TotalDepth Documentation

Release 0.2.0

Paul Ross

Sep 26, 2017

CONTENTS

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

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 **Alpha** version 0.2.0, release 0.2.0. For the licence see *TotalDepth 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.

1.1.1 Software Technology

For rapid development and performance TotalDepth is written in Python with performance critical code written in C/C++ or Cython.

TotalDepth was developed using TDD (Test Driven Development) in an agile fashion using *nano-iterations*.

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.

[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.

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`
- 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: *tdplotlogs*

1.2.2 LIS Log HTML Summaries

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

How This HTML was Made

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

[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.

```
$ 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: *tdlistohtml*

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

First make a virtual environment in your *<PYTHONVENVS>*, say *~/pyvenvs*:

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

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

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

If you want to build the documentation you need to:

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

1.3.4 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.

1.3.5 Unit Testing

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

1.4 Petrophysical File Formats that TotalDepth can Process

TotalDepth is currently at **Alpha**, release version 0.1.0.

Format	Support	Since	Notes.
LIS79	Full	0.1.0	
LIS-A	Partial	0.1.0	No direct XNAM support.
LAS version 1.2	Full	0.1.0	
LAS version 2.0	Full	0.1.0	
LAS version 3.0	None		
WellLogML	None		Absence of good test cases. The format seems to have made little impact on the industry.
RP66v1 "DLIS"	Proto.		Absence of good test cases.
RP66v2 "DLIS"	Proto.		Absence of good test cases.
SEG-Y etc.	None		Use cases?
Positioning	None		Fairly simple, use cases?

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/`

