnuplot `.dat` file.

Returns:

- { process_id : [rows of data, . . .], . . . }
- { process_id : t min, . . . }
- { process_id : t max, . . . }
- { process_id : RSS min, . . . }
- { process_id : RSS max, . . . }

A row of data is:

time, RSS, PageFaults, User, Mean CPU, Insantanous CPU, Timestamp, PID, Label

TotalDepth.common.process.**invoke_gnuplot**(*log_path: str*, *gnuplot_dir: str*) → int
Reads a log file, extracts the data, writes it out to gnuplot_dir and invokes gnuplot on it.

TotalDepth.common.process.**add_message_to_queue**(*msg: str*) → None
Adds a message onto the queue.

class TotalDepth.common.process.**ProcessLoggingThread**(*group=None*, *target=None*, *name=None*, *args=()*, *kwargs=None*, ***, *daemon=None*)
Thread that regularly logs out process parameters.

__init__(*group=None*, *target=None*, *name=None*, *args=()*, *kwargs=None*, ***, *daemon=None*)
Constructor. args[0], or interval=... must be the reporting interval in seconds, default 1.0. args[1], or log_level=... must be the log level to report with, default logging.INFO.

run() → None
thread.run(). Write to log then sleep.

join(**args*, ***kwargs*)
thread.join(). Write to log last time.

TotalDepth.common.process.**log_process**(**args*, ***kwargs*)
Context manager to log process data at regular intervals.

TotalDepth.common.process.**add_process_logger_to_argument_parser**(*parser: argparse.ArgumentParser*) → None
Add a `--log-process` option to the argument parser.

TotalDepth.common.process.**main**() → int
Main CLI entry point. For testing.

TotalDepth.common.Rle (Run Length Encoding Data Compression)

Code for Run Length Encoding.

class TotalDepth.common.Rle.**RLEItem**(*datum: Union[int, float]*)
Class that represents a single entry in a Run Length Encoding set. v - The datum value.

__init__(*datum: Union[int, float]*)
Initialize self. See help(type(self)) for accurate signature.

__str__() → str
String representation.

__len__() → int
Total number of values.

add(*v: Union[int, float]*) → bool
Returns True if v has been absorbed in this entry. False means a new entry is required.

values() → Sequence[Union[int, float]]
Generates all values.

value(*i*) → Tuple[int, Union[int, float, None]]
Returns a particular value.

range() → range
Returns a range object that has (start, stop, step).

last()
Returns the last value.

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

class TotalDepth.common.Rle.**RLE**(*theFunc=None*)
Class that represents Run Length Encoding.

theFunc - optional unary function to convert all values with.

__init__(*theFunc=None*)
Constructor, optionally takes a unary function to convert all values with.

__str__()
Return str(self).

__len__()
The number of RLEItem(s).

__getitem__(*key*)
Returns a RLEItem.

num_values()
Total number of record values.

add(*v*)
Adds a value to this RLE object.

values()
Generates all values entered.

value(*i*)
Indexing; this returns the i'th value added.

ranges()
Returns a list of range() objects.

first()
Returns the first value or None if no values added.

last()
Returns the last value or None if no values added.

largest_le(*value: Union[int, float]*) → Union[int, float]
Return the largest value less than or equal to the given value.

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

TotalDepth.common.Rle.**create_rle**(*values: Iterable*, *fn: Callable = None*) → TotalDepth.common.Rle.RLE
Create a RLE object from an iterable.

TotalDepth.common.Slice (Data Slicing)

class TotalDepth.common.Slice.**SliceABC**

abstract first(*length: int*) → int
The index of the first element of a sequence of length.

abstract last(*length: int*) → int
The index of the last element of a sequence of length.

abstract step(*length: int*) → int
The sequence of length step.

abstract count(*length: int*) → int
Returns the number of values that will result if the slice is applied to a sequence of given length.

abstract gen_indices(*length: int*) → range
Generates the indices for the sequence of the given length.

abstract indices(*length: int*) → List[int]
Returns a fully composed list of indices for the sequence of the given length.

abstract __eq__(*other*) → bool
Mostly used for testing.

abstract long_str(*length: int*) → str
Return a long string.

abstract __str__() → str
String representation.

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

class TotalDepth.common.Slice.**Slice**(*start: Union[None, int] = None*, *stop: Union[None, int] = None*, *step: Union[None, int] = None*)
Class that wraps a builtin slice object for integers and provides some useful APIs. NOTE: The builtin slice object can take non-integer values but raises later, for example:

```
slice(1, 4, 2.0).indices(45)
```

__init__(*start: Union[None, int] = None*, *stop: Union[None, int] = None*, *step: Union[None, int] = None*)
Initialize self. See help(type(self)) for accurate signature.

first(*length: int*) → int
The index of the first element of a sequence of length.

last(*length: int*) → int
The index of the last element of a sequence of length.

step(*length: int*) → int
The sequence of length step.

count(*length: int*) → int
Returns the number of values that will result if the slice is applied to a sequence of given length.

gen_indices(*length: int*) → range
Generates the indices for the sequence of the given length.

indices(*length: int*) → List[int]
Returns a fully composed list of indices for the sequence of the given length.

__eq__(*other*) → bool
Mostly used for testing.

long_str(*length: int*) → str
Return a long string.

__str__() → str
String representation.

class TotalDepth.common.Slice.**Sample**(*sample_size: int*)
This has the same API as Slice but takes a single integer.

NOTE: This may not produce a regular sequence. For example sampling 7 items out of a 12 element list gives the indices [0, 1, 3, 5, 6, 8, 10]

__init__(*sample_size: int*)
Initialize self. See help(type(self)) for accurate signature.

first(*length*) → int
The index of the first element of a sequence of length.

last(*length*) → int
The index of the last element of a sequence of length.

step(*length*) → int
The sequence of length step.

count(*length: int*) → int
Returns the number of values that will result if the slice is applied to a sequence of given length.

gen_indices(*length: int*) → range
Generates the indices for the sequence of the given length.

indices(*length: int*) → List[int]
Returns a fully composed list of indices for the sequence of the given length.

__eq__(*other*) → bool
Mostly used for testing.

long_str(*length: int*) → str
Long descriptive string.

__str__() → str
String representation.

TotalDepth.common.Slice.**create_slice_or_sample**(*slice_string: str*) → Union[TotalDepth.common.Slice.Slice, TotalDepth.common.Slice.Sample]
Returns a Slice object from a string such as: ‘’, ‘None,72’, ‘None,72,14’

TotalDepth.common.statistics

class TotalDepth.common.statistics.**LengthDict**
Provides statistics about a summary of lengths such as file lengths or logical data lengths.

__init__()
Initialize self. See help(type(self)) for accurate signature.

__len__() → int
Number of distinct length entries.

__getitem__(*item*) → int
Returns the count of a particular length. Does not write to the dict.

property min
Minimum length.

property max
Maximum length.

property count
Return the total number of entries.

property zero_count
Return the number of length zero.

keys() → KeysView[int]
Return the keys (lengths seen).

add(*length: int*)
Add a length to the summary.

reduced_power_2() → DefaultDict[int, int]
Return a histogram with keys reduced to power of 2.

histogram_power_of_2(*width: int = 40*, *bar_char: str = '+'*) → List[str]
Return the count of zero and a histogram of strings of the lengths to power of 2.

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

TotalDepth.common.units

This provides Unit conversion information from lookup sources.

The primary source is Schlumberger's Oilfield Services Data Dictionary (OSDD): https://www.apps.slb.com/cmd/units.aspx

The fallback, secondary source, is from our static snapshot of that page which lives in `src/TotalDepth/common/data/osdd_units.json`

When running tests with `--runslow` the `tests.integration.common.test_units.test_slb_units_write_to_json` test will re-populate that static data file.

This included currencies that all have zero offset and unit scale. Currencies: https://en.wikipedia.org/wiki/ISO_4217 ISO 4217 official currency codes in XML: https://www.currency-iso.org/dam/downloads/lists/list_one.xml

exception TotalDepth.common.units.**ExceptionUnits**
Base class exception for this module.

exception TotalDepth.common.units.**ExceptionUnitsLookup**
Raised if the unit lookup fails.

exception TotalDepth.common.units.**ExceptionUnitsDimension**
Raised if two units are of different dimensions.

class TotalDepth.common.units.**Unit**
Represents one row in the table at https://www.apps.slb.com/cmd/units.aspx

Examples:

```
Code    Name                Standard Form   Dimension   Scale               Offset
DEGC    'degree celsius'    degC            Temperature 1                   -273.
↪15
DEGF    'degree fahrenheit' degF            Temperature 0.555555555555556   -459.
↪67
DEGK    'kelvin'            K               Temperature 1                   0
DEGR    'degree rankine'    degR            Temperature 0.555555555555556   0
```

There will also be an entry for RP66V1 files:

```
degF    'degree fahrenheit' "5/9 degC +32"  Temperature 0.555555555555556   -459.
↪67
```

So conversion from, say DEGC to DEGF is:

```
((value - DEGC.offset) * DEGC.scale) / DEGF.scale + DEGF.offset

((0.0 - -273.15) * 1.0) / 0.555555555555556 + -459.67 == 32.0
```

property code
Alias for field number 0

property name
Alias for field number 1

property standard_form
Alias for field number 2

property dimension
Alias for field number 3

property scale
Alias for field number 4

property offset
Alias for field number 5

property is_primary
True if this is looks like a primary unit.

has_offset() → bool
True if this has an offset, for example DEGC. False otherwise, for example metres.

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *code: str*, *name: str*, *standard_form: str*, *dimension: str*, *scale: float*, *offset: float*)
Create new instance of Unit(code, name, standard_form, dimension, scale, offset)

__repr__()
Return a nicely formatted representation string

TotalDepth.common.units.**osdd_data_file_path**() → str
Path to our static snapshot of the OSDD units page.

TotalDepth.common.units.**read_osdd_static_data**() → Dict[str, TotalDepth.common.units.Unit]
Read our static snapshot of the OSDD units page.

TotalDepth.common.units.**slb_load_units**()
Eagerly load the units into the cache.

TotalDepth.common.units.**has_slb_units**(*unit_code: str*) → bool
Returns True if the Schlumberger Unit exists.

TotalDepth.common.units.**slb_units**(*unit: str*) → TotalDepth.common.units.Unit
Returns the Schlumberger Unit corresponding to the unit code.

TotalDepth.common.units.**has_slb_standard_form**(*standard_form: str*) → bool
Returns True if an entry for the standard form exists.

TotalDepth.common.units.**slb_standard_form_to_unit_code**(*standard_form: str*) → List[TotalDepth.common.units.Unit]
Returns the unit(s) corresponding to the standard form. Example given 'degC' this returns the Units corresponding to ['DEGC', 'deg C', 'oC'].

TotalDepth.common.units.**same_dimension**(*a: TotalDepth.common.units.Unit*, *b: TotalDepth.common.units.Unit*) → bool
Returns True if both units have the same dimension.

TotalDepth.common.units.**convert**(*value: float*, *unit_from: TotalDepth.common.units.Unit*, *unit_to: TotalDepth.common.units.Unit*) → float
Converts a value from one unit to another.

Examples:

```
Code    Name                Standard Form   Dimension   Scale                Offset
DEGC    'degree celsius'    degC            Temperature 1                    -273.
↪15
DEGF    'degree fahrenheit' degF            Temperature 0.555555555555556    -459.
↪67
```

So conversion from, say DEGC to DEGF is:

```
((value - DEGC.offset) * DEGC.scale) / DEGF.scale + DEGF.offset

((0.0 - -273.15) * 1.0) / 0.555555555555556 + -459.67 == 32.0
```

TotalDepth.common.units.**convert_function**(*unit_from: TotalDepth.common.units.Unit*, *unit_to: TotalDepth.common.units.Unit*) → Callable
Return a partial function to convert from one units to another.

TotalDepth.common.units.**convert_array**(*array: numpy.ndarray*, *unit_from: TotalDepth.common.units.Unit*, *unit_to: TotalDepth.common.units.Unit*) → numpy.ndarray
Convert an array of values.

TotalDepth.common.units.**convert_array_inplace**(*array: numpy.ndarray*, *unit_from: TotalDepth.common.units.Unit*, *unit_to: TotalDepth.common.units.Unit*) → None
Convert an array of values in-place.

TotalDepth.common.xml (Importing XML Libraries)

Imports the best XML libraries as etree. Modified from: https://lxml.de/tutorial.html

TotalDepth.common.xxd

TotalDepth.common.xxd.**xxd**(*by: bytes*, *columns: int = 16*, *uppercase: bool = False*, *ebcdic: bool = False*, *binary: bool = False*, *length: int = 0*, *offset: int = 0*, *seek: int = 0*) → str

Returns an xxd style string of the bytes. For example:

```
0084 8000 8400 2647 3546 3239 2020 2020 2020 ......&G5F29
```

`columns` - Number of octets in each row.

`uppercase` - use upper case hex letters.

`ebcdic` - show characters in EBCDIC. Default ASCII.

`binary` - binary digit dump. Default hex.

`length` - stop after <length> octets.

`offset` - add <offset> to the displayed file position.

`seek` - start at <seek> bytes infile offset.

1.9.9 TotalDepth.util - Utility Package Reference

Contents:

TotalDepth.util.CopyBinFiles

Copies files of a specific type from one file tree to another.

TotalDepth.util.CopyBinFiles.**copy_files**(*path_in: str*, *path_out: str*, *binary_file_types: Set[str]*, *move: bool*, *nervous: bool*) → List[str]

Copies binary files from path_in to path_out.

If move is True the file is moved, if False the file is copied. Returns a list of destination paths.

TotalDepth.util.CopyBinFiles.**main**() → int

Main entry point. Copies of moves particular file types from one tree to another

TotalDepth.util.DictTree (Tree-like Dictionary)

A dictionary that takes a list of hashables as a key and behaves like a tree.

exception TotalDepth.util.DictTree.**ExceptionDictTree**

Exception when handling a DictTree object.

exception TotalDepth.util.DictTree.**ExceptionDictTreeHtmlTable**

Exception when handling a DictTreeHtmlTable object.

class TotalDepth.util.DictTree.**DictTree**(*value_iterable=None*)

A dictionary that takes a list of hashables as a key and behaves like a tree. A node can have multiple values represented as a set or list.

__init__(*value_iterable=None*)
Initialize self. See help(type(self)) for accurate signature.

add(*key: Sequence[Hashable], value: Any*) → None
Add a key/value. k is a list of hashables.

remove(*key: Sequence[Hashable], value: Any = None*) → None
Remove a key/value.

value(*key: Sequence[Hashable]*) → Optional[Any]
Value corresponding to a key or None. k is a list of hashables.

values() → List[Any]
Returns a list of all values.

keys() → List[Hashable]
Return a list of keys where each key is a list of hashables.

items() → Sequence[Tuple[Sequence[Hashable], Any]]
Yields a sequence of key, value pairs.

__len__() → int
Returns the number of keys.

depth() → int
Returns the maximum tree depth as an integer.

indented_string() → str
Returns an indented string.

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

class TotalDepth.util.DictTree.**DictTreeTableEvent**
POD class that contains the data needed for a HTML table entry. branch - the data route to this node. node - the columns of the table entry. row_span - the HTML rowspan attribute for the <td>. col_span - the HTML colspan attribute for the <td>.

property branch
Alias for field number 0

property node
Alias for field number 1

property row_span
Alias for field number 2

property col_span
Alias for field number 3

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls, branch: List[Any], node: Any, row_span: int, col_span: int*)
Create new instance of DictTreeTableEvent(branch, node, row_span, col_span)

__repr__()
Return a nicely formatted representation string

class TotalDepth.util.DictTree.**DictTreeHtmlTable**(**args*, ***kwargs*)
A sub-class of DictTree that helps writing HTML row/col span tables Suppose we have a tree like this:

```
                        |- AAA
                        |
                |- AA --|- AAB
                |       |
                |       |- AAC
        |- A ---|
Root ---|       |- AB
        |       |
        |       |- AC ---- ACA
        |
        |- B
        |
        |- C ---- CA ---- CAA
```

And we want to represent the tree like this when laid out as an HTML table:

```
|-----------------------|
| A     | AA    | AAA   |
|       |       |-------|
|       |       | AAB   |
|       |       |-------|
|       |       | AAC   |
|       |---------------|
|       | AB            |
|       |---------------|
|       | AC    | ACA   |
|-----------------------|
| B                     |
|-----------------------|
| C     | CA    | CAA   |
|-----------------------|
```

In this example the tree is loaded branch by branch thus:

```
myTree = DictTreeHtmlTable()
myTree.add(('A', 'AA', 'AAA'), None)
myTree.add(('A', 'AA', 'AAB'), None)
myTree.add(('A', 'AA', 'AAC'), None)
myTree.add(('A', 'AB',), None)
myTree.add(('A', 'AC', 'ACA'), None)
myTree.add(('B',), None)
myTree.add(('C', 'CA', 'CAA'), None)
```

The HTML code generator can be used like this:

```
# Write: <table border="2" width="100%">
with XmlWrite.Element(xhtml_stream, 'table', {}):
    for event in myTree.genColRowEvents():
        if event == myTree.ROW_OPEN:
            # Write out the '<tr>' element
            xhtml_stream.startElement('tr', {})
        elif event == myTree.ROW_CLOSE:
            # Write out the '</tr>' element
            xhtml_stream.endElement('tr')
        else:
            # Write '<td rowspan="..." colspan="...">...</td>' % (r, c, v)
            with XmlWrite.Element(xhtml_stream, 'td', event.html_attrs()):
```

(continues on next page)

(continued from previous page)

```
                xhtml_stream.characters(str(event.node))
# Write: </table>
```

And the HTML will look like this:

```
<table border="2" width="100%">
    <tr valign="top">
        <td rowspan="5">A</td>
        <td rowspan="3">AA</td>
        <td>AAA</td>
    </tr>
    <tr valign="top">
        <td>AAB</td>
    </tr>
    <tr valign="top">
        <td>AAC</td>
    </tr>
    <tr valign="top">
        <td colspan="2">AB</td>
    </tr>
    <tr valign="top">
        <td>AC</td>
        <td>ACA</td>
    </tr>
    <tr valign="top">
        <td colspan="3">B</td>
    </tr>
    <tr valign="top">
        <td>C</td>
        <td>CA</td>
        <td>CAA</td>
    </tr>
</table>
```

ROW_OPEN = DictTreeTableEvent(branch=[], node=None, row_span=0, col_span=0)
HTML table event: open row with <tr ...>

ROW_CLOSE = DictTreeTableEvent(branch=[], node=None, row_span=-1, col_span=-1)
HTML table event: close row with </tr>

__init__(**args*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

is_row_open(*event: TotalDepth.util.DictTree.DictTreeTableEvent*) → bool
Return True if the event I have generated is a ROW_OPEN event.

is_row_close(*event: TotalDepth.util.DictTree.DictTreeTableEvent*) → bool
Return True if the event I have generated is a ROW_CLOSE event.

add(*key: Sequence[Hashable], value: Any*) → None
Add a key/value.

remove(*key: Sequence[Hashable], value: Any = None*)
Remove a key/value.

set_row_column_span() → None
Top level call that sets colspan and rowspan attributes.

gen_row_column_events() → Sequence[TotalDepth.util.DictTree.DictTreeTableEvent]

Returns a set of events that are quadruples. (key_branch, value, rowspan_int, colspan_int) The branch is a list of keys the from the branch of the tree. The rowspan and colspan are both integers. At the start of the a <tr> there will be a ROW_OPEN and at row end (</tr> a ROW_CLOSE will be yielded

gen_row_column_events_from_branch(*key_branch: List[Hashable]*) → Sequence[TotalDepth.util.DictTree.DictTreeTableEvent]
Yields a set of events that are a tuple of quadruples. (key_branch, value, rowspan_integer, colspan_integer) For example: (['a', 'b'], 'c', 3, 7) At the start of the a <tr> there will be a ROW_OPEN and at row end (</tr>) a ROW_CLOSE will be yielded

depth_from_branch(*key_branch: Sequence[Hashable]*) → int
Finds the remainder of depth from the branch.

walk_row_col_span() → str
Return the internal tree as a string.

TotalDepth.util.DirWalk (Directory Walking)

Provides various ways of walking a directory tree

Created on Jun 9, 2011

class TotalDepth.util.DirWalk.**FileInOut**(*filePathIn*, *filePathOut*)

property filePathIn
Alias for field number 0

property filePathOut
Alias for field number 1

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *filePathIn: str*, *filePathOut: str*)
Create new instance of FileInOut(filePathIn, filePathOut)

__repr__()
Return a nicely formatted representation string

exception TotalDepth.util.DirWalk.**ExceptionDirWalk**
Exception class for this module.

TotalDepth.util.DirWalk.**gen_big_first**(*directory*)
Generator that yields the biggest files (name not path) first. This is fairly simple in that it it only looks the current directory not only sub-directories. Useful for multiprocessing.

TotalDepth.util.DirWalk.**dirWalk**(*theIn: str*, *theOut: str = ''*, *theFnMatch: str = ''*, *recursive: bool = False*, *bigFirst: bool = False*) → Sequence[TotalDepth.util.DirWalk.FileInOut]
Walks a directory tree generating file paths as FileInOut(in, out) objects.

theIn - The input directory.

theOut - The output directory. If an empty string the out path will be an empty string. NOTE: This does not create the output directory structure, it is up to the caller to do that.

theFnMatch - A glob like match pattern for file names (not tested for directory names).

recursive - Boolean to recurse or not.

bigFirst - If True then the largest files in directory are given first. If False it is alphabetical.

TotalDepth.util.EBCDIC

TotalDepth.util.ExecTimer (Timing Code Execution)

Has classes for timing execution

exception TotalDepth.util.ExecTimer.**ExceptionExecTimer**
Specialisation of exception for this module.

class TotalDepth.util.ExecTimer.**Timer**(*description: str*)
Records the timing of a single event.

__init__(*description: str*)
Initialize self. See help(type(self)) for accurate signature.

stop(*work_done: int = 0*) → None
Stop the timer and record how much work was done.

add_work_done(*work_done: int*) → None
Adds work done.

property elapsed_perf_counter
Executions time in seconds as seen by a wall clock.

property elapsed_wall_clock
Executions time in seconds as seen by a wall clock.

__str__() → str
Return str(self).

property ms_mb
Return the work rate in ms/MB.

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

class TotalDepth.util.ExecTimer.**TimerList**
Maintains a list of execution time objects

__init__()
Constructor

__len__() → int
Number of task timers.

add_timer(*description: str*) → None
Load a new task timer starting right now.

property timer
The current timer.

property has_active_timer
True if there is a running timer, False if there are either no timers or the latest timer is halted.

stop(*work_done=0*) → None
Stop current timer.

__str__()
Return str(self).

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

TotalDepth.util.FileBuffer (Look-ahead File Buffer)

Provides a 'look ahead' file buffer where the caller can inspect bytes ahead of the current position.

Created on Oct 26, 2011

exception TotalDepth.util.FileBuffer.**ExceptionFileBuffer**
Specialisation of Exception for the FileBuffer module.

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

exception TotalDepth.util.FileBuffer.**ExceptionFileBufferEOF**
Specialisation of Exception for the FileBuffer EOF.

class TotalDepth.util.FileBuffer.**FileBuffer**(*f*)
Provides a buffer interface to a file where the user can look ahead any distance from the current position.

__init__(*f*)
Initialize self. See help(type(self)) for accurate signature.

tell()
Current file position.

step()
Increment the file position by one byte, returns the byte just read.

__getitem__(*i*)
Get an arbitrary byte or slice.

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

TotalDepth.util.FileStatus

Created on 28 Jun 2010

exception TotalDepth.util.FileStatus.**ExceptionFileStatus**

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

class TotalDepth.util.FileStatus.**FileInfo**(*thePath*)
Obtains the status of a file, hash SLOC etc.

__init__(*thePath*)
Initialize self. See help(type(self)) for accurate signature.

writeHeader(*theS=<colorama.ansitowin32.StreamWrapper object>*)
Write the summary header.

write(*theS=<colorama.ansitowin32.StreamWrapper object>*, *incHash=True*)
Write the summary.

property sloc
SLOC

property size
Size in bytes.

property count
Number of files, 0 or 1.

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

class TotalDepth.util.FileStatus.**FileInfoSet**(*thePath*, *glob=None*, *isRecursive=False*, *isTestOnly=False*)
Represents a set of files from a directory tree.