Plotting Well Logs

These command line tools plot wireline data.

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
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
    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' \
        )
)

... 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:

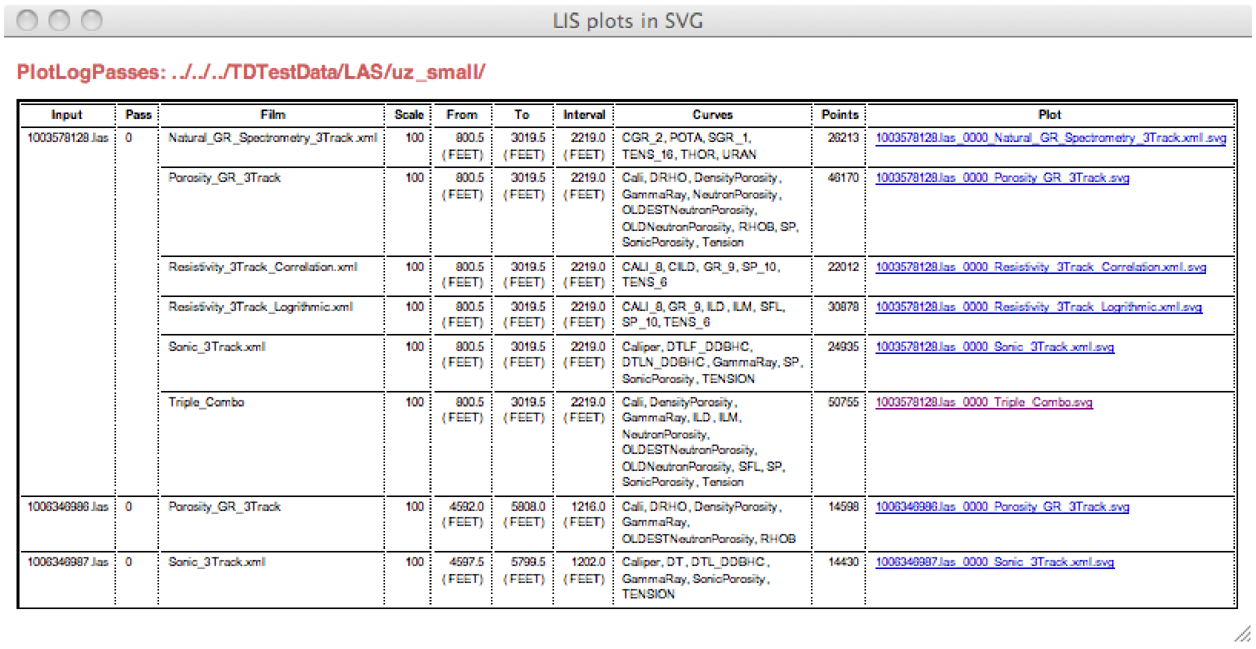
- Input file name.
- LogPass number in the file. For example "Repeat Section" might be 0 and "Main Log" 1.
- LgFormat used for the plot (several plots my be generated from one LogPass).
- **An IndexTableValue object (used to generate the index.html file) that has the following fields:**
  - Plot scale 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.

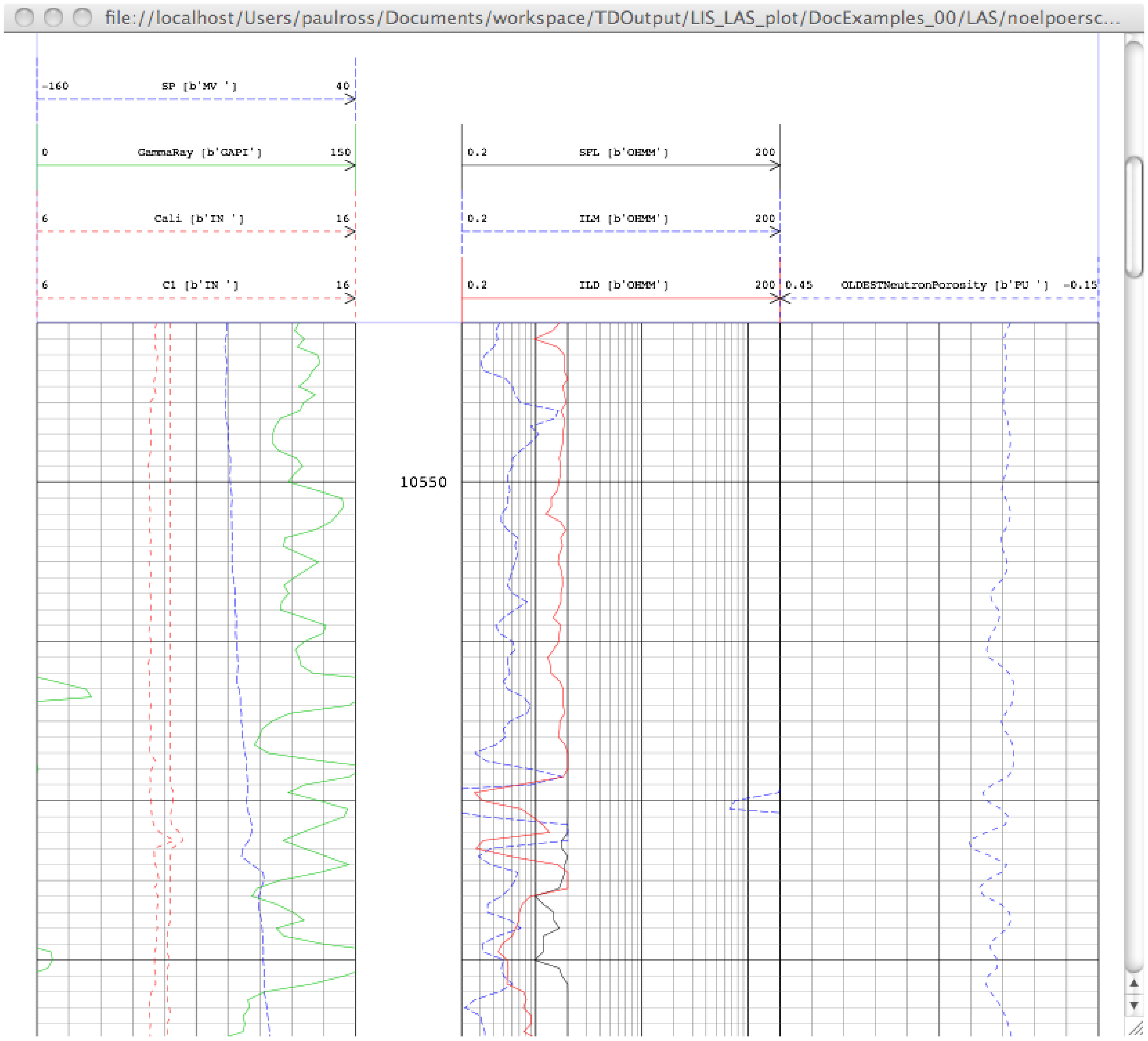
In the output directory will be an index.html file that has a table with the fields that duplicate those on the command line output. It looks like this:



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
1006346986.las	0	Porosity_GR_3Track	100	4592.0 (FEET)	5808.0 (FEET)	1216.0 (FEET)	Cali, DRHO, DensityPorosity, GammaRay, OLDESTNeutronPorosity, RHOB	14598	1006346986.las_0000_Porosity_GR_3Track.svg
1006346987.las	0	Sonic_3Track.xml	100	4597.5 (FEET)	5799.5 (FEET)	1202.0 (FEET)	Caliper, DT, DTL_DDBHC, GammaRay, SonicPorosity, TENSION	14430	1006346987.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 LIS Command Line Tools

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

Tool Name	Description
`tdlisdetif`	Removes TIF markers from a LIS file.
`tdlisdumpframeset`	Writes out the frame values as a CSV file.
`tdlisindex`	Indexes a LIS file.
`tdlisplotlogpasses`	Plots the log data as SVG pages.
`tdlisscanlogidata`	Scans the logical data.
`tdlisscanlogirecord`	Scans all Logical records.
`tdlisscanphysrec`	Scans all the Physical Records.
`tdlistablehistogram`	Analyses the contents of table Logical Records.
`tdlistohtml`	Generates a HTML page about the LIS file.
`tdplotlogs`	Plots logs as SVG files.

Scanning LIS Files

These command line tools scan LIS files without changing them.

`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.

`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.
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!
```

`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\
↪'...
0x00003168       1010     0  b
↪'\x00\x00E\xfe\xb6\x16\xbe\x12\xde\xf0@\xcbl\xe7@zF\xc2Ew\xba/
↪D\xd0\xca\xd9E\xfe\xb3\xa6\xbe\x17'...
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\xf
↪'...
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\
↪'...
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'
...
```

Extracting Data from LIS

`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
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.

`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
```

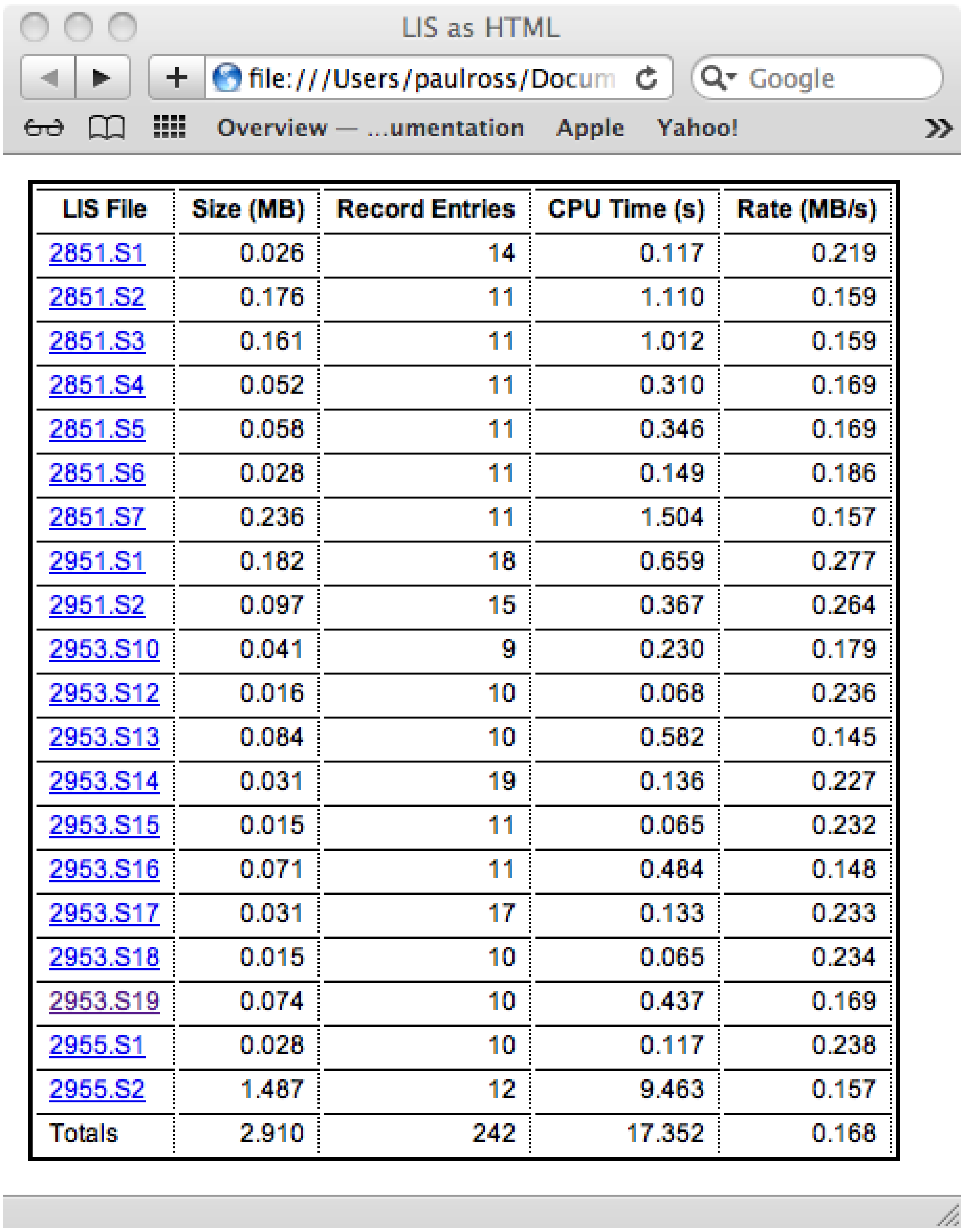
Output:

```
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

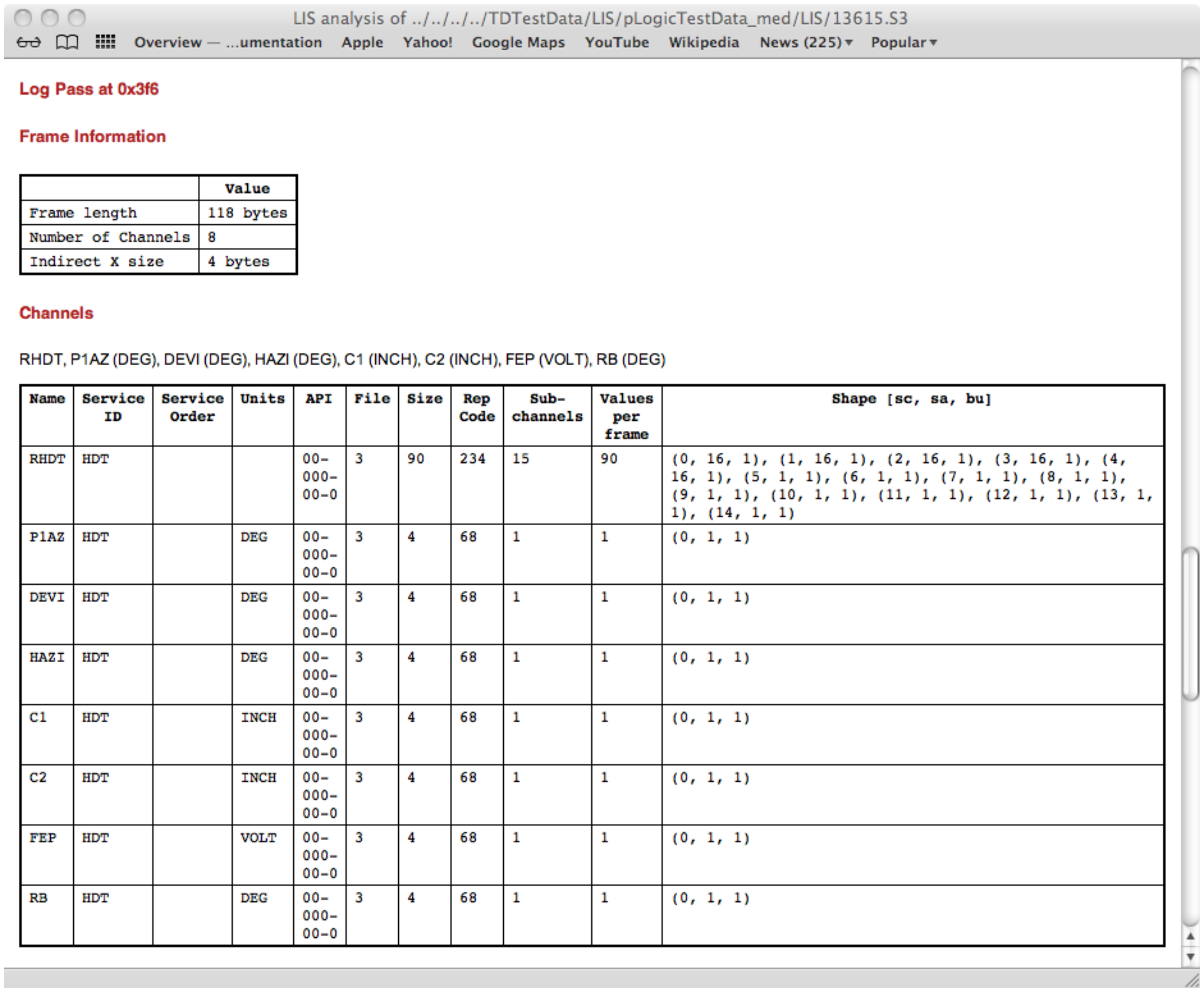