__init__(*thePath*, *glob=None*, *isRecursive=False*, *isTestOnly=False*)
Initialize self. See help(type(self)) for accurate signature.

processPath(*theP*, *glob=None*, *isRecursive=False*, *isTestOnly=False*)
Process a file or directory.

processDir(*theDir*, *glob*, *isRecursive*, *isTestOnly*)
Read a directory and return a map of {path : class FileInfo, ... }

write(*theS=<colorama.ansitowin32.StreamWrapper object>*)
Write out the summary.

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

TotalDepth.util.FileStatus.**main**()
Process a path and write out the file summary.

TotalDepth.util.Histogram

Produces histograms.

Created on Nov 29, 2011

class TotalDepth.util.Histogram.**Histogram**(*pre_load=None*)
A histogram class.

__init__(*pre_load=None*)
Initialize self. See help(type(self)) for accurate signature.

add(*x*, *count=1*)
Increments the count of value x by count (default 1).

__getitem__(*x*)
Returns the current count of x.

strRep(*width=75*, *chr='+'*, *valTitle=''*, *inclCount=False*)
Returns a string representation of the histogram in ASCII.

width - The maximum width to use.

chr - The character to use in the plot.

valTitle - The title to use for values.

inclCount - Include the actual count for each value?

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

Testing

Tests are in `test/TestHistogram.py`

Running Tests

```
$ python3 test/testHistogram.py
```

Test Coverage

```
$ coverage run test/testHistogram.py
...
$ coverage report -m
```

Examples

```
from TotalDepth.util import Histogram

myH = Histogram.Histogram()
for x in range(1, 12):
    self._hist.add(x, x)
print(self._hist.strRep())
# Prints """ 1 | ++++++
 2 | ++++++++++++
 3 | ++++++++++++++++++
 4 | ++++++++++++++++++++++++
 5 | ++++++++++++++++++++++++++++++++
 6 | ++++++++++++++++++++++++++++++++++++++
 7 | ++++++++++++++++++++++++++++++++++++++++++++++
 8 | ++++++++++++++++++++++++++++++++++++++++++++++++++++
 9 | ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
10 | ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
11 | ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++"""
```

TotalDepth.util.HtmlUtils (HTML Utilities)

HTML utility functions.

`TotalDepth.util.HtmlUtils.`**`retHtmlFileName`**(*thePath*)
Creates a unique, short, human readable file name base on the input file path.

`TotalDepth.util.HtmlUtils.`**`retHtmlFileLink`**(*theSrcPath*, *theLineNum*)
Returns a link to a file/line.

`TotalDepth.util.HtmlUtils.`**`writeHtmlFileLink`**(*theS*, *theSrcPath*, *theLineNum*, *theText=''*, *theClass=None*)
Writes a link to another HTML file that represents source code. theS is an XHTML stream. theSrcPath is the path of the original source, which will be encoded with retHtmlFileName(). theLineNum is an integer line number in the target. theText is optional navigation text. theClass is optional CSS class for the navigation text.

TotalDepth.util.HtmlUtils.**writeHtmlFileAnchor**(*theS*, *theLineNum*, *theText=''*, *theClass=None*)

Write an anchor to the stream.

TotalDepth.util.HtmlUtils.**pathSplit**(*p*)

Split a path into its components.

TotalDepth.util.HtmlUtils.**writeFileListAsTable**(*theS*, *theFileLinkS*, *tableAttrs*, *includeKeyTail*)

Writes a list of file names as an HTML table looking like a directory structure. theFileLinkS is a list of pairs (file_path, href). The navigation text in the cell will be the basename of the file_path.

TotalDepth.util.HtmlUtils.**writeFileListTrippleAsTable**(*theS*, *theFileLinkS*, *tableAttrs*, *includeKeyTail*)

Writes a list of file names as an HTML table looking like a directory structure. theFileLinkS is a list of triples (file_name, href, nav_text).

TotalDepth.util.HtmlUtils.**writeDictTreeAsTable**(*theS*, *theDt*, *tableAttrs*, *includeKeyTail*)

Writes a DictTreeHtmlTable object as a table, for example as a directory structure.

The key list in the DictTreeHtmlTable object is the path to the file i.e. os.path.abspath(p).split(os.sep) and the value is expected to be a pair of (link, nav_text) or None.

TotalDepth.util.HtmlUtils.**writeFilePathsAsTable**(*valueType*, *theS*, *theKvS*, *tableStyle*, *fnTd*)

Writes file paths as a table, for example as a directory structure.

valueType - The type of the value: None | 'list' | 'set'

theKvS - A list of pairs (file_path, value).

tableStyle - The style used for the table.

fnTd - A callback function that is executed for a <td> element when there is a non-None value. This is called with the following arguments:

- theS - The HTML stream.
- attrs - A map of attrs that include the rowspan/colspan for the <td>
- k - The key as a list of path components.
- v - The value given by the caller.

TotalDepth.util.PatternSearch (Search for Patterns in Binary Files)

Searches for runs of data in binary files.

Created on Oct 24, 2011

@author: paulross

TotalDepth.util.PatternSearch.**reportAll**(*theFile*, *theS=<colorama.ansitowin32.StreamWrapper object>*)

Print a histogram of byte values.

TotalDepth.util.PatternSearch.**report0x80**(*theFile*, *theS=<colorama.ansitowin32.StreamWrapper object>*, *theMin=1*, *showTell=False*)

Print a for 0x80.

TotalDepth.util.PatternSearch.**reportIDENT**(*theFile*, *theS=<colorama.ansitowin32.StreamWrapper object>*, *theMin=2*, *showTell=False*)

IDENT rep code: The valid character subset consists of null (0) plus the codes 33 0x21 (!) to 96 0x60 (`) and

from 123 0x7b ({) to 126 0x7e (~) inclusive. This excludes all control characters, all "white space", and the lower-case alphabet.

TotalDepth.util.RemoveDupeFiles (Remove Duplicate Files)

Scans a directory and removes duplicate files based on their SHA.

`TotalDepth.util.RemoveDupeFiles.`**`remove_dupes`**(*path: str*, *nervous: bool*) → Tuple[int, int]
Scans a directory tree removing duplicate files detected by their SHA512.

`TotalDepth.util.RemoveDupeFiles.`**`main`**() → int
Main CLI entry point.

TotalDepth.util.XmlWrite (XML/XHTML/HTML Writer)

Writes XML and XHTML.

exception `TotalDepth.util.XmlWrite.`**`ExceptionXml`**
Exception specialisation for the XML writer.

exception `TotalDepth.util.XmlWrite.`**`ExceptionXmlEndElement`**
Exception specialisation for end of element.

`TotalDepth.util.XmlWrite.`**`encodeString`**(*theS*, *theCharPrefix='_'*)
Returns a string that is the argument encoded. RFC3548:

Table 1: The Base 64 Alphabet

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w	(pad)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

See section 3 of : http://www.faqs.org/rfcs/rfc3548.html

`TotalDepth.util.XmlWrite.`**`decodeString`**(*theS*)
Returns a string that is the argument decoded. May raise a TypeError.

`TotalDepth.util.XmlWrite.`**`nameFromString`**(*theStr*)
Returns a name from a string.

See http://www.w3.org/TR/1999/REC-html401-19991224/types.html#type-cdata

"ID and NAME tokens must begin with a letter ([A-Za-z]) and may be followed by any number of letters, digits ([0-9]), hyphens ("-"), underscores ("_"), colons (":"), and periods (".").

This also works for in namespaces as ':' is not used in the encoding.

class TotalDepth.util.XmlWrite.**XmlStream**(*theFout*, *theEnc='utf-8'*, *theDtdLocal=None*, *theId=0*)

Creates and maintains an XML output stream.

__init__(*theFout*, *theEnc='utf-8'*, *theDtdLocal=None*, *theId=0*)

Initialise with a writable file like object or a file path.

theFout - The file-like object or a string. If the latter it will be closed on __exit__.

theEnc - The encoding to be used.

theDtdLocal - Any local DTD as a string.

id - An integer value to use as an ID string.

property id

A unique ID in this stream. The ID is incremented on each call.

xmlSpacePreserve()

Suspends indentation for this element and its descendants.

characters(*theString*)

Encodes the string and writes it to the output.

literal(*theString*)

Writes theString to the output without encoding.

comment(*theS*)

Writes a comment to the output stream.

pI(*theS*)

Writes a Processing Instruction to the output stream.

endElement(*name*)

Ends an element.

writeECMAScript(*theScript*)

Writes the ECMA script.

Example:

```
<script type="text/ecmascript">
//<![CDATA[
...
// ]]>
</script>
```

__enter__()

Context manager support.

__exit__(*exc_type*, *exc_value*, *traceback*)

Context manager support.

__weakref__

list of weak references to the object (if defined)

class TotalDepth.util.XmlWrite.**XhtmlStream**(*theFout*, *theEnc='utf-8'*, *theDtdLocal=None*, *theId=0*)

__enter__()

Context manager support.

charactersWithBr(*sIn*)
Writes the string replacing any n characters with
 elements.

class TotalDepth.util.XmlWrite.**Element**(*theXmlStream*, *theElemName*, *theAttrs=None*)
Represents an element in a markup stream.

__init__(*theXmlStream*, *theElemName*, *theAttrs=None*)
Initialize self. See help(type(self)) for accurate signature.

__enter__()
Context manager support.

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

__exit__(*excType*, *excValue*, *tb*)
Context manager support.

TotalDepth.util.archive (Archive Management)

Extract summary data from archives of log files.

This is a bit hacked together to help create a good archive of test data. It is not production code.

class TotalDepth.util.archive.**FileBase**(*path: str*)
Base class to represent a file, either on-disc or a ZIP file.

__init__(*path: str*)
Initialize self. See help(type(self)) for accurate signature.

__str__()
Return str(self).

__eq__(*other*)
Return self==value.

__lt__(*other*)
Return self<value.

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

class TotalDepth.util.archive.**FileOnDisc**(*path: str*)
Represents an on-disc file.

__init__(*path: str*)
Initialize self. See help(type(self)) for accurate signature.

class TotalDepth.util.archive.**FileInMemory**(*path: str*, *size: int*, *binary_type: str*, *mod_date: datetime.datetime*, *by: bytes*)
Represents an in-memory file, for example contained in a ZIP. We need to be given the file data as we can't read it from disc.

__init__(*path: str*, *size: int*, *binary_type: str*, *mod_date: datetime.datetime*, *by: bytes*)
Initialize self. See help(type(self)) for accurate signature.

class TotalDepth.util.archive.**FileMembers**(*archive_path: str*, *depth: int*)
Represents a tree of files for example a ZIP might contain a ZIP.

__init__(*archive_path: str*, *depth: int*)
Initialize self. See help(type(self)) for accurate signature.

__str__()
Return str(self).

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

class TotalDepth.util.archive.**FileArchive**(*archive_path: str*, *depth: int*)
Represents a file that is an archive of other files.

__init__(*archive_path: str*, *depth: int*)
Initialize self. See help(type(self)) for accurate signature.

class TotalDepth.util.archive.**FileZip**(*archive_path: str*)
Represents an on-disc file that is a ZIP file.

__init__(*archive_path: str*)
Initialize self. See help(type(self)) for accurate signature.

__str__()
Return str(self).

TotalDepth.util.archive.**analyse_archive**(*files: List[TotalDepth.util.archive.FileBase]*, *file_types: List[str]*, *num_bytes: int*, *include_size_histogram: bool*) → None
Take a list of FileBase and write out the analysis.

TotalDepth.util.archive.**explore_tree_single_process**(*path: str*, *recurse: bool*) → List[TotalDepth.util.archive.FileBase]
Single process code to scan an archive.

TotalDepth.util.archive.**explore_tree_multi_process**(*path: str*, *recurse: bool*, *jobs: int*) → List[TotalDepth.util.archive.FileBase]
Multiprocessing code to scan an archive.

TotalDepth.util.archive.**copy_tree**(*path_from: str, path_to: str, recurse: bool, file_types: List[str], nervous: bool, over_write: bool*) → Tuple[Dict[str, int], int]
Copies particular binary file types from one directory structure to another.

TotalDepth.util.archive.**expand_and_delete_archives**(*target_dir: str*, *nervous: bool*) → Tuple[int, int]
Recursively searches for archives in target_dir, expands them and deletes the original archive. This repeatedly scans the target_dir until no more archive paths are found.

TotalDepth.util.bin_file_type (Binary File Type Classification)

Identifies the type of file as a string such as "PDF", "RP66V1" by an analysis (mostly) of the initial bytes of the file.

TotalDepth.util.bin_file_type.**RE_COMPILED = {'RP66V1': {'Comment_1': re.compile(b'^[0]***
Regular expressions for RP66 files. Keys refer to the documentation.

TotalDepth.util.bin_file_type.**ASCII_PRINTABLE_BYTES = {9, 10, 11, 12, 13, 32, 33, 34, 35, 3**
string.printable contains tab, backspace etc. which is undesirable: '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~ tnrx0bx0c'

TotalDepth.util.bin_file_type.**RE_LAS_VERSION_LINE = re.compile(b'^\\s*VERS\\s*\\.\\s+([\\d**
Regex for extracting the LAS version.

TotalDepth.util.bin_file_type.**LAS_BINARY_FILE_TYPES = {'LAS1.2', 'LAS2.0'}**
LAS binary file types we support

TotalDepth.util.bin_file_type.**FUNCTION_ID_MAP = ((<function _rcd>, 'RCD'), (<function _stk**
Ordered so that more specific files are earlier in the list, more general ones later. Also, as an optimisation, the more common file formats appear earlier.

TotalDepth.util.bin_file_type.**BINARY_FILE_TYPE_CODE_WIDTH = 8**
Length of the longest binary file type supported.

TotalDepth.util.bin_file_type.**BINARY_FILE_TYPES_SUPPORTED = {'ASCII', 'BIT', 'CFBF', 'DAT',**
Set of binary file types supported.

TotalDepth.util.bin_file_type.**LIS_BINARY_FILE_TYPES = {'LIS', 'LISt', 'LIStr'}**
LIS binary file types we support

TotalDepth.util.bin_file_type.**is_lis_file_type**(*file_type: str*) → bool
True if the binary file type is supported by the LIS parser.

TotalDepth.util.bin_file_type.**summary_file_types_supported**(*short: bool*) → str
Returns a string of the supported file types. If short is False this is a multi-line string.

TotalDepth.util.bin_file_type.**binary_file_type**(*fobj: BinaryIO*) → str
Function that takes a file object that supports read() and seek() and returns a file type based on the analysis of the contents of the file. On success fobj will be at the start of file. On failure fobj will be in an indeterminate state.

TotalDepth.util.bin_file_type.**binary_file_type_from_path**(*path: str*) → str
Returns a file type based on the analysis of the contents of the file.

TotalDepth.util.bin_file_type.**format_bytes**(*by: bytes*) → str
Formats bytes with xxd.

TotalDepth.util.gnuplot (Gnuplot Graphs Support)

A HOWTO is here: *Performance Graphs via Gnuplot with TotalDepth.util.gnuplot*

Provides gnuplot support to command line tools.

TotalDepth.util.gnuplot.**add_gnuplot_to_argument_parser**(*parser: argparse.ArgumentParser*) → None
Adds `--gnuplot=<DIRECTORY_FOR_GNUPLOT_OUTPUT>` to the argument parser as `args.gnuplot`.

TotalDepth.util.gnuplot.**version**() → bytes
For example: b'gnuplot 5.2 patchlevel 6'

TotalDepth.util.gnuplot.**create_gnuplot_dat**(*table: Sequence[Sequence[Any]]*) → str
Returns a pretty formatted string of the data in the given table suitable for use as a gnuplot `.dat` file.

TotalDepth.util.gnuplot.**invoke_gnuplot**(*path: str, name: str, table: Sequence[Sequence[Any]], plt: str*) → int
Create the plot for name. path - the directory to write the data and plot files to. name - the name of those files. table - the table of values to write to the data file.

Returns the gnuplot error code.

TotalDepth.util.gnuplot.**write_test_file**(*path: str, typ: str*) → int
Writes out a Gnuplot test file.

Plot Package Reference

Contents:

TotalDepth.util.plot.AREACfg (AREA Plotting Configuration)

Module for plotting areas of particular patterns with well log data.

Created on 1 Apr 2011

Provides access to patterns either using the Data URI Scheme (https://en.wikipedia.org/wiki/Data_URI_scheme) or as PNG files. Both can be in monochrome or RGB

`TotalDepth.util.plot.AREACfg.`**`png_location`**(*pattern: str*, *is_monochrome: bool*) → str
Returns the absolute path of the location of a PNG file for the pattern.

TotalDepth.util.plot.Coord (Plot Coordinates)

Main Classes

Most classes in this module are `collections.namedtuple` objects.

Class	Description	Attributes
`Dim`	Linear dimension	value units
`Box`	A Box	width depth
`Pad`	Padding around a tree object	prev next, parent child
`Margin`	Padding around an object	left right top bottom
`Pt`	A point in Cartesian space	x y

Reference

Provides a fairly basic two dimensional coordinate system.

exception `TotalDepth.util.plot.Coord.`**`ExceptionCoord`**
Exception class for representing Coordinates.

exception `TotalDepth.util.plot.Coord.`**`ExceptionCoordUnitConvert`**
Exception raised when converting units.

`TotalDepth.util.plot.Coord.`**`BASE_UNITS = 'px'`**
Base units for dimensions

`TotalDepth.util.plot.Coord.`**`UNIT_MAP = {None: 1.0, 'px': 1.0, 'pt': 1.0, 'pc': 12.0, 'i`**
Map of {unit name : conversion factor to base units, . . . }

`TotalDepth.util.plot.Coord.`**`exactConversion`**(*units_a*, *units_b='px'*)
Returns True it the two dimension can be converted exactly. This is the case where the units are the same or the factors are exact multiples.

`TotalDepth.util.plot.Coord.`**`UNIT_MAP_DEFAULT_FORMAT = {None: '%.4f', 'px': '%d', 'pt': '`**
Formatting strings for writing attributes. We are trying not to write 3.999999999mm here!

`TotalDepth.util.plot.Coord.`**`UNIT_MAP_DEFAULT_FORMAT_WITH_UNITS = {None: '%.4f%s', 'px': '`**
Map of formatting strings for value and units e.g. to create '0.667in' from (2.0 / 3.0, 'in')

TotalDepth.util.plot.Coord.**units**()
Returns the unsorted list of acceptable units.

TotalDepth.util.plot.Coord.**convert**(*val*, *unitFrom*, *unitTo*)
Convert a value from one set of units to another.

class TotalDepth.util.plot.Coord.**Dim**
Represents a dimension as an engineering value i.e. a number and units.

scale(*factor*)
Returns a new Dim() multiplied by a factor, units are unchanged.

divide(*factor*)
Returns a new Dim() divided by a factor, units are unchanged.

convert(*u*)
Returns a new Dim() with units changed and value converted.

__str__()
Return str(self).

__repr__()
Return a nicely formatted representation string

__format__(*format_spec*)
Default object formatter.

__add__(*other*)
Overload self+other, returned result has the sum of self and other. The units chosen are self's unless self's units are None in which case other's units are used (if not None).

__sub__(*other*)
Overload self-other, returned result has the difference of self and other. The units chosen are self's unless self's units are None in which case other's units are used (if not None).

__iadd__(*other*)
Addition in place, value of other is converted to my units and added.

__isub__(*other*)
Subtraction in place, value of other is subtracted.

__mul__(*other*)
Multiply by a factor that is a number.

__truediv__(*other*)
Divide by a factor that is a number.

__imul__(*other*)
Indirect multiply by a factor that is a number.

__itruediv__(*other*)
Indirect divide by a factor that is a number.

__lt__(*other*)
Return self<value.

__le__(*other*)
Return self<=value.

__eq__(*other*)
Return self==value.

__ne__(*other*)
Return self!=value.

__gt__(*other*)
Return self>value.

__ge__(*other*)
Return self>=value.

TotalDepth.util.plot.Coord.**dimIn**(*v*)
Returns a Dim object with the value in inches.

class TotalDepth.util.plot.Coord.**Box**

__str__()
Return str(self).

__repr__()
Return a nicely formatted representation string

__format__(*format_spec*)
Default object formatter.

class TotalDepth.util.plot.Coord.**Pad**
Padding around another object that forms the Bounding Box. All 4 attributes are Dim() objects

__str__()
Return str(self).

__repr__()
Return a nicely formatted representation string

__format__(*format_spec*)
Default object formatter.

class TotalDepth.util.plot.Coord.**Margin**
Margin padding around another object. All 4 attributes are Coord.Dim() objects.

__str__()
Return str(self).

__repr__()
Return a nicely formatted representation string

__format__(*format_spec*)
Default object formatter.

class TotalDepth.util.plot.Coord.**Pt**
A point, an absolute x/y position on the plot area. Members are Coord.Dim().

__eq__(*other*)
Comparison.

__str__()
Return str(self).

__repr__()
Return a nicely formatted representation string

__format__(*format_spec*)
Default object formatter.

convert(*u*)
Returns a new Pt() with units changed and value converted.

scale(*factor*)
Returns a new Pt() scaled by a factor, units are unchanged.

normalise_units(*units=None*)
Returns a point with both x and y with the same units. If units is given then x and y will be in those units. This may return self or a new point.

TotalDepth.util.plot.Coord.**to_cartesian**(*origin: TotalDepth.util.plot.Coord.Pt*, *radius: TotalDepth.util.plot.Coord.Dim*, *angle: float*) → TotalDepth.util.plot.Coord.Pt
Displaces a point by radius in direction angle in radians which is an x to y rotation. dx is cos(angle) and dy is sin(angle). For example in a SVG coordinate system where +x is right and +y down an angle of less than pi/2 will move the point to the right and down. For use in a mapping system where +x is northing/Latitude N and +y easting/Longitude E an angle of less then pi/2 will move the point up and to the right.

TotalDepth.util.plot.Coord.**to_polar**(*pt_from: TotalDepth.util.plot.Coord.Pt*, *pt_to: TotalDepth.util.plot.Coord.Pt*) → Tuple[TotalDepth.util.plot.Coord.Dim, float]
Returns a radius as a Dim and angle in radians. NOTE: This uses math.atan2() so returns the result is between -pi and pi.

Will raise if the given points are identical.

TotalDepth.util.plot.Coord.**baseUnitsDim**(*theLen*)
Returns a Coord.Dim() of length and units BASE_UNITS.

Parameters **theLen** (`float, int`) – Length.

Returns `cpip.plot.Coord.Dim([float, str])` – A new dimension of theLen in base units.

TotalDepth.util.plot.Coord.**zeroBaseUnitsDim**()
Returns a Coord.Dim() of zero length and units BASE_UNITS.

Returns `cpip.plot.Coord.Dim([float, str])` – A new dimension of zero.

TotalDepth.util.plot.Coord.**zeroBaseUnitsBox**()
Returns a Coord.Box() of zero dimensions and units BASE_UNITS.

TotalDepth.util.plot.Coord.**zeroBaseUnitsPad**()
Returns a Coord.Pad() of zero dimensions and units BASE_UNITS.

TotalDepth.util.plot.Coord.**zeroBaseUnitsPt**()
Returns a Coord.Dim() of zero length and units BASE_UNITS.

Returns `cpip.plot.Coord.Pt([cpip.plot.Coord.Dim([float, str]), cpip.plot.Coord.Dim([float, str])])` – A new point with the values [0, 0].

TotalDepth.util.plot.Coord.**pxUnitsDim**(*theLen*)
Returns a Coord.Dim() of length and units 'px'.

Parameters **theLen** (`float, int`) – Length.

Returns `cpip.plot.Coord.Dim([float, str])` – A new dimension of theLen in base units.

TotalDepth.util.plot.Coord.**pxBaseUnitsDim**()
Returns a Coord.Dim() of zero length and units 'px'.

Returns `cpip.plot.Coord.Dim([float, str])` – A new dimension of zero.

TotalDepth.util.plot.Coord.**pxBaseUnitsBox**()
Returns a Coord.Box() of zero dimensions and units 'px'.

TotalDepth.util.plot.Coord.**pxBaseUnitsPad**()
Returns a Coord.Pad() of zero dimensions and units 'px'.

TotalDepth.util.plot.Coord.**pxBaseUnitsPt**()
Returns a Coord.Dim() of zero length and units 'px'.

Returns cpip.plot.Coord.Pt([cpip.plot.Coord.Dim([float, str]), cpip.plot.Coord.Dim([float, str])]) – A new point with the values [0, 0].

TotalDepth.util.plot.Coord.**newPt**(*theP*, *incX=None*, *incY=None*)
Returns a new Pt object by incrementing existing point incX, incY that are both Dim() objects or None.

Parameters

- **theP** (cpip.plot.Coord.Pt([cpip.plot.Coord.Dim([float, str]), cpip.plot.Coord.Pt([cpip.plot.Coord.Dim([float, str])])) – The initial point.
- **incX** (NoneType, cpip.plot.Coord.Dim([float, str]), cpip.plot.Coord.Dim([int, str])) – Distance to move in the x axis.
- **incY** (NoneType, cpip.plot.Coord.Dim([float, str]), cpip.plot.Coord.Dim([int, str])) – Distance to move in the y axis.

Returns cpip.plot.Coord.Pt([cpip.plot.Coord.Dim([float, str]), cpip.plot.Coord.Dim([float, str])]) – The new point.

TotalDepth.util.plot.Coord.**convertPt**(*theP*, *theUnits*)
Returns a new point with the dimensions of theP converted to theUnits.

TODO: Deprecate this.

TotalDepth.util.plot.Coord.**mirrorPt**(*start: TotalDepth.util.plot.Coord.Pt*, *finish: TotalDepth.util.plot.Coord.Pt*) → TotalDepth.util.plot.Coord.Pt
Returns a new point that is the mirror of the finish point, 180 degrees from start to finish.

Examples

Coord.Dim()

Creation, addition and subtraction:

```
d = Coord.Dim(1, 'in') + Coord.Dim(18, 'px')
# d is 1.25 inches
d = Coord.Dim(1, 'in') - Coord.Dim(18, 'px')
# d is 0.75 inches
d += Coord.Dim(25.4, 'mm')
# d is 1.75 inches
```

Scaling and unit conversion returns a new object:

```
a = Coord.Dim(12, 'px')
b = myObj.scale(6.0)
# b is 72 pixels
c = b.convert('in')
# 1 is 1 inch
```

Comparison:

```
assert(Coord.Dim(1, 'in') == Coord.Dim(72, 'px'))
assert(Coord.Dim(1, 'in') >= Coord.Dim(72, 'px'))
assert(Coord.Dim(1, 'in') <= Coord.Dim(72, 'px'))
assert(Coord.Dim(1, 'in') > Coord.Dim(71, 'px'))
assert(Coord.Dim(1, 'in') < Coord.Dim(73, 'px'))
```

Coord.Pt()

Creation:

```
p = Coord.Pt(
        Coord.Dim(12, 'px'),
        Coord.Dim(24, 'px'),
        )
print(p)
# Prints: 'Pt(x=Dim(12px), y=Dim(24px))'
p.x # Coord.Dim(12, 'px'))
p.y # Coord.Dim(24, 'px'))
# Scale up by 6 and convert units
pIn = p.scale(6).convert('in')
# pIn now 'Pt(x=Dim(1in), y=Dim(2in))'
```

Testing

The unit tests are in `test/TestCoord.py`.

TotalDepth.util.plot.FILMCfg (FILM Plotting Configuration)

Represents the part of a plot configuration that, typically, can be obtained from a LIS FILM table.

Created on 21 Mar 2011

Example of the data in a film table:

```
Table record (type 34) type: FILM

MNEM  GCOD  GDEC  DEST  DSCA
------------------------------

1     EEE   ----  PF2   D200
2     EEE   ----  PF1   DM
```

Other FILM Table Examples:

```
Table record (type 34) type: FILM

MNEM  GCOD  GDEC  DEST  DSCA
------------------------------
1     EEB   ----  PF1   D200
2     EEB   ----  PF2   DM

Table record (type 34) type: FILM

```

(continues on next page)

(continued from previous page)

```
MNEM  GCOD  GDEC  DEST  DSCA
-----------------------------
1     E20   -4--  PF1   D200
2     EEE   ----  PF2   D200

MNEM  GCOD  GDEC  DEST  DSCA
-----------------------------
1     EEE   ----  PF1   D200
2     E1E   -4-   PF2   D200

Table record (type 34) type: FILM

MNEM  GCOD  GDEC  DEST  DSCA
-----------------------------

D     E3E   -3-   PFD   D200
E     E3E   -3-   PFE   D500
5     EB0   ---   PF5   D200
6     EEB   ---   PF6   D200

Table record (type 34) type: FILM

MNEM  GCOD  GDEC  DEST  DSCA
-----------------------------

8     EB0   ---   PF8   D200
A     LLLL  1111  PFA   DM
E     E4E   -4-   PFE   D200
K     E4E   -4-   PFK   D500

Table record (type 34) type: FILM

MNEM      CINT      GCOD  GDEC  DEST  DSCA
-----------------------------------------

1          0.00000  E2E   -2-   PF1   D200
2          0.00000  E2E   -1-   PF2   D500
3          0.00000  EEE   ----  NEIT  S5
4          0.00000  EEE   ----  NEIT  S5

Table record (type 34) type: FILM

MNEM      CINT      GCOD  GDEC  DEST  DSCA
-----------------------------------------

1          300.000  E2E   -2-   PF1   D200
2          300.000  E2E   -2-   PF2   D500
3          300.000  EEE   ----  NEIT  S5
4          300.000  EEE   ----  NEIT  S5

Table record (type 34) type: FILM

MNEM      CINT      GCOD  GDEC  DEST  DSCA
-----------------------------------------

1          50.0000  EEE   ----  PF1   D200
2          0.00000  EEE   ----  PF2   D200
```

(continues on next page)

(continued from previous page)

```
3          0.00000  EEE   ----  NEIT  D200
4          0.00000  EEE   ----  NEIT  D200
```

Minimal, but not complete interpretation:

Ignore GDEC as dupe.

- E - Equi-spaced (linear).
- n - Log with number of decades.
- B - Blank.
- L - ?

What to do with 0 (continuation?). Examples: E20 -4– means 4 decades over track 23.

Four tracks, from 200099.S07:

```
Table record (type 34) type: FILM
MNEM  GCOD  GDEC  DEST  DSCA
------------------------------
...
A     LLLL  1111  PFA   DM
...

Table record (type 34) type: PRES
MNEM  OUTP  STAT  TRAC  CODI  DEST  MODE      FILT          LEDG          REDG      ␣
↪COLO
-------------------------------------------------------------------------------------
↪---
...
AX1A  AX    ALLO  F2    LLIN  A     SHIF      0.500000      -9.00000       9.00000 ␣
↪400
AY1A  AY    ALLO  F2    LSPO  A     SHIF      0.500000      -9.00000       9.00000 ␣
↪400
AZ1A  AZ    ALLO  F2    LGAP  A     NB        0.500000      -9.00000       9.00000 ␣
↪400
AN1A  ANOR  ALLO  F4    HDAS  A     NB        0.500000       9.00000       11.0000 ␣
↪420
CS1A  CS    DISA  F1    LDAS  A     NB        0.500000       0.00000       150000. ␣
↪000
FX1A  FX    ALLO  F3    LLIN  A     NB        0.500000     -0.700000      0.700000 ␣
↪003
FY1A  FY    ALLO  F3    LGAP  A     NB        0.500000     -0.700000      0.700000 ␣
↪003
FZ1A  FZ    ALLO  F3    LSPO  A     NB        0.500000     -0.700000      0.700000 ␣
↪003
FN1A  FNOR  ALLO  F4    LLIN  A     NB        0.500000      0.200000      0.700000 ␣
↪420
FI1A  FINC  ALLO  F4    LGAP  A     NB         1.00000       0.00000       90.0000 ␣
↪420
GA1A  GADZ  DISA  F1    LLIN  A     NB         1.00000      -1.00000       1.00000 ␣
↪020
GP1A  GPV   DISA  F1    LGAP  A     NB         1.00000       14.0000       16.0000 ␣
↪020
GN1A  GNV   DISA  F1    LDAS  A     NB         1.00000      -16.0000      -14.0000 ␣
↪020
GM1A  GMT   DISA  F3    HDAS  A     SHIF       1.00000       80.0000       130.000 ␣
↪020
```

(continues on next page)

(continued from previous page)

```
GA2A  GAT   DISA  F2    HDAS  A     SHIF       1.00000       80.0000       130.000 ␣
↪020
SI1A  SILO  DISA  FD    HLIN  A     NB         1.00000       0.00000       20.0000 ␣
↪000
ST1A  STIT  DISA  FD    LLIN  A     NB         1.00000       0.00000       20.0000 ␣
↪000
ST2A  STIA  ALLO  FD    LLIN  A     NB         1.00000       0.00000       20.0000 ␣
↪000
TE1A  TENS  DISA  FD    LDAS  A     WRAP      0.500000       2000.00       7000.00 ␣
↪000
...
```

TODO: What about CINT and FORM headings?

2011-05-27 Frequency Analysis done on:

```
$ python3 TableHistogram.py -k --name=FILM -l40 --col=DEST ../../../pLogicTestData/
↪LIS/
Cmd: TableHistogram.py -k --name=FILM -l40 --col=DEST ../../../pLogicTestData/LIS/
2011-05-27 09:23:23,870 ERROR    Can not read LIS file ../../../pLogicTestData/LIS/
↪13576.S1 with error: Can not fit integer number of frames length 120 into LR length␣
↪824, modulo 104 [indirect size 0].
2011-05-27 09:23:24,649 ERROR    Can not read LIS file ../../../pLogicTestData/LIS/
↪13610.S1 with error: Can not fit integer number of frames length 7176 into LR␣
↪length 13354, modulo 6178 [indirect size 0].
...
```

GCOD:

```
======================== Count of all table entries ========================
{
"(34, b'FILM', b'GCOD', b'BBB ')": 2,
"(34, b'FILM', b'GCOD', b'E1E ')": 5,
"(34, b'FILM', b'GCOD', b'E20 ')": 26,
"(34, b'FILM', b'GCOD', b'E2E ')": 40,
"(34, b'FILM', b'GCOD', b'E3E ')": 3,
"(34, b'FILM', b'GCOD', b'E40 ')": 2,
"(34, b'FILM', b'GCOD', b'E4E ')": 3,
"(34, b'FILM', b'GCOD', b'EB0 ')": 2,
"(34, b'FILM', b'GCOD', b'EEB ')": 22,
"(34, b'FILM', b'GCOD', b'EEE ')": 225,
"(34, b'FILM', b'GCOD', b'LLLL')": 1,
}
======================== Count of all table entries END ========================
```

GDEC:

```
======================== Count of all table entries ========================
{
"(34, b'FILM', b'GDEC', b'--- ')": 10,
"(34, b'FILM', b'----')": 227,
"(34, b'FILM', b'-1- ')": 2,
"(34, b'FILM', b'-2- ')": 6,
"(34, b'FILM', b'-2--')": 32,
"(34, b'FILM', b'-3- ')": 3,
"(34, b'FILM', b'-4- ')": 10,
"(34, b'FILM', b'-4--')": 26,
```

(continues on next page)

(continued from previous page)

```
"(34, b'FILM', b'GDEC', b'1111')": 1,
"(34, b'FILM', b'GDEC', b'EEE ')": 14,
}
====================== Count of all table entries END ====================
```

DEST:

```
======================== Count of all table entries ======================
{
"(34, b'FILM', b'DEST', b'NEIT')": 66,
"(34, b'FILM', b'DEST', b'PF1 ')": 124,
"(34, b'FILM', b'DEST', b'PF2 ')": 125,
"(34, b'FILM', b'DEST', b'PF5 ')": 1,
"(34, b'FILM', b'DEST', b'PF6 ')": 5,
"(34, b'FILM', b'DEST', b'PF8 ')": 1,
"(34, b'FILM', b'DEST', b'PFA ')": 1,
"(34, b'FILM', b'DEST', b'PFD ')": 2,
"(34, b'FILM', b'DEST', b'PFE ')": 3,
"(34, b'FILM', b'DEST', b'PFJ ')": 2,
"(34, b'FILM', b'DEST', b'PFK ')": 1,
}
====================== Count of all table entries END ====================
```

DSCA:

```
======================== Count of all table entries ======================
{
"(34, b'FILM', b'DSCA', b'D200')": 156,
"(34, b'FILM', b'DSCA', b'D500')": 17,
"(34, b'FILM', b'DSCA', b'DM  ')": 76,
"(34, b'FILM', b'DSCA', b'S5  ')": 82,
}
====================== Count of all table entries END ====================
```

exception TotalDepth.util.plot.FILMCfg.**ExceptionFILMCfg**
Specialisation of exception for this module.

exception TotalDepth.util.plot.FILMCfg.**ExceptionPhysFilmCfg**
Specialisation of exception for PhysFilmCfg.

exception TotalDepth.util.plot.FILMCfg.**ExceptionFilmCfgLISRead**
Specialisation of exception for FilmCfgLISRead in this module.

class TotalDepth.util.plot.FILMCfg.**PhysFilmCfg**(*theName*, *theTracks*, *theDest*, *theX*)
Contains the configuration equivalent to a single line in a FILM table.

theName is a hashable.

theTracks is a list of Track.Track objects.

theX is an integer scale.

__init__(*theName*, *theTracks*, *theDest*, *theX*)
Constructor. theName is a hashable. theTracks is a list of Track.Track objects. theX is an integer scale.

property name
Name of the FILM.

property xScale
The FILM X axis scale as a number.

__len__()
Number of Track.Track objects.

__getitem__(*i*)
Returns the Track.Track object at position i.

genTracks()
Generate all tracks.

interpretTrac(*theTracStr*)
Turns TRAC information into left/right positions as Cooord.Dim() objects and the number of half-tracks of the start and half-tracks covered. The later two values are used for stacking and packing the plot header and footer scales so that they take the minimum space.

e.g. b'T23 ' returns: (the left position of T2, right of T3, 4, 4).

e.g. b'T2 ' returns: (the left position of T2, right of T2, 4, 2).

Note: There is some fudging going on here

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

class TotalDepth.util.plot.FILMCfg.**PhysFilmCfgLISRead**(*theRow*)
Tracks from a LIS FILM table, essentially the pair of GCOD and GDEC defines the left-to-right layout of the plot.

Grid codes from analysis above: b'BBB ', b'E1E ', b'E20 ', b'E2E ', b'E3E ', b'E40 ', b'E4E ', b'EB0 ', b'EEB ', b'EEE ', b'LLLL'

The DSCA defines the top-to-bottom nature of the plot. DSCA codes from analysis above: b'D200', b'D500', b'DM ', b'S5 '

But we can guess some others...

DSCA_MAP = {b'D20 ': 20, b'D200': 200, b'D240 ': 240, b'D40 ': 40, b'D500': 500, I
X axis scale from a LIS FILM table DSCA codes from analysis above: b'D200', b'D500', b'DM ', b'S5 '
But we can guess some others...

__init__(*theRow*)
Reads a LogiRec.TableRow object and populates a CurveCfg.

Example:

```
MNEM  GCOD  GDEC  DEST  DSCA
-----------------------------
1     E20   -4--  PF1   D200
```

supportedFilmTracks()
A list of supported film (name, decade) pairs.

class TotalDepth.util.plot.FILMCfg.**FilmCfg**
Contains the configuration equivalent to a complete FILM table.

__init__()
Initialize self. See help(type(self)) for accurate signature.

add(*k*, *thePfc*)
Add a PhysFilmCfg object to the map with key k, typically a FILM mnemonic in bytes such as Mnem.Mnem(b'1 ') or some other ID such as a string, the filename of an XML file.

keys()
All FILM Mnemonics.

__len__()
Number of unique film destination names.

__getitem__(*name*)
Returns the PhysFilmCfg object corresponding to name - a Mnem() object. Will raise KeyError if not exact match. See retFilmDest() for an API that can handle curve destinations of BOTH, ALL etc.

__contains__(*name*)
Membership test.

retAllFILMDestS(*curveDestID*)
Returns an unordered list of FILM destinations for a curve destination. For example if curveDestID is b'BOTH' this might return [b'2 ', b'1 ']

retFILMDest(*filmDestID*, *curveDestID*)
Returns a PhysFilmCfg object by matching curveDestID to the filmDestID. Returns None on failure. For LIS curveDestID can be 1, BOTH, ALL, NEIT etc. This is commonly used by the PRESCfg module so that interpretTrac() can be called on the result and thus build up a map of track positions for all possible logical film outputs.

interpretTrac(*filmDestID*, *curveDestID*, *trackStr*)
Given a film destination ID and a curve destination (which could be b'ALL') and a track string (e.g. b'T23') this returns the left/right positions as Cooord.Dim() objects and the number of half-tracks of the start and half-tracks covered (used for plot header and footer scales). Returns None if there is no match for the filmDestID/curveDestID (for example if curveDestID is b'NEIT'). e.g. (b'1 ', b'BOTH', b'T23 ') returns: (the left position of T2, right of T3, 4, 4).

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

class TotalDepth.util.plot.FILMCfg.**FilmCfgLISRead**(*theLr*)
Interprets a FILM table from a LIS Logical Record.

__init__(*theLr*)
Reads a LogiRec.Table object and creates a PhysFilmCfgLISRead for each row.

Typical FILM table:

```
MNEM  GCOD  GDEC  DEST  DSCA
------------------------------
1     E20   -4--  PF1   D200
2     EEE   ----  PF2   D200
```

retAllFILMDestS(*curveDestID*)
Returns an unordered list of FILM destinations for a curve destination.

For example if curveDestID is b'BOTH' this might return [b'2 ', b'1 ']

retFILMDest(*filmDestID*, *curveDestID*)
Returns a PhysFilmCfg object by matching curveDestID to the filmDestID. Returns None on failure. curveDestID can be 1, BOTH, ALL, NEIT etc.

This is commonly used by the PRESCfg module so that interpretTrac() can be called on the result and thus build up a map of track positions for all possible logical film outputs.

TotalDepth.util.plot.FILMCfgXML (FILM Plotting Configuration from XML files)

Creates FILM configurations from LgFormat XML files.

Created on Dec 14, 2011

exception TotalDepth.util.plot.FILMCfgXML.**ExceptionFILMCfgXML**
Specialisation of exception for FILMCfgXML module.

exception TotalDepth.util.plot.FILMCfgXML.**ExceptionFILMCfgXMLRead**
Specialisation of exception for FILMCfgXMLRead module.

exception TotalDepth.util.plot.FILMCfgXML.**ExceptionFILMCfgXMLReadLookUp**
Specialisation of exception for FILMCfgXMLRead module when a lookup fails that would normally raise a KeyError.

class TotalDepth.util.plot.FILMCfgXML.**PhysFilmCfgXMLRead**(*root*)
Extracts FILM information from a single XML file root element.

TRAC_STANDARD_SPAN = 2
Default number of half-tracks to span

TRAC_UNIQUEID_TO_NON_STANDARD_SPAN = {'TrackFC1': 1, 'TrackFC2': 1, 'TrackFC3': 1,
Map of track IDs to start track location in half-tracks

TRAC_UNIQUEID_TO_NON_STANDARD_START = {'TrackFC1': 4, 'TrackFC2': 5, 'TrackFC3': 6,
Number of half-tracks of the start of the track, this is a fudge around deciding this from the order of the LgTrack elements

__init__(*root*)
Constructor. theName is a hashable. theTracks is a list of Track.Track objects. theX is an integer scale.

interpretTrac(*theTracStr*)
Turns TRAC information into left/right positions as Cooord.Dim() objects and the number of half-tracks of the start and half-tracks covered. The later two values are used for stacking and packing the plot header and footer scales so that they take the minimum space. e.g. b'T23 ' returns: (the left position of T2, right of T3, 4, 4). e.g. b'T2 ' returns: (the left position of T2, right of T2, 4, 2). Note: There is some fudging going on here

class TotalDepth.util.plot.FILMCfgXML.**FilmCfgXMLRead**(*directory=None*)
Contains the configuration equivalent to a complete FILM table from a set of XML files.

__init__(*directory=None*)
Constructor with a directory, all files in the directory (non-recursive) are read as LGFormat XML files. If dir is None then the "formats/" directory relative to this module is searched. If dir is an empty string then no search will be done - useful for testing with addXMLRoot(etree.fromstring(..)).

readDir(*d=None*)
Read a directory of XML files (not recursive).

addXMLRoot(*theRoot*)
Adds a parsed LgFormat XML document to the IR.

chOutpMnemInFilmId(*chOutp*, *filmID*)
Returns True if this curve appears in the film.

longStr(*verbose=1*)
Returns a long string of the XML UniqueIds and their description.

uniqueIdS()
Returns the UniqueId values that I know about.

description(*theUID*)
Returns the XML description corresponding to the Unique ID or None if unknown.

rootNode(*theUID*)
Returns the XML root node corresponding to the Unique ID.

retAllFILMDestS(*curveDestID*)
Returns an unordered list of FILM destinations for a curve destination. For example if curveDestID is b'BOTH' this might return [b'2 ', b'1 ']

retFILMDest(*filmDestID*, *curveDestID*)
Returns a PhysFilmCfgXMLRead object by matching curveDestID to the filmDestID. Returns None on failure. filmDestID and curveDestID are implementation specific e.g. for LIS the curveDestID can be 1, BOTH, ALL, NEIT etc. This is commonly used by the PRESCfg module so that interpretTrac() can be called on the result and thus build up a map of track positions for all possible logical film outputs.

TotalDepth.util.plot.LogHeader (Plotting Well Log API headers)

Plots various headers as SVG. Of note is the API header

Created on Dec 30, 2011

exception TotalDepth.util.plot.LogHeader.**ExceptionLogHeader**
Exception for plotting Log Headers.

exception TotalDepth.util.plot.LogHeader.**ExceptionLogHeaderLIS**
Exception for plotting Log Headers from LIS data.

exception TotalDepth.util.plot.LogHeader.**ExceptionLogHeaderLAS**
Exception for plotting Log Headers from LAS data.

class TotalDepth.util.plot.LogHeader.**Static**(*x*, *y*, *w*, *d*, *text*, *font*, *size*, *tAttr*, *rAttr*, *mnem*, *xMnem*)
Tuple to describe static data locations If text is None then no text will be plotted If rAttr is None then no box will be plotted If mnem is non-None then a Mmem can be plotted at xMenm increment, a Coord.Dim()

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *x*, *y*, *w*, *d*, *text*, *font*, *size*, *tAttr*, *rAttr*, *mnem*, *xMnem*)
Create new instance of Static(x, y, w, d, text, font, size, tAttr, rAttr, mnem, xMnem)

__repr__()
Return a nicely formatted representation string

property d
Alias for field number 3

property font
Alias for field number 5

property mnem
Alias for field number 9

property rAttr
Alias for field number 8

property size
Alias for field number 6

property tAttr
Alias for field number 7

property text
Alias for field number 4

property w
Alias for field number 2

property x
Alias for field number 0

property xMnem
Alias for field number 10

property y
Alias for field number 1

TotalDepth.util.plot.LogHeader.**FONT_PROP = 'Verdana'**
Default font

TotalDepth.util.plot.LogHeader.**RECT_ATTRS_FINE = {'fill': 'none', 'stroke': 'black', 'st**
SVG attributes for a fine lined rectangle

TotalDepth.util.plot.LogHeader.**TEXT_ATTRS_FINE = {'text-anchor': 'start'}**
SVG attributes for a fine text

TotalDepth.util.plot.LogHeader.**TEXT_ATTRS_LARGE = {'font-weight': 'bold', 'text-anchor':**
SVG attributes for a bold text

TotalDepth.util.plot.LogHeader.**TEXT_ATTRS_LARGE_WOB = {'fill': 'white', 'font-weight': 'b**
SVG attributes for a large WOB text

TotalDepth.util.plot.LogHeader.**HEADER_PLOT_UNITS = 'in'**
Standard plot units used in the layout definition

class TotalDepth.util.plot.LogHeader.**APIHeaderBase**(*isTopOfLog=False*)
Base class to be used by APIHeaderLIS or APIHeaderLAS.

If isTopOfLog is True plot is rotated 90 deg as if to fit on top of a traditional log.

__init__(*isTopOfLog=False*)
Constructor with flag that controls plot orientation. If True plot is rotated 90 deg as if to fit on top of a traditional log.

missingFields(*theWsd*)
Returns two sets: A set of mnemonics that could be plotted but are not in the Logical Record(s). A set of mnemonics that are in the Logical Record(s) but could not be plotted.

size()
Returns a Coord.Box for my size, currently a single page on fan folded paper.

viewPort(*theTl*)
The SVG viewport.

plot(*xS*, *theTl*, *theWsdS=None*)
Write the header to the SVG stream at position offset top left. theWsd is a list of records that contain well site data. Will raise ExceptionLogHeader is wrong type of Logical Record.

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

class TotalDepth.util.plot.LogHeader.**APIHeaderLIS**(*isTopOfLog=False*)
Can lay out an API header from LIS information, specifically a type 34 CONS record.

If isTopOfLog is True plot is rotated 90 deg as if to fit on top of a traditional log.