The columns are:

- 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:



Log Pass at 0x3f6

Frame Information

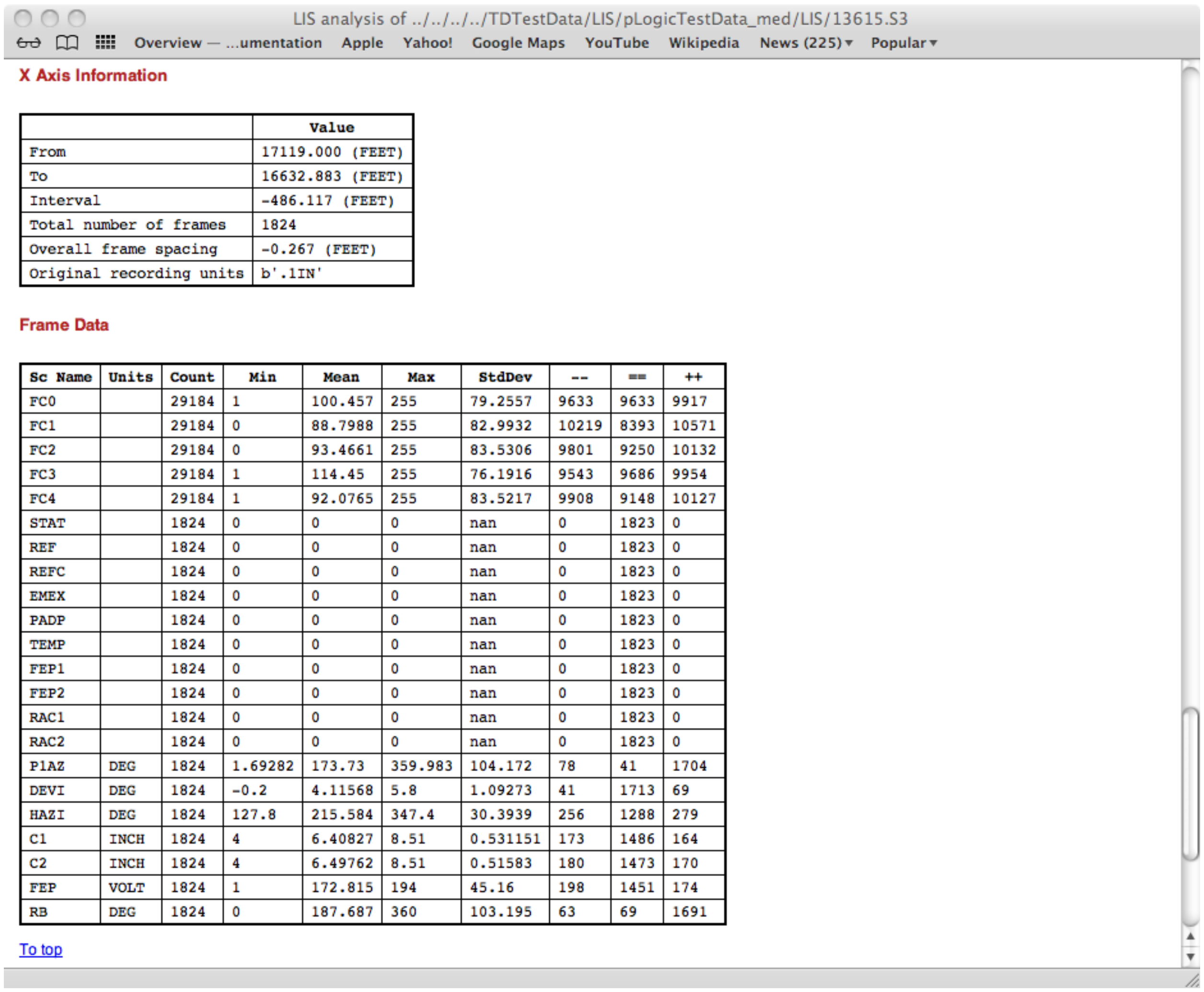
	Value
Frame length	118 bytes
Number of Channels	8
Indirect X size	4 bytes

Channels

RHDT, P1AZ (DEG), DEVI (DEG), HAZI (DEG), C1 (INCH), C2 (INCH), FEP (VOLT), RB (DEG)

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:



LIS analysis of ../../../../TDTestData/LIS/pLogicTestData_med/LIS/13615.S3

X Axis Information

	Value
From	17119.000 (FEET)
To	16632.883 (FEET)
Interval	-486.117 (FEET)
Total number of frames	1824
Overall frame spacing	-0.267 (FEET)
Original recording units	b'.1IN'

Frame Data

Sc Name	Units	Count	Min	Mean	Max	StdDev	--	==	++