__init__(*isTopOfLog=False*)
Constructor with a Logical Record.

missingFields(*theWsd*)
Well site data (theWsd) in LIS is a list of CONS Logical Records.

Returns two sets:

A set of mnemonics that could be plotted but are not in the Logical Record(s).

A set of mnemonics that are in the Logical Record(s) but could not be plotted.

lrDataCount(*theLrS*)
Returns the number of Mnem's in MNEM_SET that could be plotted in a header that are found in all the Logical Records.

class TotalDepth.util.plot.LogHeader.**APIHeaderLAS**(*isTopOfLog=False*)
Can lay out an API header from a representation of a LAS file.

LIS_MNEM_TO_LAS_MNEM = {'BLI': 'STRT', 'CN': 'COMP', 'DFD': 'FD', 'FL': 'LOC', 'FL1':
Some conversions from LIS standard to LAS standard All as truncated ascii strings i.e. using pStr(strip=True)

__init__(*isTopOfLog=False*)
Constructor with a Logical Record.

missingFields(*theLasFile*)
Returns two sets:

A set of mnemonics that could be plotted but are not in the Logical Record(s).

A set of mnemonics that are in the Logical Record(s) but could not be plotted.

TotalDepth.util.plot.Plot (Plotting Well Logs)

Created on 28 Feb 2011

Plotting LIS data requires these components:

- A PlotConfig object
- A data set (e.g. a LogPass with a FrameSet).
- An output driver (e.g. screen, print PDF, web SVG).

User creates a PlotConfig (this reflects a PRES table). This is reusable.

User specifies a data set (from, to, channels etc.). e.g. Invoke LogPass.setFrameSet(File, theFrameSlice=theFrameSlice, theChList=theChList)

User says 'plot this data set with this configuration to this output device'. e.g. Plot(PlotConfig, LogPass, PlotDevice) Plot uses LogPass.genChScValues(ch, sc) to plot individual curves.

Lacunae

Area plotting. Caching (e.g. SVG fragments - is this worth it?)

PlotConfig

PlotTracks

Typically a three track (+depth) have these dimensions in inches:

Track	Left	Right	Width
1	0	2.4	2.4
Depth	2.4	3.2	0.8
2	3.2	5.6	2.4
3	5.6	8.0	2.4

Track names can be split (e.g. LHT1 is left hand track 1) or merged (T23 is spread across tracks two and three).

Examples of PRES and FILM records:

```
 Table record (type 34) type: PRES

MNEM  OUTP  STAT  TRAC  CODI  DEST  MODE      FILT          LEDG          REDG
--------------------------------------------------------------------------------

SP    SP    ALLO  T1    LLIN  1     SHIF      0.500000      -80.0000       20.0000
CALI  CALI  ALLO  T1    LDAS  1     SHIF      0.500000       5.00000       15.0000
MINV  MINV  DISA  T1    LLIN  1     SHIF      0.500000       30.0000       0.00000
MNOR  MNOR  DISA  T1    LDAS  1     SHIF      0.500000       30.0000       0.00000
LLD   LLD   ALLO  T23   LDAS  1     GRAD      0.500000      0.200000       2000.00
LLDB  LLD   ALLO  T2    HDAS  1     GRAD      0.500000       2000.00       200000.
LLG   LLG   DISA  T23   LDAS  1     GRAD      0.500000      0.200000       2000.00
LLGB  LLG   DISA  T2    HDAS  1     GRAD      0.500000       2000.00       200000.
LLS   LLS   ALLO  T23   LSPO  1     GRAD      0.500000      0.200000       2000.00
LLSB  LLS   ALLO  T2    HSPO  1     GRAD      0.500000       2000.00       200000.
MSFL  MSFL  ALLO  T23   LLIN  1     GRAD      0.500000      0.200000       2000.00

Table record (type 34) type: FILM

MNEM  GCOD  GDEC  DEST  DSCA
------------------------------

1     E20   -4--  PF1   D200
2     EEE   ----  PF2   D200

Table record (type 34) type: PRES

MNEM  OUTP  STAT  TRAC  CODI  DEST  MODE      FILT          LEDG          REDG
--------------------------------------------------------------------------------

NPHI  NPHI  ALLO  T23   LDAS  1     SHIF      0.500000      0.450000      -0.150000
DRHO  DRHO  ALLO  T3    LSPO  1     NB        0.500000      -0.250000      0.250000
PEF   PEF   ALLO  T23   LGAP  1     SHIF      0.500000       0.00000       10.0000
SGR         DISA  T1    LLIN  1     SHIF      0.500000       0.00000       300.000
```

(continues on next page)

(continued from previous page)

CGR		DISA	T1	LGAP	1	SHIF	0.500000	0.00000	300.000
TENS	TENS	DISA	T3	LGAP	1	SHIF	0.500000	14000.0	4000.00
CAL	CALI	ALLO	T1	LSPO	1	SHIF	0.500000	5.00000	15.0000
BS	BS	DISA	T1	LGAP	1	SHIF	0.500000	5.00000	15.0000
FFLS	FFLS	DISA	T1	LLIN	2	NB	0.500000	-0.150000	0.150000
FFSS	FFSS	DISA	T1	LDAS	2	NB	0.500000	-0.150000	0.150000
LSHV	LSHV	DISA	T3	LLIN	2	WRAP	0.500000	2150.00	2250.00
SSHV	SSHV	DISA	T3	LDAS	2	WRAP	0.500000	1950.00	2050.00
FLS	FLS	DISA	T2	LLIN	2	SHIF	0.500000	0.00000	150.000
FSS	FSS	DISA	T2	LDAS	2	SHIF	0.500000	0.00000	150.000
RHOB	RHOB	ALLO	T23	LLIN	1	SHIF	0.500000	1.95000	2.95000
PHIX	PHIX	ALLO	T1	LLIN	1	NB	0.500000	0.500000	0.00000

TotalDepth.util.plot.Plot.**COMMENTS_IN_SVG_TRACE = False**
Allows detailed trace comments to appear in the SVG

TotalDepth.util.plot.Plot.**COMMENTS_IN_SVG_SECTION = True**
A few comments in SVG to delinialte header, Grid, Xaxis, curves etc

TotalDepth.util.plot.Plot.**COMMENTS_IN_SVG_SECTION_WIDTH = 40**
Width of comment for sections

TotalDepth.util.plot.Plot.**COMMENTS_IN_SVG_SECTION_LEVEL_TUPLE = ('=', '.', '^')**
Map of section level to comment padding character

exception TotalDepth.util.plot.Plot.**ExceptionTotalDepthLISPlot**
Exception for plotting.

exception TotalDepth.util.plot.Plot.**ExceptionTotalDepthPlotRoll**
Exception for plotting.

class TotalDepth.util.plot.Plot.**CurvePlotScale**
Holds a minimal amount of curve plot scale and so on for the layout of the scale pane at each end of the log.

halfTrackStart is the start of the curve for a standard three track log T1=0, TD=2, T2=4, T3=6

halfTracks is the curve span as an integer so T23=4 LHT1=1 and so on.

__lt__(*other*)
Slightly weird sort order, larger halfTracks come first then names.

class TotalDepth.util.plot.Plot.**ScaleSliceCurve**(*slice*, *curveName*, *start*, *span*)

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *slice*, *curveName*, *start*, *span*)
Create new instance of ScaleSliceCurve(slice, curveName, start, span)

__repr__()
Return a nicely formatted representation string

property curveName
Alias for field number 1

property slice
Alias for field number 0

property span
Alias for field number 3

property start
Alias for field number 2

class TotalDepth.util.plot.Plot.**CurvePlotScaleSlotMap**(*theCpsS*)
Keeps track of which slots are available for putting the curve scales in at the top and bottom of the log. This scale are is divided into slices that span the plot from left to right. These slices are subdivided into slots that correspond to a half-track in that slice.

__init__(*theCpsS*)
Ctor with a list of CurvePlotScale objects (can be unsorted).

reset()
Clears the slot map for the current slice.

canFit(*theCps*)
Returns True if I can fit the CurvePlotScale object in the current slice.

fit(*theCps*)
Populates slots from a CurvePlotScale, caller should call canFit() first.

genScaleSliceCurve()
Generates a ordered list of ScaleSliceCurve objects laid out in a 'nice' fashion.

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

class TotalDepth.util.plot.Plot.**PlotRoll**(*theXStart*, *theXStop*, *theScale*, *theLegendDepth*, *theHeadDepth=Dim(0, 'in')*, *theTailDepth=Dim(0, 'in')*, *plotUp=True*, *theWidth=Dim(8.5, 'in')*, *theMargin=Margin(left=Dim(0.25, 'in')*, *right=Dim(0.25, 'in')*, *top=Dim(0.25, 'in')*, *bottom=Dim(0.25, 'in'))*)
Describes the plot canvas as if it were a roll of paper. This can compute the various dimensions and positions of plot panes:

Legend:

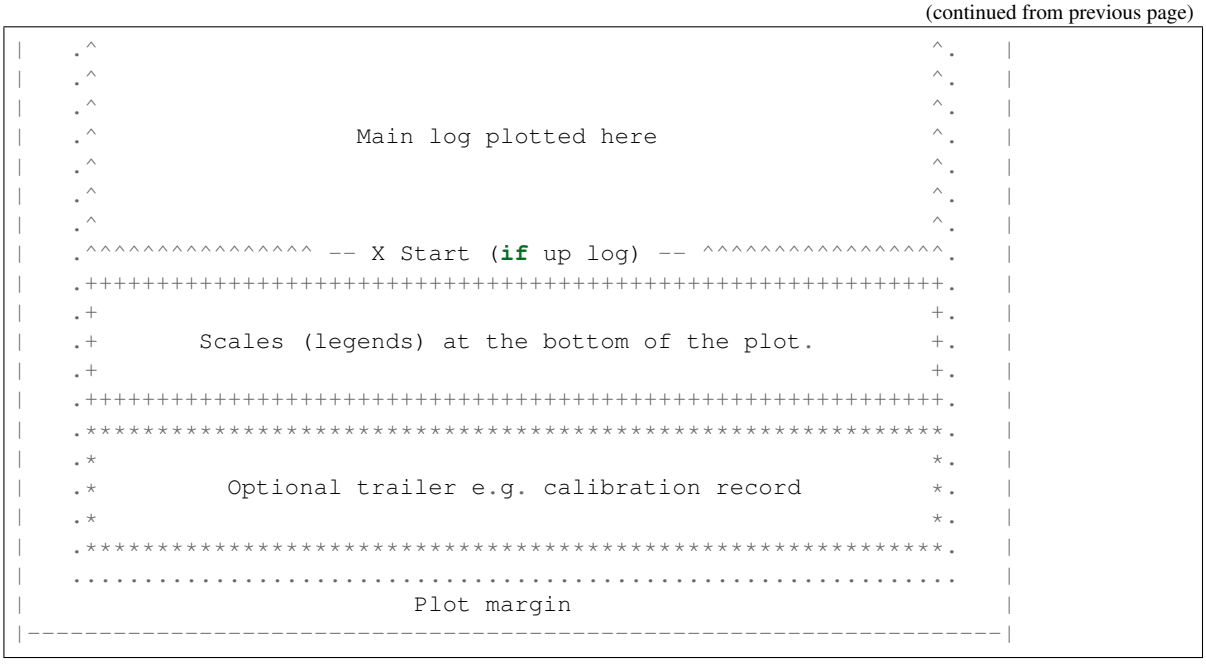
- `...` - The plot within margins.
- `***` - Optional headers and trailers.
- `+++` - The upper and lower headers for scales (legends).
- `^^^` - The main plotting area.

In this diagram adjacent lines overlay:

```
|-----------------------------------------------------------------|
|                           Plot margin                           |
|   .........................................................     |
|   .*********************************************************.    |
|   .*                                                       *.    |
|   .*            Optional header e.g. API header            *.    |
|   .*                                                       *.    |
|   .*********************************************************.    |
|   .+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++.    |
|   .+                                                       +.    |
|   .+       Scales (legends) at the top of the plot.        +.    |
|   .+                                                       +.    |
|   .+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++.    |
|   .^^^^^^^^^^^^^^^ -- X Stop (if up log) -- ^^^^^^^^^^^^^^^^^.    |
```

(continues on next page)

(continued from previous page)



__init__(*theXStart*, *theXStop*, *theScale*, *theLegendDepth*, *theHeadDepth=Dim(0, 'in')*, *theTailDepth=Dim(0, 'in')*, *plotUp=True*, *theWidth=Dim(8.5, 'in')*, *theMargin=Margin(left=Dim(0.25, 'in')*, *right=Dim(0.25, 'in')*, *top=Dim(0.25, 'in')*, *bottom=Dim(0.25, 'in'))*)

Initialise with:

theXStart The X start position as an `EngVal`.

theXStop The X stop position as an `EngVal`.

theScale The plot scale as a number.

theLegendDepth A `Coord.Dim()` that is the depth of the scales used in the header and footer.

theHeadDepth A `Coord.Dim()` that is the depth of any log header.

theTailDepth A `Coord.Dim()` that is the depth of any log trailer.

plotUp, bool True if X start is at the bottom of the main pane.

theWidth The absolute width of the plot as a `Coord.Dim()`.

theMargin A `Coord.Margin()` that describes the untouchable edges of the plot.

property viewBox

The overall size of the plot.

property width

The overall width as a number in PlotConstants.DEFAULT_PLOT_UNITS.

property depth

The overall width as a number in PlotConstants.DEFAULT_PLOT_UNITS.

property widthDim

The overall width as a Coord.Dim().

property depthDim

The overall width as a Coord.Dim().

property trackTopLeft
"A Coord.Pt() that is the top left of the pane that tracks are plotted within.

property mainPanePlotDepth
The depth of the plot of the main pane as a Coord.Dim().

property availableWidth
The available width inside the margins.

retHeadPane()
Returns a pair of top-left Coord.Pt(), Coord.Box() for the top header where the header goes.

retLegendPane(*isTop*)
Returns a pair of top-left Coord.Pt(), Coord.Box() for the top or bottom legend pane where the scales go.

retMainPane()
Returns a pair of top-left Coord.Pt(), Coord.Box() for the pane where the main log goes.

retTailPane()
Returns a pair of top-left Coord.Pt(), Coord.Box() for the pane where the trailer goes.

xDepth(*theX*)
Returns a Coord.Dim() on the main pane that corresponds to theX as a number or an EngVal. If this is a number it is expected to be in the units of the xStart/xStop in the constructor.

polyLinePt(*theX*, *theTracPos*)
Returns a Coord.Pt from theX axis value (or EngVal) and theTracPos that is a value in DEFAULT_PLOT_UNITS, for example given by a tracValueFunction. The Coord.Pt() will be scaled by VIEW_BOX_UNITS_PER_PLOT_UNITS.

retMainPaneStart()
Returns the start Coord.Pt() for the pane where the main log goes. For and upPlot this will be pane-bottom-left, for a downPlot this will be pane-top-left.

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

class TotalDepth.util.plot.Plot.**Plot**(*theFilmCfg*, *thePresCfg*, *theScale=0*)
Defines a plot configuration. The basic architecture follows the data. The constructor takes all the static data, typically this can be obtained from the PRES and FILE tables. The dynamic (or user selected data) is passed in to plotLogPassLIS(). This is intended as a single Plot object might be used multiple times (e.g. on 'scroll up').

TITLE_FONT_FAMILY = 'Verdana'
Title font

TITLE_FONT_SIZE = 10
Title font size

LEGEND_DEPTH_PER_CURVE = Dim(0.5, 'in')
How much depth to give each legend (curve scale) at the top and bottom of the log

LEGEND_DEPTH_SPARE = Dim(0.5, 'in')
How much spare depth to give over the legend (curve scale) sections at the top and bottom of the log

LEGEND_HORIZONTAL_LINE_DEPTH_PROPORTION = 0.625
Where the curve line in the legend section appears as a proportion of LEGEND_DEPTH_PER_CURVE

LEGEND_ARROW_DISPLAY = True
Arrow heads on legend scales

LEGEND_ARROW_WIDTH = Dim(0.08, 'in')
Arrow head width on legend scales

LEGEND_ARROW_DEPTH = Dim(0.04, 'in')
Arrow head depth on legend scales

LEGEND_ARROW_WIDTH_PX = 0.75
Arrow head line width on legend scales

CURVE_LEGEND_FONT_FAMILY = 'Courier'
Legend font.

CURVE_LEGEND_FONT_SIZE = 9
Legend font family.

MICRO_MARGIN = Dim(0.04, 'in')
This is just a very small margin to provide a tiny bit of whitespace between elements in certain places

MAX_BACKUP_TRACK_CROSSING_LINES = 4
Maximum number of backup lines that can cross a single track in a single X step See the source of `_filterCrossLineList()` for an explanation.

__init__(*theFilmCfg*, *thePresCfg*, *theScale=0*)
Initialize self. See help(type(self)) for accurate signature.

xScale(*theFilmID*)
Returns the X axis scale as a number given the FILM ID.

filmIdS()
Returns an unordered list of FILM IDs.

hasDataToPlotLIS(*theLogPass*, *theFilmId*)
Returns True if a call to plotLogPassLIS() is likely to lead to some plot data being produced.

plotLogPassLIS(*theLisFile*, *theLogPass*, *theXStart*, *theXStop*, *theFilmId*, *theFpOut*, *frameStep=1*, *title=''*, *lrCONS=None*, *timerS=None*)
Plot a part of a LogPass and returns a list of Channel IDs plotted.

theLisFile The LIS File object.

theLogPass A `LogPass` object, the FrameSet will be populated here.

theXStart The start X axis position as an `EngVal`.

theXStop The stop X axis position as an `EngVal`.

theFilmId The ID of the output device from the film table

theFpOut A file path for the output SVG.

frameStep Integer number of frame steps, 1 is all frames.

title A string for the title that will appear in `LEGEND_DEPTH_SPARE`

lrCONS A `CONS` Logical Record that will be used to plot an API header in SVG.

timerS Optional `ExecTimer.ExecTimerList` for performance measurement.

TODO: If title is empty do not use the space `self.LEGEND_DEPTH_SPARE`

hasDataToPlotLAS(*theLasFile*, *theFilmId*)
Returns True if a call to plotLogPassLIS() is likely to lead to some plot data being produced.

plotLogPassLAS(*theLasFile*, *theXStart*, *theXStop*, *theFilmId*, *theFpOut*, *frameStep=1*, *title=''*, *plotHeader=False*, *timerS=None*)
Plot a part of a LogPass and returns a list of Channel IDs plotted.

theLisFile - The LIS File object.

theLogPass - A LogPass object, the FrameSet will be populated here.

theXStart - The start X axis position as an EngVal.

theXStop - The stop X axis position as an EngVal.

theFilmId - The ID of the output device from the film table

theFpOut - A file path for the output SVG.

frameStep - Integer number of frame steps, 1 is all frames.

title - A string for the title that will appear in LEGEND_DEPTH_SPARE

lrCONS - A CONS Logical Record that will be used to plot an API header in SVG.

timerS - Optional ExecTimer.ExecTimerList for performance measurement.

TODO: If title is empty do not use the space self.LEGEND_DEPTH_SPARE

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

class TotalDepth.util.plot.Plot.**PlotReadLIS**(*lrFILM*, *lrPRES*, *lrAREA=None*, *lrPIP=None*, *theScale=0*)
A subclass of Plot that is configured from FILM, PRES and (optionally) AREA, PIP Logical Records.

__init__(*lrFILM*, *lrPRES*, *lrAREA=None*, *lrPIP=None*, *theScale=0*)
Initialize self. See help(type(self)) for accurate signature.

class TotalDepth.util.plot.Plot.**PlotReadXML**(*uniqueId*, *theScale=0*)
A subclass of Plot that is configured from XML file(s) using LgFormat.

__init__(*uniqueId*, *theScale=0*)
Initialize self. See help(type(self)) for accurate signature.

TotalDepth.util.plot.PlotConstants (Plot Constants for Wireline Logs)

Some basic constants used in plotting.

Created on Jan 5, 2012

TotalDepth.util.plot.PlotConstants.**DEFAULT_PLOT_UNITS = 'in'**
Default SVG plot units

TotalDepth.util.plot.PlotConstants.**DEFAULT_PLOT_LIS_UNITS = b'IN '**
Units that we can convert to in LIS terms

TotalDepth.util.plot.PlotConstants.**VIEW_BOX_UNITS_PER_PLOT_UNITS = 96.0**
Number of pixels per standard plot unit

TotalDepth.util.plot.PlotConstants.**MarginQtrInch = Margin(left=Dim(0.25, 'in'), right=Dim(**
Definition of a quarter inch margin

TotalDepth.util.plot.PlotConstants.**STANDARD_PAPER_WIDTH = Dim(8.5, 'in')**
Standard wireline log fan paper width i.e. long side.

TotalDepth.util.plot.PlotConstants.**STANDARD_PAPER_DEPTH = Dim(6.25, 'in')**
Standard wireline log fan paper depth i.e. short side.

TotalDepth.util.plot.PlotConstants.**STANDARD_PAPER_MARGIN = Margin(left=Dim(0.25, 'in'), rig**
Standard wireline log fan paper margin.

TotalDepth.util.plot.PRESCfg (PRES Plotting Configuration)

Holds all the information to draw a curve on a plot.

TODO: Split this to:

- Generic: PRESCfg
- LIS specific: PRESCfgLIS
- XML specific: PRESCfgXML

Created on 20 Mar 2011

Examples of PRES records summaries:

```
 Table record (type 34) type: PRES

MNEM  OUTP  STAT  TRAC  CODI  DEST  MODE      FILT          LEDG          REDG
--------------------------------------------------------------------------------

SP    SP    ALLO  T1    LLIN  1     SHIF      0.500000      -80.0000       20.0000
CALI  CALI  ALLO  T1    LDAS  1     SHIF      0.500000       5.00000       15.0000
MINV  MINV  DISA  T1    LLIN  1     SHIF      0.500000       30.0000       0.00000
MNOR  MNOR  DISA  T1    LDAS  1     SHIF      0.500000       30.0000       0.00000
LLD   LLD   ALLO  T23   LDAS  1     GRAD      0.500000      0.200000       2000.00
LLDB  LLD   ALLO  T2    HDAS  1     GRAD      0.500000       2000.00       200000.
LLG   LLG   DISA  T23   LDAS  1     GRAD      0.500000      0.200000       2000.00
LLGB  LLG   DISA  T2    HDAS  1     GRAD      0.500000       2000.00       200000.
LLS   LLS   ALLO  T23   LSPO  1     GRAD      0.500000      0.200000       2000.00
LLSB  LLS   ALLO  T2    HSPO  1     GRAD      0.500000       2000.00       200000.
MSFL  MSFL  ALLO  T23   LLIN  1     GRAD      0.500000      0.200000       2000.00
11    DUMM  DISA  T1    LLIN  NEIT  NB        0.500000       0.00000       1.00000
12    DUMM  DISA  T1    LLIN  NEIT  NB        0.500000       0.00000       1.00000
13    DUMM  DISA  T1    LLIN  NEIT  NB        0.500000       0.00000       1.00000
14    DUMM  DISA  T1    LLIN  NEIT  NB        0.500000       0.00000       1.00000
15    DUMM  DISA  T1    LLIN  NEIT  NB        0.500000       0.00000       1.00000
16    DUMM  DISA  T1    LLIN  NEIT  NB        0.500000       0.00000       1.00000
17    DUMM  DISA  T1    LLIN  NEIT  NB        0.500000       0.00000       1.00000
18    DUMM  DISA  T1    LLIN  NEIT  NB        0.500000       0.00000       1.00000
19    DUMM  DISA  T1    LLIN  NEIT  NB        0.500000       0.00000       1.00000

Table record (type 34) type: PRES

MNEM  OUTP  STAT  TRAC  CODI  DEST  MODE      FILT          LEDG          REDG
--------------------------------------------------------------------------------
NPHI  NPHI  ALLO  T23   LDAS  1     SHIF      0.500000       0.450000     -0.150000
DRHO  DRHO  ALLO  T3    LSPO  1     NB        0.500000      -0.250000      0.250000
PEF   PEF   ALLO  T23   LGAP  1     SHIF      0.500000       0.00000       10.0000
SGR         DISA  T1    LLIN  1     SHIF      0.500000       0.00000       300.000
CGR         DISA  T1    LGAP  1     SHIF      0.500000       0.00000       300.000
TENS  TENS  DISA  T3    LGAP  1     SHIF      0.500000       14000.0       4000.00
CAL   CALI  ALLO  T1    LSPO  1     SHIF      0.500000       5.00000       15.0000
BS    BS    DISA  T1    LGAP  1     SHIF      0.500000       5.00000       15.0000
FFLS  FFLS  DISA  T1    LLIN  2     NB        0.500000      -0.150000      0.150000
FFSS  FFSS  DISA  T1    LDAS  2     NB        0.500000      -0.150000      0.150000
LSHV  LSHV  DISA  T3    LLIN  2     WRAP      0.500000       2150.00       2250.00
SSHV  SSHV  DISA  T3    LDAS  2     WRAP      0.500000       1950.00       2050.00
FLS   FLS   DISA  T2    LLIN  2     SHIF      0.500000       0.00000       150.000
FSS   FSS   DISA  T2    LDAS  2     SHIF      0.500000       0.00000       150.000
```

(continues on next page)

(continued from previous page)

```
RHOB  RHOB  ALLO  T23   LLIN  1     SHIF      0.500000       1.95000       2.95000
PHIX  PHIX  ALLO  T1    LLIN  1     NB        0.500000      0.500000       0.00000
```

TRAC nomenclature:

`<LH | RH> T n <m>`

Or as a regex: `re.compile(r'^(LH|RH)*T(\d)(\d)*\s*$')`

But this is handled by FILMCfg.

Example:

```
$ python3 TableHistogram.py -k --name=PRES -l40 --col=TRAC ../../../pLogicTestData/
↪LIS/
Cmd: TableHistogram.py -k --name=PRES -l40 --col=TRAC ../../../pLogicTestData/LIS/
2011-05-27 09:26:12,324 ERROR    Can not create Logical Record, error: can't convert␣
↪negative value to unsigned int
2011-05-27 09:26:12,335 ERROR    Can not create Logical Record, error: can't convert␣
↪negative value to unsigned int
2011-05-27 09:26:12,346 ERROR    Can not create Logical Record, error: can't convert␣
↪negative value to unsigned int
2011-05-27 09:26:13,086 ERROR    Can not read LIS file ../../../pLogicTestData/LIS/
↪13576.S1 with error: Can not fit integer number of frames length 120 into LR length␣
↪824, modulo 104 [indirect size 0].
2011-05-27 09:26:13,907 ERROR    Can not read LIS file ../../../pLogicTestData/LIS/
↪13610.S1 with error: Can not fit integer number of frames length 7176 into LR␣
↪length 13354, modulo 6178 [indirect size 0].
======================== Count of all table entries ======================
{"(34, b'PRES', b'TRAC', b'F1  ')": 4,
 "(34, b'PRES', b'TRAC', b'F2  ')": 4,
 "(34, b'PRES', b'TRAC', b'F3  ')": 4,
 "(34, b'PRES', b'TRAC', b'F4  ')": 3,
 "(34, b'PRES', b'TRAC', b'FD  ')": 4,
 "(34, b'PRES', b'TRAC', b'LHT1')": 4,
 "(34, b'PRES', b'TRAC', b'LHT2')": 8,
 "(34, b'PRES', b'TRAC', b'LHT3')": 59,
 "(34, b'PRES', b'TRAC', b'RHT1')": 4,
 "(34, b'PRES', b'TRAC', b'RHT2')": 22,
 "(34, b'PRES', b'TRAC', b'RHT3')": 63,
 "(34, b'PRES', b'TRAC', b'T1  ')": 2363,
 "(34, b'PRES', b'TRAC', b'T2  ')": 354,
 "(34, b'PRES', b'TRAC', b'T23 ')": 192,
 "(34, b'PRES', b'TRAC', b'T3  ')": 178,
 "(34, b'PRES', b'TRAC', b'TD  ')": 93,
 "(34, b'PRES', b'TRAC', b'XXXX')": 18}
====================== Count of all table entries END ====================
```

DEST:

```
======================== Count of all table entries ======================
{"(34, b'PRES', b'DEST', b'1   ')": 457,
 "(34, b'PRES', b'DEST', b'123 ')": 2,
 "(34, b'PRES', b'DEST', b'134 ')": 20,
 "(34, b'PRES', b'DEST', b'2   ')": 139,
 "(34, b'PRES', b'DEST', b'5   ')": 26,
 "(34, b'PRES', b'DEST', b'6   ')": 122,
 "(34, b'PRES', b'DEST', b'8   ')": 16,
```

(continues on next page)

(continued from previous page)

```
 "(34, b'PRES', b'DEST', b'A   ')": 19,
 "(34, b'PRES', b'DEST', b'ALL ')": 4,
 "(34, b'PRES', b'DEST', b'BOTH')": 774,
 "(34, b'PRES', b'DEST', b'D   ')": 63,
 "(34, b'PRES', b'DEST', b'E   ')": 60,
 "(34, b'PRES', b'DEST', b'J   ')": 52,
 "(34, b'PRES', b'DEST', b'K   ')": 13,
 "(34, b'PRES', b'DEST', b'NEIT')": 1610}
====================== Count of all table entries END ====================
```

MODE:

```
$ python3 TableHistogram.py -k --name=PRES -l40 --col=MODE ../../../pLogicTestData/
↪LIS/
Cmd: TableHistogram.py -k --name=PRES -l40 --col=MODE ../../../pLogicTestData/LIS/
2011-06-02 08:22:54,839 ERROR    Can not read LIS file ../../../pLogicTestData/LIS/
↪13576.S1 with error: Can not fit integer number of frames length 120 into LR length␣
↪824, modulo 104 [indirect size 0].
2011-06-02 08:22:55,671 ERROR    Can not read LIS file ../../../pLogicTestData/LIS/
↪13610.S1 with error: Can not fit integer number of frames length 7176 into LR␣
↪length 13354, modulo 6178 [indirect size 0].
======================== Count of all table entries ======================
{"(34, b'PRES', b'MODE', b'GRAD')": 245,
 "(34, b'PRES', b'MODE', b'NB  ')": 2329,
 "(34, b'PRES', b'MODE', b'SHIF')": 597,
 "(34, b'PRES', b'MODE', b'SWF ')": 3,
 "(34, b'PRES', b'MODE', b'VDLN')": 3,
 "(34, b'PRES', b'MODE', b'WRAP')": 186,
 "(34, b'PRES', b'MODE', b'X10 ')": 10}
====================== Count of all table entries END ====================
```

COLO:

```
======================== Count of all table entries ======================
{"(34, b'PRES', b'COLO', b'000 ')": 173,
 "(34, b'PRES', b'COLO', b'002 ')": 3,
 "(34, b'PRES', b'COLO', b'003 ')": 6,
 "(34, b'PRES', b'COLO', b'004 ')": 33,
 "(34, b'PRES', b'COLO', b'014 ')": 6,
 "(34, b'PRES', b'COLO', b'020 ')": 9,
 "(34, b'PRES', b'COLO', b'021 ')": 8,
 "(34, b'PRES', b'COLO', b'030 ')": 34,
 "(34, b'PRES', b'COLO', b'034 ')": 2,
 "(34, b'PRES', b'COLO', b'044 ')": 6,
 "(34, b'PRES', b'COLO', b'104 ')": 3,
 "(34, b'PRES', b'COLO', b'134 ')": 14,
 "(34, b'PRES', b'COLO', b'203 ')": 3,
 "(34, b'PRES', b'COLO', b'221 ')": 3,
 "(34, b'PRES', b'COLO', b'312 ')": 10,
 "(34, b'PRES', b'COLO', b'400 ')": 44,
 "(34, b'PRES', b'COLO', b'404 ')": 6,
 "(34, b'PRES', b'COLO', b'420 ')": 9,
 "(34, b'PRES', b'COLO', b'430 ')": 17,
 "(34, b'PRES', b'COLO', b'AQUA')": 3,
 "(34, b'PRES', b'COLO', b'BLAC')": 1260,
 "(34, b'PRES', b'COLO', b'BLUE')": 8,
 "(34, b'PRES', b'COLO', b'GREE')": 37,
```

(continues on next page)

(continued from previous page)

```
 "(34, b'PRES', b'COLO', b'RED ')": 12}
====================== Count of all table entries END ====================
```

What are these numbers, RGB Base 5?

Track names F1 to F4 and FD.

This seems to be the nomenclature for four track plots. For example:

```
Table record (type 34) type: FILM

MNEM  GCOD  GDEC  DEST  DSCA
-----------------------------

8     EB0   ---   PF8   D200
A     LLLL  1111  PFA   DM
E     E4E   -4-   PFE   D200
K     E4E   -4-   PFK   D500

Table record (type 34) type: PRES

MNEM  OUTP  STAT  TRAC  CODI  DEST  MODE      FILT          LEDG          REDG      ␣
↪COLO
------------------------------------------------------------------------------------
↪---
...
AX1A  AX    ALLO  F2    LLIN  A     SHIF      0.500000      -9.00000       9.00000 ␣
↪400
AY1A  AY    ALLO  F2    LSPO  A     SHIF      0.500000      -9.00000       9.00000 ␣
↪400
AZ1A  AZ    ALLO  F2    LGAP  A     NB        0.500000      -9.00000       9.00000 ␣
↪400
AN1A  ANOR  ALLO  F4    HDAS  A     NB        0.500000       9.00000       11.0000 ␣
↪420
CS1A  CS    DISA  F1    LDAS  A     NB        0.500000       0.00000       150000. ␣
↪000
FX1A  FX    ALLO  F3    LLIN  A     NB        0.500000     -0.700000      0.700000 ␣
↪003
FY1A  FY    ALLO  F3    LGAP  A     NB        0.500000     -0.700000      0.700000 ␣
↪003
FZ1A  FZ    ALLO  F3    LSPO  A     NB        0.500000     -0.700000      0.700000 ␣
↪003
FN1A  FNOR  ALLO  F4    LLIN  A     NB        0.500000      0.200000      0.700000 ␣
↪420
FI1A  FINC  ALLO  F4    LGAP  A     NB         1.00000       0.00000       90.0000 ␣
↪420
GA1A  GADZ  DISA  F1    LLIN  A     NB         1.00000      -1.00000       1.00000 ␣
↪020
GP1A  GPV   DISA  F1    LGAP  A     NB         1.00000       14.0000       16.0000 ␣
↪020
GN1A  GNV   DISA  F1    LDAS  A     NB         1.00000      -16.0000      -14.0000 ␣
↪020
GM1A  GMT   DISA  F3    HDAS  A     SHIF       1.00000       80.0000       130.000 ␣
↪020
GA2A  GAT   DISA  F2    HDAS  A     SHIF       1.00000       80.0000       130.000 ␣
↪020
SI1A  SILO  DISA  FD    HLIN  A     NB         1.00000       0.00000       20.0000 ␣
↪000
```

(continues on next page)

(continued from previous page)

```
ST1A  STIT  DISA  FD    LLIN  A     NB         1.00000       0.00000       20.0000 ␣
↪000
ST2A  STIA  ALLO  FD    LLIN  A     NB         1.00000       0.00000       20.0000 ␣
↪000
TE1A  TENS  DISA  FD    LDAS  A     WRAP      0.500000       2000.00       7000.00 ␣
↪000
...
```

exception TotalDepth.util.plot.PRESCfg.**ExceptionPRESCfg**
Specialisation of exception for this module.

exception TotalDepth.util.plot.PRESCfg.**ExceptionLineTransBase**
Specialisation of exception for LineTransBase and descendants.

exception TotalDepth.util.plot.PRESCfg.**ExceptionLineTransBaseMath**
For LineTransBase and descendants where math errors occur.

exception TotalDepth.util.plot.PRESCfg.**ExceptionPRESCfgLISRead**
Specialisation of exception for this module.

exception TotalDepth.util.plot.PRESCfg.**ExceptionPresCfg**
Specialisation of exception for PresCfg.

exception TotalDepth.util.plot.PRESCfg.**ExceptionCurveCfg**
Specialisation of exception for CurveCfg.

exception TotalDepth.util.plot.PRESCfg.**ExceptionCurveCfgCtor**
Construction exception when making a CurveCfg object or descendant.

exception TotalDepth.util.plot.PRESCfg.**ExceptionCurveCfgLISRead**
Specialisation of exception for CurveCfgLISRead and its travails.

TotalDepth.util.plot.PRESCfg.**DEFAULT_LLINE_WIDTH_PX = 0.5**
Deafult light line width in pixels

TotalDepth.util.plot.PRESCfg.**DEFAULT_HLINE_WIDTH_PX = 1.5**
Deafult heavy line width in pixels

TotalDepth.util.plot.PRESCfg.**LIS_CODI_MAP = {None: Stroke(width=0.5, colour='black', codi**
Maps LIS CODI mnemonics to a Stroke object: If either value is None an SVG attribute is not needed i.e. default SVG behaviour

TotalDepth.util.plot.PRESCfg.**LIS_COLO_MAP = {Mnem(b'AQUA'): 'aqua', Mnem(b'BLAC'): 'black',**
Maps LIS COLO mnemonics to CSS/SVG style colour specifications.

TotalDepth.util.plot.PRESCfg.**lisColo**(*theMnem*)
Returns a SVG colour as a string from a Mnem or None on failure.

TotalDepth.util.plot.PRESCfg.**coloStroke**(*theSt*, *theCm*)
Takes a Stroke object and returns a new stroke object with the colour replaced with the Color Mnem looked up, or determined from, the LIS_COLO_MAP.

TotalDepth.util.plot.PRESCfg.**BACKUP_NONE = (1, -1)**
No backup

TotalDepth.util.plot.PRESCfg.**BACKUP_ALL = (0, 0)**
Every backup i.e. 'wrap' Note: Plot.py has a way of limiting ludicrous backup lines to a sensible number; say 8

TotalDepth.util.plot.PRESCfg.**BACKUP_ONCE = (-1, 1)**
Single backup to left or right

TotalDepth.util.plot.PRESCfg.**BACKUP_TWICE = (-2, 2)**
Two backups to left or right

TotalDepth.util.plot.PRESCfg.**BACKUP_LEFT = (0, -1)**
Single backup to left only

TotalDepth.util.plot.PRESCfg.**BACKUP_RIGHT = (1, 0)**
Single backup to right only

TotalDepth.util.plot.PRESCfg.**BACKUP_FROM_MODE_MAP = {None: (0, 0), b'SHIF': (-1, 1), b'GR**
Map of backup mode to internal representation

class TotalDepth.util.plot.PRESCfg.**LineTransBase**(*leftP*, *rightP*, *leftL*, *rightL*, *backup*)
Base class for line generators.

__init__(*leftP*, *rightP*, *leftL*, *rightL*, *backup*)
Ctor with values; leftP, rightP are physical positions as numbers. leftL, rightL are logical scales as numbers. backup is a pair (left, right). Will raise a ExceptionLineTransBase if leftP >= rightP.

property leftL
The left value of the curve scale as a number.

property rightL
The right value of the curve scale as a number.

__str__()
Return str(self).

L2P(*val*)
Scale a given value to a dimension.

wrapPos(*val*)
For a given value returns a pair (wrap, pos). wrap is an integer number of times the curve is wrapped. pos is a float that is the physical plot position of the value. TODO: Benchmark this, it could be slow.

offScale(*w*)
Returns 0 if wrap integer is on scale depending on the backup setting. Returns -1 if off scale low, +1 if off scale high.

isOffScaleLeft(*w*)
True is wrap integer is off-scale low according to the backup setting.

isOffScaleRight(*w*)
True is wrap integer is off-scale high according to the backup setting.

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

class TotalDepth.util.plot.PRESCfg.**LineTransLin**(*leftP*, *rightP*, *leftL*, *rightL*, *backup=(0, 0)*)
Linear grid.

__init__(*leftP*, *rightP*, *leftL*, *rightL*, *backup=(0, 0)*)
Ctor with values; leftP, rightP are physical positions as numbers. leftL, rightL are logical scales as numbers. backup is a pair (left, right).

L2P(*val*)
Scale a given value to a dimension.

wrapPos(*val*)
For a given value returns a pair (wrap, pos). wrap is an integer number of times the curve is wrapped. pos is a float that is the physical plot position of the value. TODO: Benchmark this, it could be slow.

class TotalDepth.util.plot.PRESCfg.**LineTransLog10**(*leftP*, *rightP*, *leftL*, *rightR*, *backup=(0, 0)*)

Logrithmic grid.

__init__(*leftP*, *rightP*, *leftL*, *rightR*, *backup=(0, 0)*)
Ctor with values; leftP, rightP are physical positions as numbers. leftL, rightL are logical scales as numbers. backup is a pair (left, right).

L2P(*val*)
Scale a given value to a dimension.

wrapPos(*val*)
For a given value returns a pair (wrap, pos). wrap is an integer number of times the curve is wrapped. pos is a float that is the physical plot position of the value. TODO: Benchmark this, it could be slow.

class TotalDepth.util.plot.PRESCfg.**TrackWidthData**(*leftP*, *rightP*, *halfTrackStart*, *halfTracks*)

Used to record the physical width data of a track leftP, rightP are Coord.Dim objects. halfTrackStart, halfTracks are integers

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *leftP*, *rightP*, *halfTrackStart*, *halfTracks*)
Create new instance of TrackWidthData(leftP, rightP, halfTrackStart, halfTracks)

__repr__()
Return a nicely formatted representation string

property halfTrackStart
Alias for field number 2

property halfTracks
Alias for field number 3

property leftP
Alias for field number 0

property rightP
Alias for field number 1

class TotalDepth.util.plot.PRESCfg.**CurveCfg**

Contains the configuration of a single curve.

__init__()
Populate attribute with reasonable default values. Second stage is to set: self.mnem, self.outp, self.trac, self.dest. Optionally other properties as well.

longStr()
Returns a long descriptive string of the internal state.

tracWidthData(*theFilmID*)
Returns a TrackWidthData object for the film ID.

tracValueFunction(*theFilmID*)
Given a FILM ID (a Mnem() object) this returns a LineTransBase or derivation that describes how this curve is plotted on that film. In particular the return value will have a function wrapPos() for generating track positions from a value.

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

class TotalDepth.util.plot.PRESCfg.**CurveCfgLISRead**(*theRow*, *theFILMCfg*)

DEFAULT_MODE = b'WRAP'
Default backup mode

__init__(*theRow*, *theFILMCfg*)
Reads a LogiRec.TableRow object from a PRES table and populates a CurveCfg. theFILMCfg is expected to be a FILMCfgLISRead object. Typical row: MNEM OUTP STAT TRAC CODI DEST MODE FILT LEDG REDG ———————————————————————— NPHI NPHI ALLO T23 LDAS 1 SHIF 0.500000 0.450000 -0.150000 May raise an ExceptionCurveCfg or an IndexError. TODO: Not raise IndexError but normalise to ExceptionCurveCfg? For example have an API missingCols(...) that returns None if all present or a tuple of those missing.

class TotalDepth.util.plot.PRESCfg.**PresCfg**
Contains the configuration equivalent to a complete PRES table.

__init__()
Initialize self. See help(type(self)) for accurate signature.

add(*theCurveCfg*, *theFilmDestS*)
Adds to the IR. theCurveCfg is a CurvCfg object, theFilmDestS is a list of film destinations expanded from the FILM table e.g. if the destination is b'ALL' then all FILM destination mnemonics should be in the list.

keys()
Returns the curve mnemonics.

__len__()
Number of curves in this table.

__getitem__(*theCurvID*)
Returns the CurveCfg object corresponding to curve ID, a Mnem.Mnem object.

__contains__(*theCurvID*)
Returns True if I have an entry for the curve ID, a Mnem.Mnem object.

hasCurvesForDest(*theDest*)
Returns True if there are curves that go to theDest i.e. FILM ID.

outpChIDs(*theDest*)
Returns a list of output channel IDs for a given film destination, a Mnem.Mnem object.

outpCurveIDs(*theDest*, *theOutp*)
Returns a list of channel IDs for a given film destination and output that feeds those curves. The curve data is accessible by __getitem__(). Arguments should be Mnem.Mnem objects

usesOutpChannel(*theDest*, *theOutp*)
Returns True if this PRES table + FILM destination uses theOutp ID. Arguments should be , a Mnem.Mnem objects.

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

class TotalDepth.util.plot.PRESCfg.**PresCfgLISRead**(*theLr*, *theFILMCfg*)
Information from a complete LIS PRES table.

__init__(*theLr*, *theFILMCfg*)
Reads a LogiRec.Table object of type PRES and creates a CurveCfgLISRead for each row. theFILMCfg is expected to be a FILMCfgLISRead object.

Typical PRES table:

```
MNEM  OUTP  STAT  TRAC  CODI  DEST  MODE      FILT          LEDG          REDG
------------------------------------------------------------------------------
↪-----
NPHI  NPHI  ALLO  T23   LDAS  1     SHIF      0.500000      0.450000     -0.
↪150000
DRHO  DRHO  ALLO  T3    LSPO  1     NB        0.500000     -0.250000      0.
↪250000
PEF   PEF   ALLO  T23   LGAP  1     SHIF      0.500000       0.00000       10.
↪0000
```

TotalDepth.util.plot.PRESCfgXML (PRES Plotting Configuration from XML files)

Creates PRES configurations from LgFormat XML files.

Created on Dec 16, 2011

exception TotalDepth.util.plot.PRESCfgXML.**ExceptionPRESCfgXML**
Specialisation of exception for PRESCfgXML module.

exception TotalDepth.util.plot.PRESCfgXML.**ExceptionCurveCfgXMLRead**
Specialisation of exception for CurveCfgXMLRead module.

exception TotalDepth.util.plot.PRESCfgXML.**ExceptionPresCfgXMLRead**
Specialisation of exception for PresCfgXMLRead module.

TotalDepth.util.plot.PRESCfgXML.**XML_CODI_MAP = {None: Stroke(width=0.5, colour='black', c**
Maps LgFormat mnemonics to a Stroke object: If either value is None an SVG attribute is not needed i.e. default SVG behaviour

TotalDepth.util.plot.PRESCfgXML.**BACKUP_FROM_MODE_MAP = {None: (0, 0), 'LG_LEFT_WRAPPED':**
Maps LgFormat backup specifications to an internal representation Taken from the <WrapMode> element.

TotalDepth.util.plot.PRESCfgXML.**BACKUP_FROM_COUNT_MAP = {None: (0, 0), '1': (-1, 1), '2'**
Fallback mapping LgFormat backup specifications to an internal representation Taken from the <WrapCount> element.

class TotalDepth.util.plot.PRESCfgXML.**CurveCfgXMLRead**(*e*, *theTrac*, *theFILMCfg*)
Represents a single curve from an XML file specification

TRAC_XML_UNIQUEID_TO_PRES = {'DepthTrack': b'TD ', 'Track2': b'T2 ', 'Track3': b'T3
First column is observed tracks in the XML, note capitalisation inconsistencies. Second column is LIS DEST equivalent

__init__(*e*, *theTrac*, *theFILMCfg*)
Creates a single CurveCfg object from an XML LgCurve element and populates a CurveCfg. e is the root LgCurve element. theTrac is the track name that this is being plotted on. theFILMCfg is expected to be a FilmCfgXMLRead object.

Example:

```
<LgCurve UniqueId="ROP5">
    <ChannelName>ROP5</ChannelName>
    <Color>0000FF</Color>
    <LeftLimit>500</LeftLimit>
    <RightLimit>0</RightLimit>
    <LineStyle>LG_DASH_LINE</LineStyle>
    <Thickness>1.75</Thickness>
    <WrapMode>LG_LEFT_WRAPPED</WrapMode>
```

(continues on next page)

(continued from previous page)

```
</LgCurve>
Or:
<LgCurve UniqueId="PSR">
    <ChannelName>PSR</ChannelName>
    <Color>00C000</Color>
    <LeftLimit>0.2</LeftLimit>
    <RightLimit>2000</RightLimit>
    <Thickness>2</Thickness>
    <Transform>LG_LOGARITHMIC</Transform>
    <WrapCount>0</WrapCount>
</LgCurve>
```

class TotalDepth.util.plot.PRESCfgXML.**PresCfgXMLRead**(*theFILMCfg*, *theUniqueId*)
Extracts all curve presentation information from a single XML file.

__init__(*theFILMCfg*, *theUniqueId*)
Reads a XML and creates a CurveCfgXMLRead for each LgFormat/LgTrack/LgCurve element.

theFILMCfg is expected to be a FILMCfgXMLRead object.

TotalDepth.util.plot.Stroke (Plot Pen Stroke)

Defines how the stroke of a pen is represented on a plot.

Created on 7 Mar 2011

class TotalDepth.util.plot.Stroke.**Stroke**(*width*, *colour*, *coding*, *opacity*)
Stroke basic properties

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *width*, *colour*, *coding*, *opacity*)
Create new instance of Stroke(width, colour, coding, opacity)

__repr__()
Return a nicely formatted representation string

property coding
Alias for field number 2

property colour
Alias for field number 1

property opacity
Alias for field number 3

property width
Alias for field number 0

TotalDepth.util.plot.Stroke.**StrokeBlackSolid = Stroke(width='1', colour='black', coding=Nor**
Prototypical black solid stroke Usage: StrokeBlackSolid._replace(width=2.0)

TotalDepth.util.plot.Stroke.**retSVGAttrsFromStroke**(*stroke*)
Returns SVG attributes as a dictionary from a Stroke() object.

TotalDepth.util.plot.SVGWriter (SVG Writer Module)

An SVG writer.

TODO: Add a float format to reduce the size of the SVG file.

exception TotalDepth.util.plot.SVGWriter.**ExceptionSVGWriter**
Exception class for SVGWriter.

TotalDepth.util.plot.SVGWriter.**DEFAULT_VALUE_FORMAT = '{:.3f}'**
Defaults format for points that are specified in inches or such like

TotalDepth.util.plot.SVGWriter.**DEFAULT_VALUE_FORMAT_POINTS = '{:.1f}'**
Defaults format for points that are specified in pixels

TotalDepth.util.plot.SVGWriter.**dimToTxt**(*theDim*)
Converts a Coord.Dim() object to text for SVG units.

class TotalDepth.util.plot.SVGWriter.**SVGWriter**(*theFile*, *theViewPort*, *rootAttrs=None*)

__init__(*theFile*, *theViewPort*, *rootAttrs=None*)
Initialise the stream with a file and Coord.Box() object. The view port units must be the same for width and depth.

__enter__()
Context manager support.

class TotalDepth.util.plot.SVGWriter.**SVGGroup**(*theXmlStream*, *attrs=None*)
A group element in SVG.

__init__(*theXmlStream*, *attrs=None*)
Initialise the group with a stream.

See: http://www.w3.org/TR/2003/REC-SVG11-20030114/struct.html#GElement

Sadly we can't use `**kwargs` because of Python restrictions on keyword names. `stroke-width` is not a valid keyword argument (although `stroke_width` would be). So instead we pass in an optional dictionary {string : string, ... }

class TotalDepth.util.plot.SVGWriter.**SVGRect**(*theXmlStream*, *thePoint*, *theBox*, *attrs=None*)
A rectangle in SVG. Initialise the rectangle with a stream, a Coord.Pt() and a Coord.Box() objects.

See: http://www.w3.org/TR/2003/REC-SVG11-20030114/shapes.html#RectElement

Typical attributes:

```
{'fill' : "blue", 'stroke' : "black", 'stroke-width' : "2", }
```

__init__(*theXmlStream*, *thePoint*, *theBox*, *attrs=None*)
Initialise the rectangle with a stream, a Coord.Pt() and a Coord.Box() objects.

See: http://www.w3.org/TR/2003/REC-SVG11-20030114/shapes.html#RectElement

Typical attributes:

```
{'fill' : "blue", 'stroke' : "black", 'stroke-width' : "2", }
```

class TotalDepth.util.plot.SVGWriter.**SVGCircle**(*theXmlStream*, *thePoint*, *theRadius*, *attrs=None*)
A circle in SVG.Initialise the circle with a stream, a Coord.Pt() and a Coord.Dim() objects.

See: http://www.w3.org/TR/2003/REC-SVG11-20030114/shapes.html#CircleElement

__init__(*theXmlStream*, *thePoint*, *theRadius*, *attrs=None*)
Initialise the circle with a stream, a Coord.Pt() and a Coord.Dim() objects.

class TotalDepth.util.plot.SVGWriter.**SVGElipse**(*theXmlStream*, *ptFrom*, *theRadX*, *theRadY*, *attrs=None*)
An elipse in SVG.Initialise the circle with a stream, a Coord.Pt() and a Coord.Dim() objects.

See: http://www.w3.org/TR/2003/REC-SVG11-20030114/shapes.html#EllipseElement

__init__(*theXmlStream*, *ptFrom*, *theRadX*, *theRadY*, *attrs=None*)
Initialise the circle with a stream, a Coord.Pt() and a Coord.Dim() objects.

class TotalDepth.util.plot.SVGWriter.**SVGLine**(*theXmlStream*, *ptFrom*, *ptTo*, *attrs=None*)
A rectangle in SVG. Initialise the line with a stream, and two Coord.Pt() objects.

See: http://www.w3.org/TR/2003/REC-SVG11-20030114/shapes.html#LineElement

__init__(*theXmlStream*, *ptFrom*, *ptTo*, *attrs=None*)
Initialise the line with a stream, and two Coord.Pt() objects.

class TotalDepth.util.plot.SVGWriter.**SVGPointList**(*theXmlStream*, *name*, *pointS*, *attrs*)
An abstract class that takes a list of points, derived by polyline and polygon.

Initialise the element with a stream, a name, and a list of Coord.Pt() objects.

NOTE: The units of the points are ignored, it is up to the caller to convert them to the User Coordinate System.

TODO: Make the caller supply points as numbers not Coord.Pt as this may be faster??? e.g.:

```
FMT_PAIR_STR = FMT_STR + ',' + FMT_STR
' '.join([self.FMT_PAIR_STR.format(x, y) for x,y in pointS])
```

__init__(*theXmlStream*, *name*, *pointS*, *attrs*)
Initialise the element with a stream, a name, and a list of Coord.Pt() objects. NOTE: The units of the points are ignored, it is up to the caller to convert them to the User Coordinate System. TODO: Make the caller supply points as numbers not Coord.Pt as this may be faster??? e.g. FMT_PAIR_STR = FMT_STR + ',' + FMT_STR ' '.join([self.FMT_PAIR_STR.format(x, y) for x,y in pointS])

class TotalDepth.util.plot.SVGWriter.**SVGPolyline**(*theXmlStream*, *pointS*, *attrs=None*)
A polyline in SVG. Initialise the polyline with a stream, and a list of Coord.Pt() objects.

NOTE: The units of the points are ignored, it is up to the caller to convert them to the User Coordinate System.

See: http://www.w3.org/TR/2003/REC-SVG11-20030114/shapes.html#PolylineElement

__init__(*theXmlStream*, *pointS*, *attrs=None*)
Initialise the polyline with a stream, and a list of Coord.Pt() objects. NOTE: The units of the points are ignored, it is up to the caller to convert them to the User Coordinate System.

class TotalDepth.util.plot.SVGWriter.**SVGPolygon**(*theXmlStream*, *pointS*, *attrs=None*)
A polygon in SVG. Initialise the polygon with a stream, and a list of Coord.Pt() objects.

NOTE: The units of the points are ignored, it is up to the caller to convert them to the User Coordinate System.

See: http://www.w3.org/TR/2003/REC-SVG11-20030114/shapes.html#PolygonElement

__init__(*theXmlStream*, *pointS*, *attrs=None*)
Initialise the polygon with a stream, and a list of Coord.Pt() objects. NOTE: The units of the points are ignored, it is up to the caller to convert them to the User Coordinate System.

class TotalDepth.util.plot.SVGWriter.**SVGText**(*theXmlStream*, *thePoint*, *theFont*, *theSize*, *attrs=None*)
Text in SVG. Initialise the text with a stream, a Coord.Pt() and font as a string and size as an integer. If thePoint is None then no location will be specified (for example for use inside a <defs> element).

See: http://www.w3.org/TR/2003/REC-SVG11-20030114/text.html#TextElement

__init__(*theXmlStream*, *thePoint*, *theFont*, *theSize*, *attrs=None*)
Initialise the text with a stream, a Coord.Pt() and font as a string and size as an integer. If thePoint is None then no location will be specified (for example for use inside a <defs> element.

Examples

All these examples assume these imports:

```
import io
from TotalDepth.util import XmlWrite
from TotalDepth.util.plot import SVGWriter
from TotalDepth.util.plot import Coord
```

Construction

Writing to an in-memory file:

```
f = io.StringIO()
vp = Coord.Box(
    Coord.Dim(100, 'mm'),
    Coord.Dim(20, 'mm'),
)
with SVGWriter.SVGWriter(myF, myViewPort):
    pass
print(myF.getvalue())
# Prints:
<?xml version='1.0' encoding="utf-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/
↪DTD/svg11.dtd">
<svg height="20.000mm" version="1.1" width="100.000mm" xmlns="http://www.w3.org/2000/
↪svg"/>
```

Writing Objects to SVG

Writing a rectangles to a stream:

```
myF = io.StringIO()
vp = Coord.Box(Coord.Dim(5, 'cm'), Coord.Dim(4, 'cm'))
with SVGWriter.SVGWriter(myF, vp) as xS:
    with XmlWrite.Element(xS, 'desc'):
        xS.characters('A couple of rectangles')
    myPt = Coord.Pt(Coord.Dim(0.5, 'cm'), Coord.Dim(0.5, 'cm'))
    myBx = Coord.Box(Coord.Dim(2.0, 'cm'), Coord.Dim(1.0, 'cm'))
    with SVGWriter.SVGRect(xS, myPt, myBx):
        pass
    myPt = Coord.Pt(Coord.Dim(0.01, 'cm'), Coord.Dim(0.01, 'cm'))
    myBx = Coord.Box(Coord.Dim(4.98, 'cm'), Coord.Dim(3.98, 'cm'))
    with SVGWriter.SVGRect(xS, myPt, myBx, attrs= {'fill' : "none", 'stroke' : "blue",
↪ 'stroke-width' : ".02cm"}):
        pass
print(myF.getvalue())
```

(continues on next page)

(continued from previous page)

```
# Prints:
<?xml version='1.0' encoding="utf-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/
↪DTD/svg11.dtd">
<svg height="4.000cm" version="1.1" width="5.000cm" xmlns="http://www.w3.org/2000/svg
↪">
    <desc>A couple of rectangles</desc>
    <rect height="1.000cm" width="2.000cm" x="0.500cm" y="0.500cm"/>
    <rect fill="none" height="3.980cm" stroke="blue" stroke-width=".02cm" width="4.
↪980cm" x="0.010cm" y="0.010cm"/>
</svg>
```

TotalDepth.util.plot.Track (Track Module)

Created on 28 Feb 2011

Plotting LIS data requires these components: * A PlotConfig object * A data set (e.g. a LogPass with a FrameSet). * An output driver (e.g. screen, print PDF, web SVG).

User creates a PlotConfig (this reflects a PRES table). This is reusable.

User specifies a data set (from, to, channels etc.). e.g. Invoke LogPass.setFrameSet(File, theFrameSlice=theFrameSlice, theChList=theChList)

User says 'plot this data set with this configuration to this output device'. e.g. Plot(PlotConfig, LogPass, PlotDevice) Plot uses LogPass.frameSet.genChScValues(ch, sc) to plot individual curves.

Lacunae

Area plotting. Caching (e.g. SVG fragments - is this worth it?)

PlotConfig

PlotTracks

Typically a three track (+depth) have these dimensions in inches:

Track	Left	Right	Width
1	0	2.4	2.4
Depth	2.4	3.2	0.8
2	3.2	5.6	2.4
3	5.6	8.0	2.4

Track names can be split (e.g. LHT1 is left hand track 1) or merged (T23 is spread across tracks two and three).

Examples of PRES and FILM records:

```
 Table record (type 34) type: PRES

MNEM  OUTP  STAT  TRAC  CODI  DEST  MODE      FILT          LEDG          REDG
--------------------------------------------------------------------------------

SP    SP    ALLO  T1    LLIN  1     SHIF      0.500000      -80.0000       20.0000
```

(continues on next page)

(continued from previous page)

```
CALI  CALI  ALLO  T1    LDAS  1     SHIF      0.500000       5.00000       15.0000
MINV  MINV  DISA  T1    LLIN  1     SHIF      0.500000       30.0000       0.00000
MNOR  MNOR  DISA  T1    LDAS  1     SHIF      0.500000       30.0000       0.00000
LLD   LLD   ALLO  T23   LDAS  1     GRAD      0.500000      0.200000       2000.00
LLDB  LLD   ALLO  T2    HDAS  1     GRAD      0.500000       2000.00       200000.
LLG   LLG   DISA  T23   LDAS  1     GRAD      0.500000      0.200000       2000.00
LLGB  LLG   DISA  T2    HDAS  1     GRAD      0.500000       2000.00       200000.
LLS   LLS   ALLO  T23   LSPO  1     GRAD      0.500000      0.200000       2000.00
LLSB  LLS   ALLO  T2    HSPO  1     GRAD      0.500000       2000.00       200000.
MSFL  MSFL  ALLO  T23   LLIN  1     GRAD      0.500000      0.200000       2000.00

Table record (type 34) type: FILM

MNEM  GCOD  GDEC  DEST  DSCA
------------------------------

1     E20   -4--  PF1   D200
2     EEE   ----  PF2   D200

Table record (type 34) type: PRES

MNEM  OUTP  STAT  TRAC  CODI  DEST  MODE      FILT          LEDG          REDG
------------------------------------------------------------------------------

NPHI  NPHI  ALLO  T23   LDAS  1     SHIF      0.500000      0.450000     -0.150000
DRHO  DRHO  ALLO  T3    LSPO  1     NB        0.500000     -0.250000      0.250000
PEF   PEF   ALLO  T23   LGAP  1     SHIF      0.500000       0.00000       10.0000
SGR         DISA  T1    LLIN  1     SHIF      0.500000       0.00000       300.000
CGR         DISA  T1    LGAP  1     SHIF      0.500000       0.00000       300.000
TENS  TENS  DISA  T3    LGAP  1     SHIF      0.500000       14000.0       4000.00
CAL   CALI  ALLO  T1    LSPO  1     SHIF      0.500000       5.00000       15.0000
BS    BS    DISA  T1    LGAP  1     SHIF      0.500000       5.00000       15.0000
FFLS  FFLS  DISA  T1    LLIN  2     NB        0.500000     -0.150000      0.150000
FFSS  FFSS  DISA  T1    LDAS  2     NB        0.500000     -0.150000      0.150000
LSHV  LSHV  DISA  T3    LLIN  2     WRAP      0.500000       2150.00       2250.00
SSHV  SSHV  DISA  T3    LDAS  2     WRAP      0.500000       1950.00       2050.00
FLS   FLS   DISA  T2    LLIN  2     SHIF      0.500000       0.00000       150.000
FSS   FSS   DISA  T2    LDAS  2     SHIF      0.500000       0.00000       150.000
RHOB  RHOB  ALLO  T23   LLIN  1     SHIF      0.500000       1.95000       2.95000
PHIX  PHIX  ALLO  T1    LLIN  1     NB        0.500000      0.500000       0.00000

Table record (type 34) type: FILM

MNEM  GCOD  GDEC  DEST  DSCA
------------------------------

1     EEE   ----  PF2   D200
2     EEE   ----  PF1   DM
```

Other FILM Table Examples:

```
Table record (type 34) type: FILM

MNEM  GCOD  GDEC  DEST  DSCA
------------------------------
1     EEB   ----  PF1   D200
2     EEB   ----  PF2   DM
```

(continues on next page)

(continued from previous page)

```
Table record (type 34) type: FILM

MNEM  GCOD  GDEC  DEST  DSCA
------------------------------
1     E20   -4--  PF1   D200
2     EEE   ----  PF2   D200

MNEM  GCOD  GDEC  DEST  DSCA
------------------------------
1     EEE   ----  PF1   D200
2     E1E   -4-   PF2   D200

Table record (type 34) type: FILM

MNEM  GCOD  GDEC  DEST  DSCA
------------------------------

D     E3E   -3-   PFD   D200
E     E3E   -3-   PFE   D500
5     EB0   ---   PF5   D200
6     EEB   ---   PF6   D200

Table record (type 34) type: FILM

MNEM  GCOD  GDEC  DEST  DSCA
------------------------------

8     EB0   ---   PF8   D200
A     LLLL  1111  PFA   DM
E     E4E   -4-   PFE   D200
K     E4E   -4-   PFK   D500

Table record (type 34) type: FILM

MNEM      CINT      GCOD  GDEC  DEST  DSCA
-----------------------------------------

1          0.00000  E2E   -2-   PF1   D200
2          0.00000  E2E   -1-   PF2   D500
3          0.00000  EEE   ----  NEIT  S5
4          0.00000  EEE   ----  NEIT  S5

Table record (type 34) type: FILM

MNEM      CINT      GCOD  GDEC  DEST  DSCA
-----------------------------------------

1          300.000  E2E   -2-   PF1   D200
2          300.000  E2E   -2-   PF2   D500
3          300.000  EEE   ----  NEIT  S5
4          300.000  EEE   ----  NEIT  S5

Table record (type 34) type: FILM

MNEM      CINT      GCOD  GDEC  DEST  DSCA
-----------------------------------------
```

(continues on next page)

(continued from previous page)

```
1          50.0000  EEE   ----  PF1   D200
2          0.00000  EEE   ----  PF2   D200
3          0.00000  EEE   ----  NEIT  D200
4          0.00000  EEE   ----  NEIT  D200
```

Minimal, but not complete interpretation:

```
Ignore GDEC as dupe.
E - Equi-spaced (linear).
n - Log with number of decades.
B - Blank.
L - ?
```

What to do with 0 (continuation?). Examples: E20 -4– means 4 decades over track 23.

exception TotalDepth.util.plot.Track.**ExceptionTotalDepthLISPlotTrack**
Exception for plotting tracks.

TotalDepth.util.plot.Track.**genLinear10**(*l*, *r*)
Generate a linear series of 10 track grid lines as (Stroke, position).

TotalDepth.util.plot.Track.**genLog10**(*l*, *r*, *cycles=1*, *start=1*)
Generate a log10 series of track grid lines as (Stroke, position). cycles is the number of log cycles to split the track up into. start is the start position of the scale e.g. 2 for common resistivity curves. So cycles=2, start=2 would give log grid 2 to 20000 i.e. to plot resistivity in the range 0.2 to 2000.

TotalDepth.util.plot.Track.**genLog10Decade2**(*l*, *r*, *, *cycles=2*, *start=1*)
Generator for 2 decades of log base 10

TotalDepth.util.plot.Track.**genLog10Decade3**(*l*, *r*, *, *cycles=3*, *start=1*)
Generator for 3 decades of log base 10

TotalDepth.util.plot.Track.**genLog10Decade4**(*l*, *r*, *, *cycles=4*, *start=1*)
Generator for 4 decades of log base 10

TotalDepth.util.plot.Track.**genLog10Decade5**(*l*, *r*, *, *cycles=5*, *start=1*)
Generator for 5 decades of log base 10

TotalDepth.util.plot.Track.**genLog10Decade1Start2**(*l*, *r*, *, *cycles=1*, *start=2*)
Generator for 1 decade of log base 10 starting at 2

TotalDepth.util.plot.Track.**genLog10Decade2Start2**(*l*, *r*, *, *cycles=2*, *start=2*)
Generator for 2 decades of log base 10 starting at 2

class TotalDepth.util.plot.Track.**Track**(*leftPos*, *rightPos*, *gridGn*, *plotXLines=True*, *plotXAlpha=False*)
Class that represents a single track. The track, as a structural graphical element, is merely a grid. The actual curves are plotted on panes that are independent from a single track (can span multiple tracks for example).

leftPos - A Coord.Dim() object that is the left edge.

rightPos- A Coord.Dim() object that is the right edge.

gridGn - A generator of line positions, None for blank track.

plotXLines - If True then plot X Axis information (depth lines) in this track.

plotXAlpha - If True then plot X Axis information (depth numbers) in this track.

DEPTH_PER_CH = Dim(0.25, 'in')
Space for plotting the scale for each curve

__init__(*leftPos*, *rightPos*, *gridGn*, *plotXLines=True*, *plotXAlpha=False*)
The track, as a structural graphical element, is merely a grid. The actual curves are plotted on panes that are independent from a single track (can span multiple tracks for example). leftPos - A Coord.Dim() object that is the left edge. rightPos- A Coord.Dim() object that is the right edge. gridGn - A generator of line positions, None for blank track. plotXLines - If True then plot X Axis information (depth lines) in this track. plotXAlpha - If True then plot X Axis information (depth numbers) in this track.

__str__()
Return str(self).

property left
The left edge as a Coord.Dim().

property right
The left edge as a Coord.Dim().

property hasGrid
True if ther is a grid to be generated for this track.

plotSVG(*topLeft*, *depth*, *theSVGWriter*)
Plot the track gridlines. topLeft - A Coord.Pt() object that is the top left of the canvas. depth - A Coord.Dim() object that is the total depth of the grid below topLeft. drive - The plot driver.

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

TotalDepth.util.plot.XGrid (X Axis Gird Line Module)

Provides plotting support for the X axis grid i.e. Depth grid lines and text.

Created on 28 Feb 2011

TODO: Remove APIs that are not used by Plot or anything. Plot only appears to use genXAxisRange() and genXAxisTextRange().

exception TotalDepth.util.plot.XGrid.**ExceptionPlotXGrid**
Exception for plotting.

TotalDepth.util.plot.XGrid.**StrokeGreySolid = Stroke(width='1', colour='grey', coding=None,**
Definition of a grey solid stroke Usage: StrokeBlackGrey._replace(width=2.0)

class TotalDepth.util.plot.XGrid.**XGrid**(*scale*)
Class that can generate depth line grid and alphanumeric values. Constructed with integer scale.

DEFAULT_INTERVAL_MAP = {1: Stroke(width=0.5, colour='black', coding=None, opacity=1.0
Default for unknown units and scale, this is basically like simple graph paper

DEFAULT_INTERVAL_TEXT = 100
Default position for text on the X axis

__init__(*scale*)
Constructor with integer scale. We make scale a constructor argument as we know that up front. We don't necessarily know the X units.

genXAxisRange(*evFrom*, *evTo*)
Generates a bounded series of X axis line plot positions as (Dim(), Stroke()). evFrom and evTo and EngVal objects.

genXPosStroke(*xFrom*, *xInc*, *units*)
Generates unbounded series of X line positions as (Dim(), Stroke()).

xFrom - The starting value as a float, positions may not include this if fractional. First x position generated will be math.ceil() if xInc, math.floor() otherwise.

xInc - A boolean, True if X increases.

units - Units of X axis e.g. b'FEET'.

genEventsUnits (*xFrom*, *xInc*, *units*)
Generates events from/to in units.

genXAxisTextRange (*evFrom*, *evTo*)
Generates a bounded series of X axis line plot positions as (Dim(), value). evFrom and evTo and EngVal objects.

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

TotalDepth.util.plot.XMLCfg (XML Plot Description Low-level support)

Provides low level support for XML plot configurations

Created on Dec 13, 2011

exception TotalDepth.util.plot.XMLCfg.**ExceptionXMLCfg**
Exception class for this module.

exception TotalDepth.util.plot.XMLCfg.**ExceptionXMLCfgWrongRootElem**
Exception when the root element is wrong.

exception TotalDepth.util.plot.XMLCfg.**ExceptionXMLCfgMissingElem**
Exception when the is a missing mandatory element.

exception TotalDepth.util.plot.XMLCfg.**ExceptionXMLCfgNoContent**
Exception when the is missing content.

class TotalDepth.util.plot.XMLCfg.**LgXMLBase**
Base class for XML functionality that can be used by both FILM and PRES XML classes.

str (*e*, *name*, *default=None*)
Returns the text in a single child element or None.

bool (*e*, *name*, *default=False*)
Returns the text in a single child element converting 1/0 to True/False.

int (*e*, *name*, *default=0*)
Returns the text in a single child element as an integer defaulting to 0.

float (*e*, *name*, *default=0.0*)
Returns the text in a single child element as a float.

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

TotalDepth.util.plot.XMLMatches (XML Plot Description Discovery)

This looks at the available XML LgFormats and works out what curves of a LogPass or LASFile can be plotted by each one.

Created on Jan 21, 2012

`TotalDepth.util.plot.XMLMatches.`**`fileCurveMap`**(*theLpOrLasFile*, *directory=None*)
Returns a map of `{FilmID : [OUTP, ...], ...}` which is a list of OUTP in theLpOrLasFile that could be plotted with that film ID.

`TotalDepth.util.plot.XMLMatches.`**`fileCurveMapFromFILM`**(*theLpOrLasFile*, *theFilmCfg*)
Returns a map of `{FilmID : [OUTP, ...], ...}` which is a list of OUTP in theLpOrLasFile that could be plotted with that film ID.

1.10 TODO's

TotalDepth is work-in-progress. This is a gathering place for features that are would be nice to have in future releases. Priority numbers are 0 (not going to be done) and >0 which is an ever more important priority. Work is the estimated amount of work from 1 upwards.

If you find a bug or need a feature then raise an issue with TotalDepth.

TotalDepth is currently at **Beta**, development version 0.4.0rc0, release version 0.4.0.

1.10.1 TotalDepth Improvements (General)

Table 6: **TotalDepth Improvements (General)**

ID	Description	Priority	Work	Status
TD.Cython	Remove Cython as a dependency. Merge C++ replacement code.	1	1	
TD.SQL	Extract data to database.	1	3	
TD.GEO	Extract Latitude and Longitude as trustworthy metadata.	1	3	
TD.svfs	Merge Sparse Virtual File System C/C++ code.	1	4	
TD.test.slow	Move slow LIS test to tests/integration/ or mark as @slow.	1	1	DONE: v0.3.0
TD.test.benchmark	Move benchmarking tests to benchmark/.	1	2	

1.10.2 LAS Improvements

Table 7: **LAS Improvements**

ID	Description	Priority	Work	Status
LAS.fast	Merge the fast array parser with ~50x performance.	2	3	
LAS.zip	Read directly from .zip files.	1	1	
LAS.v3	Support version 3.0. However this is barely used by the industry.	0	N/A	
LAS.merge_O_P	Merge `~O` section into `~P` if correct format.	1	1	
LAS.consist	Consistency checking of mutual data such as STRT/STOP/STEP.	1	1	

1.10.3 LIS Improvements

Table 8: **LIS Improvements**

ID	Description	Priority	Work	Status
LIS.HDT	Expand/contract 'sub-channels' to actual channels and use the universal Frame Array.	2	3	
LIS.XNAM	Full XNAM direct support for LIS-A.	0	3	No. These only occur in a small number from a minority producer using specialised software. They do not occur in mainstream LIS files.
LIS.index_c	Merge fast indexer in C for 100x performance.	2	3	
LIS.index_inline	Insert or append binary representation of the index to the LIS file.	1	3	
LIS.rc_over	Check Rep Code overflow/underflow on write.	1	1	
LIS.test.slow	Move slow LIS test to tests/integration/ or mark as @slow.	1	1	DONE: v0.3.0

1.10.4 RP66V1 Improvements

Table 9: **RP66V1 Improvements**

ID	Description	Priority	Work	Status
RP66V1.test.benchmark	Write benchmarking tests in `benchmark/`.	2	2	
RP66V1.index_0	Add the multi-level index code implemented in C/C++ that is much faster and smaller. See 2019-11-12.	2	3	
RP66V1.index_1	Mid level index implemented in C/C++.	1	3	
RP66V1.RepCode	Add in the code that does Representation Code conversion in C/C++.	1	3	
RP66V1.numpy.read	Directly populate the Frame Array in Numpy from C/C++. See also Frame.buffer.	1	3	
RP66V1.units	Conformance of unit conversion with the RP66V1 and, possibly, RP66V2 standard. NOTE: The RP66V2 standard is expanded on RP66V1 but barely used. Many producers deviate from these standards in any case.	1	2	
RP66V1.plot	Plot RP66V1 files. See Plot.spec.	1	3	
RP66V1.fail	When a file deviates from the standard then the user can specify what deviations are acceptable. Examples: UNITS Rep Code, multiple ORIGIN and CHANNEL records.	1	2	

1.10.5 Plotting Improvements

Table 10: **Plotting Improvements**

ID	Description	Priority	Work	Status
Plot.spec	There is quite a lot of technical debt built up since we added LgFormat support, this area needs a review. Implement the XML design.	2	4	
Plot.head	Header: Some mud parameters being dropped.	1	2	
Plot.perf	Benchmarks to characterise execution time and profiling.	1	3	
Plot.cXML	Integrate the existing XML writer written in C for x4 speedup.	1	3	
Plot.hover	Display values when hovering over curves in SVG.	1	3	
Plot.PDF	PDF output of plots. Probably use reportlab.	1	4	

1.10.6 File Formats

Table 11: **File Format Support**

ID	Description	Priority	Work	Status
Format.RP66V2	Unused by the industry.	0	N/A	
Format.WellLogML	Unused by the industry.	0	N/A	
Format.ATLAS_BIT	Legacy format.	1	1	DONE v0.4.0
Format.SEGY	Other FOSS projects specialise in this.	0	N/A	
Format.SEGD	Used at all?	0	N/A	
Format.DAT	An informal format used for mud logs.	1	1	DONE v0.4.0

1.10.7 Frame Array Improvements

Table 12: **Frame Array Improvements**

ID	Description	Priority	Work	Status
Frame.common	Common Frame Array code for all file formats.	1	3	
Frame.buffer	Implement frame processing in C++ using the buffer protocol. Also shared memory with multi-processing.	1	3	

1.11 Licence

1.11.1 OpenSource Licence

TotalDepth is released under GPL 2.:

```
                    GNU GENERAL PUBLIC LICENSE
                       Version 2, June 1991

 Copyright (C) 1989, 1991 Free Software Foundation, Inc.,
 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                            Preamble

  The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users.  This
General Public License applies to most of the Free Software
Foundation's software and to any other program whose authors commit to
using it.  (Some other Free Software Foundation software is covered by
the GNU Lesser General Public License instead.)  You can apply it to
your programs, too.
```

(continues on next page)

(continued from previous page)

```
  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
this service if you wish), that you receive source code or can get it
if you want it, that you can change the software or use pieces of it
in new free programs; and that you know you can do these things.

  To protect your rights, we need to make restrictions that forbid
anyone to deny you these rights or to ask you to surrender the rights.
These restrictions translate to certain responsibilities for you if you
distribute copies of the software, or if you modify it.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must give the recipients all the rights that
you have.  You must make sure that they, too, receive or can get the
source code.  And you must show them these terms so they know their
rights.

  We protect your rights with two steps: (1) copyright the software, and
(2) offer you this license which gives you legal permission to copy,
distribute and/or modify the software.

  Also, for each author's protection and ours, we want to make certain
that everyone understands that there is no warranty for this free
software.  If the software is modified by someone else and passed on, we
want its recipients to know that what they have is not the original, so
that any problems introduced by others will not reflect on the original
authors' reputations.

  Finally, any free program is threatened constantly by software
patents.  We wish to avoid the danger that redistributors of a free
program will individually obtain patent licenses, in effect making the
program proprietary.  To prevent this, we have made it clear that any
patent must be licensed for everyone's free use or not licensed at all.

  The precise terms and conditions for copying, distribution and
modification follow.

                    GNU GENERAL PUBLIC LICENSE
   TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

  0. This License applies to any program or other work which contains
a notice placed by the copyright holder saying it may be distributed
under the terms of this General Public License.  The "Program", below,
refers to any such program or work, and a "work based on the Program"
means either the Program or any derivative work under copyright law:
that is to say, a work containing the Program or a portion of it,
either verbatim or with modifications and/or translated into another
language.  (Hereinafter, translation is included without limitation in
the term "modification".)  Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not
covered by this License; they are outside its scope.  The act of
running the Program is not restricted, and the output from the Program
is covered only if its contents constitute a work based on the
Program (independent of having been made by running the Program).
```

(continues on next page)

(continued from previous page)

```
Whether that is true depends on what the Program does.

  1. You may copy and distribute verbatim copies of the Program's
source code as you receive it, in any medium, provided that you
conspicuously and appropriately publish on each copy an appropriate
copyright notice and disclaimer of warranty; keep intact all the
notices that refer to this License and to the absence of any warranty;
and give any other recipients of the Program a copy of this License
along with the Program.

You may charge a fee for the physical act of transferring a copy, and
you may at your option offer warranty protection in exchange for a fee.

  2. You may modify your copy or copies of the Program or any portion
of it, thus forming a work based on the Program, and copy and
distribute such modifications or work under the terms of Section 1
above, provided that you also meet all of these conditions:

    a) You must cause the modified files to carry prominent notices
    stating that you changed the files and the date of any change.

    b) You must cause any work that you distribute or publish, that in
    whole or in part contains or is derived from the Program or any
    part thereof, to be licensed as a whole at no charge to all third
    parties under the terms of this License.

    c) If the modified program normally reads commands interactively
    when run, you must cause it, when started running for such
    interactive use in the most ordinary way, to print or display an
    announcement including an appropriate copyright notice and a
    notice that there is no warranty (or else, saying that you provide
    a warranty) and that users may redistribute the program under
    these conditions, and telling the user how to view a copy of this
    License.  (Exception: if the Program itself is interactive but
    does not normally print such an announcement, your work based on
    the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole.  If
identifiable sections of that work are not derived from the Program,
and can be reasonably considered independent and separate works in
themselves, then this License, and its terms, do not apply to those
sections when you distribute them as separate works.  But when you
distribute the same sections as part of a whole which is a work based
on the Program, the distribution of the whole must be on the terms of
this License, whose permissions for other licensees extend to the
entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest
your rights to work written entirely by you; rather, the intent is to
exercise the right to control the distribution of derivative or
collective works based on the Program.

In addition, mere aggregation of another work not based on the Program
with the Program (or with a work based on the Program) on a volume of
a storage or distribution medium does not bring the other work under
the scope of this License.
```

(continues on next page)

(continued from previous page)

```
  3. You may copy and distribute the Program (or a work based on it,
under Section 2) in object code or executable form under the terms of
Sections 1 and 2 above provided that you also do one of the following:

    a) Accompany it with the complete corresponding machine-readable
    source code, which must be distributed under the terms of Sections
    1 and 2 above on a medium customarily used for software interchange; or,

    b) Accompany it with a written offer, valid for at least three
    years, to give any third party, for a charge no more than your
    cost of physically performing source distribution, a complete
    machine-readable copy of the corresponding source code, to be
    distributed under the terms of Sections 1 and 2 above on a medium
    customarily used for software interchange; or,

    c) Accompany it with the information you received as to the offer
    to distribute corresponding source code.  (This alternative is
    allowed only for noncommercial distribution and only if you
    received the program in object code or executable form with such
    an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for
making modifications to it.  For an executable work, complete source
code means all the source code for all modules it contains, plus any
associated interface definition files, plus the scripts used to
control compilation and installation of the executable.  However, as a
special exception, the source code distributed need not include
anything that is normally distributed (in either source or binary
form) with the major components (compiler, kernel, and so on) of the
operating system on which the executable runs, unless that component
itself accompanies the executable.

If distribution of executable or object code is made by offering
access to copy from a designated place, then offering equivalent
access to copy the source code from the same place counts as
distribution of the source code, even though third parties are not
compelled to copy the source along with the object code.

  4. You may not copy, modify, sublicense, or distribute the Program
except as expressly provided under this License.  Any attempt
otherwise to copy, modify, sublicense or distribute the Program is
void, and will automatically terminate your rights under this License.
However, parties who have received copies, or rights, from you under
this License will not have their licenses terminated so long as such
parties remain in full compliance.

  5. You are not required to accept this License, since you have not
signed it.  However, nothing else grants you permission to modify or
distribute the Program or its derivative works.  These actions are
prohibited by law if you do not accept this License.  Therefore, by
modifying or distributing the Program (or any work based on the
Program), you indicate your acceptance of this License to do so, and
all its terms and conditions for copying, distributing or modifying
the Program or works based on it.

  6. Each time you redistribute the Program (or any work based on the
Program), the recipient automatically receives a license from the
```

(continues on next page)

(continued from previous page)

```
original licensor to copy, distribute or modify the Program subject to
these terms and conditions.  You may not impose any further
restrictions on the recipients' exercise of the rights granted herein.
You are not responsible for enforcing compliance by third parties to
this License.

  7. If, as a consequence of a court judgment or allegation of patent
infringement or for any other reason (not limited to patent issues),
conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License.  If you cannot
distribute so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you
may not distribute the Program at all.  For example, if a patent
license would not permit royalty-free redistribution of the Program by
all those who receive copies directly or indirectly through you, then
the only way you could satisfy both it and this License would be to
refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under
any particular circumstance, the balance of the section is intended to
apply and the section as a whole is intended to apply in other
circumstances.

It is not the purpose of this section to induce you to infringe any
patents or other property right claims or to contest validity of any
such claims; this section has the sole purpose of protecting the
integrity of the free software distribution system, which is
implemented by public license practices.  Many people have made
generous contributions to the wide range of software distributed
through that system in reliance on consistent application of that
system; it is up to the author/donor to decide if he or she is willing
to distribute software through any other system and a licensee cannot
impose that choice.

This section is intended to make thoroughly clear what is believed to
be a consequence of the rest of this License.

  8. If the distribution and/or use of the Program is restricted in
certain countries either by patents or by copyrighted interfaces, the
original copyright holder who places the Program under this License
may add an explicit geographical distribution limitation excluding
those countries, so that distribution is permitted only in or among
countries not thus excluded.  In such case, this License incorporates
the limitation as if written in the body of this License.

  9. The Free Software Foundation may publish revised and/or new versions
of the General Public License from time to time.  Such new versions will
be similar in spirit to the present version, but may differ in detail to
address new problems or concerns.

Each version is given a distinguishing version number.  If the Program
specifies a version number of this License which applies to it and "any
later version", you have the option of following the terms and conditions
either of that version or of any later version published by the Free
Software Foundation.  If the Program does not specify a version number of
this License, you may choose any version ever published by the Free Software
```

(continues on next page)

(continued from previous page)

```
Foundation.

  10. If you wish to incorporate parts of the Program into other free
programs whose distribution conditions are different, write to the author
to ask for permission.  For software which is copyrighted by the Free
Software Foundation, write to the Free Software Foundation; we sometimes
make exceptions for this.  Our decision will be guided by the two goals
of preserving the free status of all derivatives of our free software and
of promoting the sharing and reuse of software generally.

                            NO WARRANTY

  11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY
FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW.  EXCEPT WHEN
OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES
PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED
OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.  THE ENTIRE RISK AS
TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU.  SHOULD THE
PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING,
REPAIR OR CORRECTION.

  12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR
REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES,
INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING
OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED
TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY
YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER
PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE
POSSIBILITY OF SUCH DAMAGES.

                     END OF TERMS AND CONDITIONS

            How to Apply These Terms to Your New Programs

  If you develop a new program, and you want it to be of the greatest
possible use to the public, the best way to achieve this is to make it
free software which everyone can redistribute and change under these terms.

  To do so, attach the following notices to the program.  It is safest
to attach them to the start of each source file to most effectively
convey the exclusion of warranty; and each file should have at least
the "copyright" line and a pointer to where the full notice is found.

    <one line to give the program's name and a brief idea of what it does.>
    Copyright (C) <year>  <name of author>

    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.
```

(continues on next page)

(continued from previous page)

```
    You should have received a copy of the GNU General Public License along
    with this program; if not, write to the Free Software Foundation, Inc.,
    51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this
when it starts in an interactive mode:

    Gnomovision version 69, Copyright (C) year name of author
    Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License.  Of course, the commands you use may
be called something other than `show w' and `show c'; they could even be
mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the program, if
necessary.  Here is a sample; alter the names:

  Yoyodyne, Inc., hereby disclaims all copyright interest in the program
  `Gnomovision' (which makes passes at compilers) written by James Hacker.

  <signature of Ty Coon>, 1 April 1989
  Ty Coon, President of Vice

This General Public License does not permit incorporating your program into
proprietary programs.  If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library.  If this is what you want to do, use the GNU Lesser General
Public License instead of this License.
```

1.11.2 Other Licences

For other licences contact: apaulross@gmail.com

1.12 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

1.12.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/paulross/TotalDepth/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

TotalDepth could always use more documentation, whether as part of the official TotalDepth docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/paulross/TotalDepth/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

1.12.2 Get Started!

Ready to contribute? Here's how to set up *TotalDepth* for local development.

1. Fork the *TotalDepth* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:paulross/TotalDepth.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv TotalDepth
$ cd TotalDepth/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 TotalDepth tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.3 and above (we don't support Python 2.x), and for PyPy. Check https://travis-ci.org/paulross/TotalDepth/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ pytest tests.test_TotalDepth
```

1.12.3 Release Checklist

In the following example the version we are moving to, in `Major.Minor.Patch` format, is `0.2.1`.

Current version should be something like `M.m.(p)rcX`, for example `0.2.1rc0`.

Increment version

Change the version to `M.m.p` in these places:

- *setup.cfg*
- *setup.py*
- *src/TotalDepth/__init__.py*
- /docs/source/conf.py (two places).

In *src/TotalDepth/__init__.py* change `VERSION = (0, 2, 1)`

Update the history:

- *HISTORY.rst*
- *src/TotalDepth/__init__.py*

Update any Trove classifiers in *setup.py*, https://pypi.python.org/pypi?%3Aaction=list_classifiers

Build and Test

Build

Build, in this case for Python 3.6:

```
$ python3.6 -m venv ~/venvs/TotalDepth_00/
$ . ~/venvs/TotalDepth_00/bin/activate
(TotalDepth_00) $ pip install --upgrade pip
(TotalDepth_00) $ pip install --user --upgrade setuptools wheel
(TotalDepth_00) $ pip install -r requirements.txt
(TotalDepth_00) $ python setup.py develop
```

Test

As a minimal test:

```
(TotalDepth_00) $ pytest tests/
```

This should take under a minute.

A full test:

```
(TotalDepth_00) $ pytest --runslow tests/
```

This can take 10 to 30 minutes.

Build a Development Environment

This includes the full testing (including test coverage) and documentation environment.

As above plus:

```
(TotalDepth_00) $ pip install -r requirements-dev.txt
```

Test Coverage

With test coverage:

```
(TotalDepth_00) $ pytest --cov=TotalDepth --cov-report html tests --runslow
```

This can take 10 to 30 minutes.

Documentation

Build the docs HTML to test them, from an environment that has Sphinx:

```
(Sphinx) $ cd docs
(Sphinx) $ make clean
(Sphinx) $ make html
```

Commit and Tag

Commit, tag and push:

```
$ git add .
$ git commit -m 'Release version 0.2.1'
$ git tag -a v0.2.1 -m 'Version 0.2.1'
$ git push
$ git push origin v0.2.1
```

PyPi

Prepare release to PyPi for Python 3.6:

Build the egg and the source distribution:

```
(TotalDepth_00) $ python setup.py install sdist
```

Check the contents of `dist/*`, unpack into `tmp/` if you want:

```
$ cp dist/* tmp/
$ cd tmp/
$ unzip TotalDepth-0.2.1-py3.6-macosx-10.6-intel.egg -d py27egg
$ tar -xzf TotalDepth-0.2.1.tar.gz
```

Release to PyPi, https://pypi.python.org/pypi/TotalDepth:

```
(TotalDepth_00) $ twine upload dist/*
```

ReadTheDocs

Build the documentation: https://readthedocs.org/projects/TotalDepth/builds/

Prepare Next Release Candidate

Finally change the version to `M.m.(p+1)rc0`, in this example `0.2.2rc0` in these places:

- *setup.cfg*
- *setup.py*
- *src/TotalDepth/__init__.py*
- */docs/source/conf.py* (two places).

In *src/TotalDepth/__init__.py* change `VERSION = (0, 2, 2, 'rc0')`

Commit and push:

```
$ git add .
$ git commit -m 'Release candidate v0.2.2rc0'
$ git push
```

1.13 Glossary

Note: Terms here starting with "RP66V1." are taken directly from the RP66V1 glossary. They are lightly edited, corrected and enhanced before being reproduced here for the convenience of the reader.

Absent Value A recorded value from a sensor that is not a true recorded value and should be ignored. A point of difficulty arises for a conformant application; within the *LIS-79* format this is specified in the *DFSR* so can be per frame array. For *RP66V1* the absent value is supposed to be represented by having an attribute count of zero. In practice this never happens, instead a bunch of ad-hoc values need to be presumed such as -999, -999.25 based on their *RepCode*. See also *RP66V1.Absent Value*

Backup Mode A means of specifying what happens to plotted lines when they go off scale. Typical examples are None (all intermediate data is lost) and 'wrap' (all data is plotted with lines at modulo scale).

Data Format Specification Record [*LIS*] An EFLR that describes type 0 or type 1 binary IFLRs containing log data. A DFSR consists of a set of Entry Blocks followed by a list of Datum Specification Blocks.

NOTE: The LIS specification associates a particular DFSR type `0 | 1` with binary IFLRs of type `0 | 1`. These collections will be independent of each other and thus permits the simultaneous recording of entirely different data sets. In practice there is no evidence that any LIS implementations supports this.

Datum Specification Block [*LIS*] A fixed format data block that defines the characteristics of a single, independent, data channel in a DFSR.

DFSR See *Data Format Specification Record*

DLIS This adds schema specific semantics to *RP66V1* or *RP66V2*.

DSB See *Datum Specification Block*.

EFLR See *Explicitly Formatted Logical Record*.

Engineering Value A numeric value together with its units of measure.

Entry Block [*LIS*] A variable length block of data that describes a static value in DFSR. This value is local to a Log Pass. For example and Entry Block might describe the NULL or absent value for any channel in a Log Pass.

Explicitly Formatted Logical Record This is an internally complete, self-describing record. See also *RP66V1.Explicitly Formatted Logical Record*

FFLR See *Fixed Format Logical Record*.

Fixed Format Logical Record This is an internally complete record whose format is described by a standard. Does not occur in *RP66V1*.

Frame An array of values for each channel at a particular depth (or time).

Frame Set

Frame Array A set of frames representing multi-channel data that is typically depth or time series based.

IFLR

Indirectly Formatted Logical Record This is a Logical Record whose format is described by another EFLR. The EFLR that describes an IFLR might be identified formally; for example by a specific reference to an EFLR (as in RP66) or informally; by some heuristic (as in LIS) such as "the immediately prior Logical Record that is type 64 i.e. a *Data Format Specification Record*. See also *RP66V1.Indirectly Formatted Logical Record*

LAS Text based flog file format managed by the Canadian Well Logging Society

LIS

LIS-79 Log Information Standard. The original file format used by Schlumberger from 1979 onwards. It is a *proprietary*, *de-facto* standard.

LIS-A *Enhanced* Log Information Standard. This adds somewhat to the [*LIS-79*] standard.

Log Pass A TotalDepth term that describes a continuous body of logging data recorded simultaneously and independent of any other recording. Examples might be "Repeat Section" or "Main Log". A Log Pass contains one or more *Frame Array* (s). The number of allowable Frame Arrays within a Log Pass depends on the log format:

LAS can only support a single Frame Array within a Log Pass.

In the *LIS* format the Log Pass is defined by a single *DFSR* Logical Record. This can describe up to two Log Passes (type 0 or type 1) Logical Records. In practice only type 0 exists so LIS has just one Frame Array per Log Pass.

RP66V1 supports any number of Frame Arrays within a Log Pass and usually does.

Logical Record [*LIS*] A formal record from a LIS file. Logical Records consist of a header and optional payload. The Logical Records *type* is identified in the header. The interpretation of the payload of (some) Logical Records types is defined in the LIS standard. Logical Records consist of the payloads of one or more Physical Records. Logical Records are either EFLR or IFLR records.

LRH Logical Record header. The bytes that describe the type and attributes of a Logical Record.

Mnem See *Mnemonic*.

Mnemonic [*LIS*] A four byte identifier that is human readable e.g. `RHOB` for Bulk Density.

Physical Record [*LIS*] A formal record in a LIS file. Physical Records consist of a header, optional payload and optional trailer. Logical Records consist of the payloads of one or more Physical Records.

RepCode

Rep Code See *Representation Code*.

Representation Code A code, usually an integer, that describes how a particular run of bytes should be interpreted as a value (number, string etc.). For example in the LIS standard representation code 68 describes a 32 bit floating point format.

RP66 *Recommended Practice 66* is an API standard that describes a more advanced file format for, among other things, wireline logs. Comes in two flavours version 1 and version 2. Often (and incorrectly) referred to as *DLIS*.

RP66V1 *Recommended Practice 66* version 1 is an API standard first published on MAY 1, 1991 that describes a (mostly) more advanced file format compared with *LIS*. The specification can be found online from the Petrotechnical Open Software Corporation here RP66V1 standard See also the RP66V1 glossary

RP66V1.Absent Value The Value of an Attribute is an Absent Value and is undefined when the Attribute Count is zero. A Channel Sample Value is an Absent Value and is undefined when its Dimension Count is zero. Under DLIS, Absent Values are explicitly absent and are not represented by specially-designated numeric quantities.

> **Warning:** Unfortunately RP66V1 does not allow this to be set *per frame* (i.e. per *RP66V1.IFLR*) but only *per channel*. So data providers use a fixed and undeclared value such as -999 for integers and -999.25 for floats. This allows per-frame *and* per-channel absent values which means that RP66V1 files from all providers should be treated with some care in this area.

RP66V1.Attribute One of possibly many specific named items of information or data associated with an Object. An Attribute is similar in function to a column value of a row in a table or to a field of a record in a database relation. Its information content can be more general, however.

RP66V1.Channel A measured or computed quantity that occurs as a sequence of samples indexed against time, depth, or some other physical dimension of a well. Also a Set Type.

RP66V1.Characteristic A descriptive feature of a Set, an Object, or an Attribute. The Characteristics of a Set are its Name and Type, of an Object its Name, and of an Attribute its Label, Count, Representation Code, Units, and Value.

RP66V1.Company Code A numeric code assigned to a company that writes information under the DLIS format. Each company (or major division) is assigned a unique Company Code by the API.

RP66V1.Component The basic syntactic unit of an Explicitly Formatted Logical Record (EFLR). A Component consists of a Component Descriptor (one byte), followed by zero or more fields that contain Characteristics associated with a Set, Object, or Attribute.

RP66V1.Component Descriptor The first byte of a Component. It has a Role field (bits 1-3), which specifies whether the Component describes a Set, Object, or Attribute, and a Format field (bits 4-8), which indicate which Characteristics of the thing described are present in the remainder of the Component.

RP66V1.Component Format See *RP66V1.Component Descriptor*.

RP66V1.Component Role See *RP66V1.Component Descriptor*.

RP66V1.Compound Representation Code A Representation Code that is defined in terms of other simpler Representation Codes.

RP66V1.Consumer The system or application program or company that reads information recorded under the DLIS Logical format. The Consumer reads what the Producer writes.

RP66V1.Copy Number A number, having meaning only within the context of a Logical File, that is used to distinguish two Objects of the same Type that have the same Identifier and Origin. The Name of an Object consists

RP66V1.Count One of the five Characteristics of an Attribute. The Count specifies how many Elements are in the Value of the Attribute. When the Count is zero, the Attribute has an Absent Value.

RP66V1.Data Descriptor Reference The first field of an Indirectly Formatted Logical Record (IFLR). The Data Descriptor Reference is the Name of an Object that identifies and describes a sequence of IFLRs. Each IFLR Type is associated with a specific Set Type to which such Objects belong.

RP66V1.Defining Origin The first Origin Object in a Logical File. The Defining Origin describes the environment under which the Logical File was created.

RP66V1.Descriptor See *RP66V1.Component Descriptor*.

RP66V1.Dictionary A Dictionary is a database in which Identifiers used under DLIS are administered. The standard does not specify the mechanisms for designing, creating, or managing Dictionaries. However, it does specify for which Set Types Identifiers should be managed. The statement, "Names of Set Type X are dictionary-controlled" means that Identifiers for such Objects have a persistent meaning in all Logical Files in which they occur (by a given Producer). Identifiers of Objects for Set Types that are not dictionary-controlled are considered void of meaning and are expected to be computer- generated.

RP66V1.Dimension This is a vector of integers which describe the form and size of a rectangular array that is represented elsewhere, for example as a Channel Sample. The first integer specifies the number of remaining integers and the dimensionality of the array (i.e., 1-d, 2-d, etc.). The remaining integers specify the number of elements along each dimension (or coordinate) of the array. The Dimension of an array is typically contained in the Dimension Attribute of some Object that is associated with the array.

RP66V1.EFLR See *RP66V1.Explicitly Formatted Logical Record*

RP66V1.Element One of a vector of homogeneous quantities that make up the Value of an Attribute or of a Channel Sample. A Value or Sample may consist of one or more Elements. All Elements have the same Representation Code and Units. The number of Elements of an Attribute Value is specified by the Attribute Count. The number of Elements of a Channel Sample is specified by its associated Dimension.

RP66V1.Encryption Packet An optional sequence of bytes that follows the Logical Record Segment Header and precedes the Logical Record Segment Body and that contains information used to decrypt the Logical Record. The first two bytes of the Encryption Packet specify its length, and the next two bytes specify the Producer's Company Code. The remaining bytes are meaningful only to the Producer.

RP66V1.Explicitly Formatted Logical Record One of two kinds of Logical Record defined under DLIS. The Body of an EFLR is a sequence of Components that combine to describe a single Set of Objects. An EFLR is self-describing and can be interpreted without the use or knowledge of any other Logical Records. More simply put, an Explicitly Formatted Logical Record is a table of rows and columns. Each row/column contains a *RP66V1.Attribute*.

RP66V1.Format Version A two-byte field immediately following the Visible Record Length in Visible Records on Record Storage Units (e.g., standard 9-track tapes). This field is used to distinguish DLIS from other formats

and to distinguish DLIS Version 1 from later major versions. The first byte of the Format Version contains the value FF (hex), and the second byte is the major version number of the standard (in the current case, 1).

RP66V1.Frame The Indirectly Formatted Data of an IFLR of Type FDATA (see Appendix A) is called a Frame. A Frame is made of a Frame Number, followed by a set of Channel sample values, one sample per Channel, all sampled at the same index value. One of the Channels may serve as an index. When this is the case, it is always the first Channel in the Frame. When there is no Channel index, then the Frame Number serves as an index.

RP66V1.Frame Data Information recorded in Frames is called Frame Data. This consists of Channel samples, one sample per Channel per Frame.

RP66V1.Frame Number A positive integer recorded at the beginning of each Frame. The Frame Number is a sequential index of the Frames of the same Frame Type. Frame n precedes Frame n+1, although other Logical Records may fall between.

RP66V1.Frame Type The Name (Origin, Copy Number, Identifier) of a Frame Object used to group Frames that have the same organization. This Name is also used as the Data Descriptor Reference in the Frames, and the Frames are known to be of the given Frame Type. Frames of a given Frame Type all contain samples of the same set of Channels and all in the same order. The Representation Code and Units used to record a Channel are the same in all Frames of a given Frame Type but may be different in another Frame Type. A Channel sample may change size (number of Elements) from Frame to Frame and may even become Absent when its number of Elements reduces to zero.

RP66V1.Header (Refer to Logical Record Segment Header) See *RP66V1.Logical Record Segment Header*.

RP66V1.Identifier That part of an Object Name that is textual. The Identifier is what commonly distinguishes one Object from another. Two Objects of the same Type may have the same Identifier, in which case the other Subfields of the Name are used to distinguish the Objects. Identifiers of certain Types of Objects uniquely identify the type of data represented in the Object, and such Identifiers (typically mnemonic in nature) are dictionary-controlled.

RP66V1.IFLR See *RP66V1.Indirectly Formatted Logical Record*

RP66V1.Index Channel The first Channel in a Frame, when the Frame has an Index Channel. A Frame may be indexed by Frame Number only and need not have an Index Channel. Whether or not a Frame has an Index Channel is specified in the associated Frame Object. When a Frame has an Index Channel, then all Channel values in the Frame are considered to be sampled at the index indicated by the value of the Index Channel.

RP66V1.Indirectly Formatted Data That part of the Body of an Indirectly Formatted Logical Record (IFLR) that follows the Data Descriptor Reference.

RP66V1.Indirectly Formatted Logical Record One of two kinds of Logical Record defined under DLIS. The Body of an IFLR consists of a Data Descriptor Reference, followed by an arbitrary number of bytes of Indirectly Formatted Data. This data is not self-descriptive. Instead, its format is determined from information contained in the Object named by the Data Descriptor Reference and possibly related Objects. For example, the format of a Frame Data IFLR is specified by a Frame Object and by one or more Channel Objects referenced by the Frame Object.

RP66V1.Invisible Envelope Data recorded on the physical medium and used as a control interface by the processor I/O subsystem, but not visible through normal application read/write requests, for example tape marks on magnetic tape.

RP66V1.Lexicon A list of dictionary-controlled words or phrases applicable as Name Part Values for a particular Name Part Type. For example, each Producer manages a Lexicon of Entity names and another Lexicon of Quantity names.

RP66V1.Locus A sequence of distinct points in space and time, each of which has a three-dimensional Position coordinate and a Time coordinate.

RP66V1.Logical File A sequence of two or more contiguous Logical Records in a Storage Set that begins with a File Header Logical Record and contains no other File Header Logical Records. A Logical File must have at

least one OLR (Origin) Logical Record immediately following the File Header Logical Record. A Logical File supports user-level organization of data.

RP66V1.Logical Format The view of DLIS data that is completely independent of any physical mapping. The DLIS Logical Format consists of a sequence of Logical Records organized into one or more Logical Files. This format is the same for any physical representation of the data.

RP66V1.Logical Record A sequence of one or more contiguous Logical Record Segments. A Logical Record supports application-level organization of data.

RP66V1.Logical Record Body The sequential concatenation of the Logical Record Segment Bodies from the Logical Record Segments that make up the Logical Record.

RP66V1.Logical Record Segment A sequence of contiguous 8-bit bytes organized to have a Logical Record Segment Header, followed (optionally) by an Encryption Packet, followed by a Logical Record Segment Body, followed (optionally) by a Logical Record Segment Trailer. Logical Record Segments are used to bind the Logical Format to a physical format.

RP66V1.Logical Record Segment Attributes Eight bits of binary data that describe the attributes of a Logical Record Segment.

RP66V1.Logical Record Segment Body The part of a Logical Record Segment that contains some or all of the data belonging to a Logical Record. The intersection of a Logical Record and one of its Logical Record Segments is the Logical Record Segment Body.

RP66V1.Logical Record Segment Encryption Packet An optional packet of information, following the Logical Record Segment Header, that contains encryption/decryption information for the Logical Record Segment. The Encryption Packet begins with its size in bytes and the Company Code of the Producer. Any additional data in the Encryption Packet is meaningful only to the Producer's organization.

RP66V1.Logical Record Segment Header The first part of a Logical Record Segment. It contains the Logical Record Segment Length, the Logical Record Segment Attributes, and the Logical Record Type.

RP66V1.Logical Record Segment Length A two-byte unsigned integer that specifies the combined length of all parts of the Logical Record Segment.

RP66V1.Logical Record Segment Trailer The last part of a Logical Record Segment. It contains three fields, all of which are optional: the Padding, the Checksum, and the Logical Record Segment Trailing Length.

RP66V1.Logical Record Structure One of the attributes specified in the Logical Record Segment Attributes. It specifies whether the Logical Record is an EFLR or an IFLR.

RP66V1.Logical Record Type A one-byte unsigned integer that indicates the general semantic content of the Logical Record.

RP66V1.Long Name A structured textual description that provides an understanding, to humans, of the named item, with enough detail to distinguish it from similar items that have different meanings. It is not a unique identifier. A Long Name is represented in a Long-Name Object.

RP66V1.Maximum Visible Record Length The maximum permitted length of a Visible on a Record Storage Unit. Its current value is 16,384 bytes.

RP66V1.Minimum Visible Record Length The minimum permitted length of a Visible Record on a Record Storage Unit. Its current value is 20 bytes, which is based on the minimum Logical Record Segment Length (16 bytes) plus the Visible Record Length (2 bytes) and the Format Version (2 bytes).

RP66V1.Name Used to refer to the Name Characteristic of a Set, Object, or Attribute. The Name of a Set or Attribute is a character string (Representation Code IDENT). The Name of an Object is an aggregate consisting of an integer Origin, an integer Copy Number, and a character Identifier.

RP66V1.Name Part Type A classification of the words or phrases that apply to a particular part of the Long Name structure, for example Entity or Quantity.

RP66V1.Name Part Value A word or phrase that applies to a particular part (Name Part Type) of a Long Name structure. For example, "Density" and "Porosity" are Name Part Values that apply to the Name Part Type "Quantity".

RP66V1.Object A data entity that has a Name and a number of Attributes. An Object is like a row in a table of information. Its Attributes are like the column values in the row. Objects are recorded in EFLRs.

RP66V1.Object Component An Object Component indicates the beginning of a new Object in a Set and is followed by zero or more Attribute Components. The Attributes of an Object that has no Attribute Components are completely specified in the Template. The Object Component contains a single Characteristic, the Object Name, which is mandatory.

RP66V1.Origin As an Object in a Logical File, an Origin contains information describing the original circumstances under which that or another Logical File was created. Only one Origin Object in a Logical File, namely the first one, describes that Logical File. Additional Origin Objects describe other Logical Files from which data has been copied. Other Objects in a Logical File are keyed to their appropriate Origin Object by means of an integer Subfield in their Names, namely the Origin Subfield. This integer value matches the Origin Subfield of the appropriate Origin Object. This integer value is also commonly referred to as the Object's Origin.

RP66V1.Pad Bytes Pad Bytes are part of the Logical Record Segment Trailer and are used to alter the size of a Logical Record Segment to satisfy minimum size requirements or more commonly to make the Logical Record Segment Length divisible by some integer. In all cases, the Logical Record Segment Length must be divisible by two. Additionally, certain encryption methods may require the length of the Logical Record Segment Body plus the Pad Bytes to be divisible by some other factor.

RP66V1.Pad Count This is the first byte of the Pad Bytes and indicates how many Pad Bytes there are. The maximum number of Pad Bytes may therefore not exceed 255.

RP66V1.Padding An informal reference to Pad Bytes.

RP66V1.Parent File The Logical File in which data are originally created. Some data in a Logical File may have been copied from other Logical Files.

RP66V1.Path A sequence of space-time coordinates, where space is typically represented by depth, radial distance from a vertical line, and angular displacement about the same vertical line. The vertical line used is the one that goes through a well's Well Reference Point, a point used to identify the location of a well. A Path may be represented by a combination of Channels, each of which represents one of the above-mentioned coordinates.

RP66V1.Physical Format The way in which recorded data is located and organized on a particular physical medium such as a magnetic tape or disk file. With some I/O systems more than one organization and view of data is supported on the same medium. Each such view corresponds to a different physical format. For example, disk files may be viewed as having variable-length record structures, block structures, byte stream structures, etc., depending on the I/O facility that is used. The physical format determines the way in which Logical Record Segments are used but generally has no effect on Logical Records.

RP66V1.Predecessor Used to indicate the relation between successive Logical Record Segments. If two Logical Record Segments belong to the same Logical Record, then one of them — the one that comes first — is a Predecessor of the other. The first Logical Record Segment of a Logical Record has no Predecessor.

RP66V1.Private A Logical Record with Type 128 is said to be Private. In particular, the semantic content of such a Logical Record is decided upon solely by the Producer and not via any public standardization process. Private Logical Records are available to consumers in general, unless encrypted. The fact that a Logical Record is Private does not imply that it is also encrypted.

RP66V1.Producer The system or application program or company that records information under the DLIS Logical Format. The Producer writes what the Consumer reads.

RP66V1.Public A Logical Record with Type < 128 is said to be Public. In particular, the semantic content of such a Logical Record is agreed to by the all users via a standardization process administered by the API. Such Logical

Records may not be used except in accordance with the standard definition. Public Logical Records may be encrypted or not, according to the needs of the Producer.

RP66V1.Radial Drift Radial Drift is the perpendicular distance of a point from a vertical line that passes through the Well Reference Point of a well.

RP66V1.Record Structure One of possibly many different physical formats. A Storage Unit is said to have Record Stucture if data is written and read in sequential, variable-length records. For all Record Structure Storage Units each record must begin with a two-byte unsigned integer Visible Record Length, followed by a two-byte Format Version, followed by an integer number of Logical Record Segments. Other requirements, for example use of Tape Marks, depend on the particular physical medium.

RP66V1.Representation Code Each distinct piece of information in the Logical Format has a well-defined representation that extends across one or more bytes. Each different representation is identified by a one-byte Representation Code. Representation Codes are defined in Appendix B and identify the various floating point, integer, and text representations permitted under the DLIS.

RP66V1.Sample (of a Channel) A Channel Sample is one of a sequence of evaluations of a Channel. A Channel Sample may be a scalar sensor reading (i.e., a single number), or it may be an array representing a waveform or some other multi-dimensional data.

RP66V1.Semantics Semantics is the definition of what data means and how it is used. Whereas syntax provides rules for recording Objects in Sets, semantics defines the Objects that may be recorded, e.g., the File-Header, Origin, Channel, Frame, etc. Objects.

RP66V1.Set A data entity that has a Type and optionally a Name, and contains a number of Objects. A Set is like a table of information in which the Objects are the rows of the table. Each Set is contained in an EFLR (exactly one Set per EFLR), and there may be more than one Set with the same Type in a Logical File.

RP66V1.Set Component A Set Component indicates the beginning of a Set and is followed by one or more Template Attribute Components. A Set Component always contains a Type Characteristic and may contain a Name Characteristic.

RP66V1.Set Type A textual identifier of the type of Objects contained in a Set. The Objects in a Set are characterized by the Attributes in the Template of the Set. The Attributes associated with each given Set Type are specified in the standard.

RP66V1.Slot A data entity that has a Type and optionally a Name, and contains a number of Objects. A Set is like a table of information in which the Objects are the rows of the table. Each Set is contained in an EFLR (exactly one Set per EFLR), and there may be more than one Set with the same Type in a Logical File. One of a fixed number of positions in a Frame for recording a single Channel Sample value. Channels are assigned to Slots in a Frame in a specific order, and all Slots follow the Frame Number. The Index Channel, if there is one, is in the first Slot of a Frame.

RP66V1.Splice A Splice is the result of concatenating two or more instances of a Channel (e.g., from different runs) to obtain a resultant Channel defined over a larger domain or interval. The information associated with a Splice is represented in a Splice Object.

RP66V1.Static Information Static Information consists of Objects typically used to describe Channels and Frames, and information about Channels and Frames. Static Information is typically required by an application prior to the processing of Frames.

RP66V1.Storage Set A group of Storage Units that contain a common DLIS Logical Format (e.g., a sequence of Logical Files) and for which at least two Storage Units are spanned by a single Logical File.

RP66V1.Storage Set Identifier A 60-character ASCII field in the Storage Unit Label used to identify a Storage Set.

RP66V1.Storage Unit Something that contains DLIS data and is manageable as a unit at the human level, (e.g., a tape or disk file).

RP66V1.Storage Unit Label The first 80 bytes of the Visible Envelope of a Storage Unit. The Storage Unit Label consists of five fixed-length ASCII fields used to identify the Storage Unit and the Storage Set of which it is a part.

RP66V1.Storage Unit Sequence Number A positive integer (its ASCII representation) in the Storage Unit Label that indicates the order in which a Storage Unit occurs in a Storage Set.

RP66V1.Storage Unit Structure An ASCII keyword in the Storage Unit Label that reflects the Physical Format of the Storage Unit and indicates the binding mechanism between the Physical Format and the DLIS Logical Format.

RP66V1.Subfield A part of a datum for which the representation is described by a simple (not compound) Representation Code. For example, the Subfields of a datum having Representation Code OBNAME are, in order, an integer (UVARI), another integer (USHORT), and a string (IDENT).

RP66V1.Successor Used to indicate the relation between successive Logical Record Segments. If two Logical Record Segments belong to the same Logical Record, then one of them — the one that comes second — is a Successor of the other. The last Logical Record Segment of a Logical Record has no Successor.

RP66V1.Syntax Syntax is the definition of the rules for how to record data but not for what the data means (at the application level) or how it is to be used. Syntax does convey meaning of data, but at a level below applications. For example, the rules of syntax tell when a Component has a Type Characteristic and how to get it, but syntax provides no information on the meaning or use of the values the Type Characteristic may have.

RP66V1.Template A sequence of Attributes at the beginning of a Set that specify defaults for the Objects in the Set. Attributes in the Template must have Labels. Objects in the Set have no Attributes other than those identified in the Template.

RP66V1.Tool Zero Point A fixed point on the tool string (usually the bottom of the bottom tool) that stands opposite the Well Reference Point when Borehole Depth is zero.

RP66V1.Trailing Length The optional last field in the Logical Record Segment Trailer that contains a copy of the Logical Record Segment Length.

RP66V1.Transient Information Transient Information consists of Objects that correspond to events that occur during the processing of Frames. These events can affect Objects in the Static Information or can correspond to messages between the operator and the system.

RP66V1.Unzoned A Parameter or Computation Object is said to be Unzoned when it has the same value everywhere. This is the case when the Zones Attribute of the Object is absent.

RP66V1.Update An Update is a change made to data represented by an Object (e.g., a Parameter) previously recorded in a Logical File. The change and information related to the change are represented in an Update Object recorded in the same Logical File.

RP66V1.Value (of an Attribute) The Value of an Attribute is the data contained in its Value Characteristic. A Value may consist of one or more Elements, each of which has the same Units and Representation Code.

RP66V1.Vertical Depth Depth measured along the Vertical Generatrix from the Well Reference Point. Vertical depth increases in a downward direction and is negative above the Well Reference Point.

RP66V1.Vertical Generatrix A vertical line that passes through the Well Reference Point.

RP66V1.Visible Envelope Information on a Storage Unit that is provided to applications as normal data by the processor's I/O subsystem, but which is not part of the DLIS Logical Format. Information in the Visible Envelope includes the Storage Unit Label. Other information in the Visible Envelope may be used to define or enhance the binding of Logical Record Segments to the Physical Format.

RP66V1.Visible Record A Visible Record is a term that applies to Record Structure Storage Units. It consists of all the data bytes accessed by means of a record read operation from the system-specific file access subsystem.

RP66V1.Visible Record Length When DLIS information is recorded in variable length pysical records, each Visible Record begins with a two-byte unsigned integer length of the Visible Record called the Visible Record Length. This length is considered to be external the DLIS Logical Format.

RP66V1.Well Reference Point A unique point that is the origin of a well's spatial coordinate system for information in a Logical File. This point is defined relative to some permanent vertical structure, such as ground level or mean sea level, and to three independent geographical coordinates, which typically include Latitude and Longitude. The same well may have different Well Reference Points in different Logical Files.

RP66V1.Zone A Zone is a single interval in depth or time. The depth coordinate may be either Vertical Depth or Borehole Depth.

RP66V1.Zoned A Parameter or Computation Object is said to be Zoned when it has different values in different intervals along a depth or time domain or is undefined in some interval of a depth or time domain. This is the case when the Zones Attribute of the Object is not absent.

RP66V2 *Recommended Practice 66* version 2 is an API standard first published in June 1996 that describes a (mostly) more advanced file format compared with *RP66V1*. It is unused by the industry and will not be referenced here. The specification can be found online from the Petrotechnical Open Software Corporation here RP66V2 standard See also the RP66V2 glossary

UOM *Unit of Measure* for engineering values. In *LIS* these are a set of fixed terms organised into several categories, such as *Linear Length*. Values can only be converted between units of in the same category. In *RP66V1* these are composed by a BNF parseable string.

Xaxis

X Axis The index channel in an array, for example an array of frames. Typically depth or time.

1.14 History

1.14.1 0.4.0 (2021-09-11)

- **General**
  - Add SLB parameter and unit online lookup.
  - Add detection of CFBF, EBCDIC, RCD, SEG-Y, STK, PDS binary file types.
  - Python 3.6 no longer supported, although it will most likely work.
- Specific File formats
  - BIT
    - Support Western Atlas BIT files.
    - Add BIT file conversion to LAS.
    - BIT float to bytes (ISINGL) encoding.
  - DAT
    - Add DAT file support using the common FrameArray.
  - LAS
    - Add tdlastohtml as an entry point.
    - Parser improvements.
    - LAS reader now ignores duplicate channels if requested.

    * Add LAS variants to binary_file_type.
    * LAS FrameArray writing now in TotalDepth.common.LogPass
  - LIS
    * Add LIS to LAS conversion.
    * Kill off XNAM LIS support.
    * Better handling of LIS Physical Record padding.
    * Fix for LIS indexer when the DFSR is missing.
    * Adds generation of AREA patterns in SVG.
    * Creates PNG pattern files from XML data. Provides an API to pattern files and Data URI Scheme inline implementations.
  - RP66V1
    * Prepare for RP66V1 C/C++ code. Update to Python 3.8, 3.9, 3.10.
    * Add units conversion.

1.14.2 0.3.1 (2020-06-15)

- Fixes for builds on Linux and Windows.

1.14.3 0.3.0 (2020-01-01)

- Adds full RP66V1 support.
- Tested against multi GB data set.

1.14.4 0.2.1 (2018-04-21)

- Minor fixes.

1.14.5 0.2.0 (2017-09-25)

- Moved to Github: https://github.com/paulross/TotalDepth
- First release on PyPI.

1.14.6 0.1.0 (2012-03-03)

- First release on Sourceforge: https://sourceforge.net/projects/TotalDepth/ registered: 2011-10-02

Earlier versions (unreleased):

- OpenLis - 2010-11-11 to 2011-08-01
- PyLis - 2009 to 2010

For many years this project was hosted by Sourceforge. Thank you Sourceforge!

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

t