FC0		29184	1	100.457	255	79.2557	9633	9633	9917
FC1		29184	0	88.7988	255	82.9932	10219	8393	10571
FC2		29184	0	93.4661	255	83.5306	9801	9250	10132
FC3		29184	1	114.45	255	76.1916	9543	9686	9954
FC4		29184	1	92.0765	255	83.5217	9908	9148	10127
STAT		1824	0	0	0	nan	0	1823	0
REF		1824	0	0	0	nan	0	1823	0
REFC		1824	0	0	0	nan	0	1823	0
EMEX		1824	0	0	0	nan	0	1823	0
PADP		1824	0	0	0	nan	0	1823	0
TEMP		1824	0	0	0	nan	0	1823	0
FEP1		1824	0	0	0	nan	0	1823	0
FEP2		1824	0	0	0	nan	0	1823	0
RAC1		1824	0	0	0	nan	0	1823	0
RAC2		1824	0	0	0	nan	0	1823	0
P1AZ	DEG	1824	1.69282	173.73	359.983	104.172	78	41	1704
DEVI	DEG	1824	-0.2	4.11568	5.8	1.09273	41	1713	69
HAZI	DEG	1824	127.8	215.584	347.4	30.3939	256	1288	279
C1	INCH	1824	4	6.40827	8.51	0.531151	173	1486	164
C2	INCH	1824	4	6.49762	8.51	0.51583	180	1473	170
FEP	VOLT	1824	1	172.815	194	45.16	198	1451	174
RB	DEG	1824	0	187.687	360	103.195	63	69	1691

To top

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.

LIS Developer Tools

tdlisdetif

Takes an input LIS79 file and writes out a new one without TIF markers.

Arguments

1. The path to the input LIS file.
2. The path to the output LIS file, 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.
`-n, --nervous`	Nervous mode (do no harm). [default: False]
`-l LOGLEVEL, --loglevel=LOGLEVEL`	Log Level (debug=10, info=20, warning=30, error=40, critical=50) [default: 20]

Examples

DeTIF with nervous mode just examines the file:

```
$ ``tdlisdetif`` -n RW.lis RW_No_TIF.lis
Cmd: DeTif.py -n RW.lis RW_No_TIF.lis
stripTif(): Tell: 0x00000000 Len: 0x00000000 TIF: TIF  True >:  0x        0  0x          ␣
↪0  0x        0
stripTif(): Tell: 0x0000000c Len: 0x0000003e TIF: TIF  True >:  0x        0  0x          ␣
↪0  0x       4a
stripTif(): Tell: 0x00000056 Len: 0x00000400 TIF: TIF  True >:  0x        0  0x          ␣
↪0  0x      456
stripTif(): Tell: 0x00000462 Len: 0x00000400 TIF: TIF  True >:  0x        0  0x         ␣
↪4a  0x       862
stripTif(): Tell: 0x0000086e Len: 0x00000400 TIF: TIF  True >:  0x        0  0x       ␣
↪456  0x      c6e
stripTif(): Tell: 0x00000c7a Len: 0x00000400 TIF: TIF  True >:  0x        0  0x       ␣
↪862  0x     107a
stripTif(): Tell: 0x00001086 Len: 0x00000400 TIF: TIF  True >:  0x        0  0x       ␣
↪c6e  0x     1486
stripTif(): Tell: 0x00001492 Len: 0x00000400 TIF: TIF  True >:  0x        0  0x      ␣
↪107a  0x     1892
stripTif(): Tell: 0x0000189e Len: 0x00000400 TIF: TIF  True >:  0x        0  0x      ␣
↪1486  0x     1c9e
...
stripTif(): Tell: 0x000d2b4e Len: 0x000000f6 TIF: TIF  True >:  0x        0  0x     ␣
↪d2748  0x    d2c44
stripTif(): Tell: 0x000d2c50 Len: 0x0000003e TIF: TIF  True >:  0x        0  0x     ␣
↪d2b42  0x    d2c8e
  CPU time =    0.022 (S)
Exec. time =    0.022 (S)
Bye, bye!
```

DeTIF with write:

```
$ ``tdlisdetif`` RW.lis RW_No_TIF.lis
Cmd: DeTif.py RW.lis RW_No_TIF.lis
  CPU time =    0.019 (S)
Exec. time =    0.019 (S)
Bye, bye!
```

`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 scan 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 scan 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>
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 is designed to measure the performance of loading and iterating across a frame-set.

1.6 Usage In a Python Project

To use TotalDepth:

```
import TotalDepth
```

1.6.1 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 necesary.

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 outptu of a specific channel. Each row is often referred to as a *Frame*

1.7 Processing LIS Files

1.7.1 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*

1.7.2 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*

1.7.3 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)
```

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`

1.8 Processing LAS Files

TODO:

1.9 TotalDepth Technical Notes

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

Contents:

1.9.1 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: *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>
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

[1] Illustrative only, it is slightly more complicated than this.

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: *Run Length Encoding Module*.

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.

Module

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

For reference documentation see: *Type01Plan*.

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.

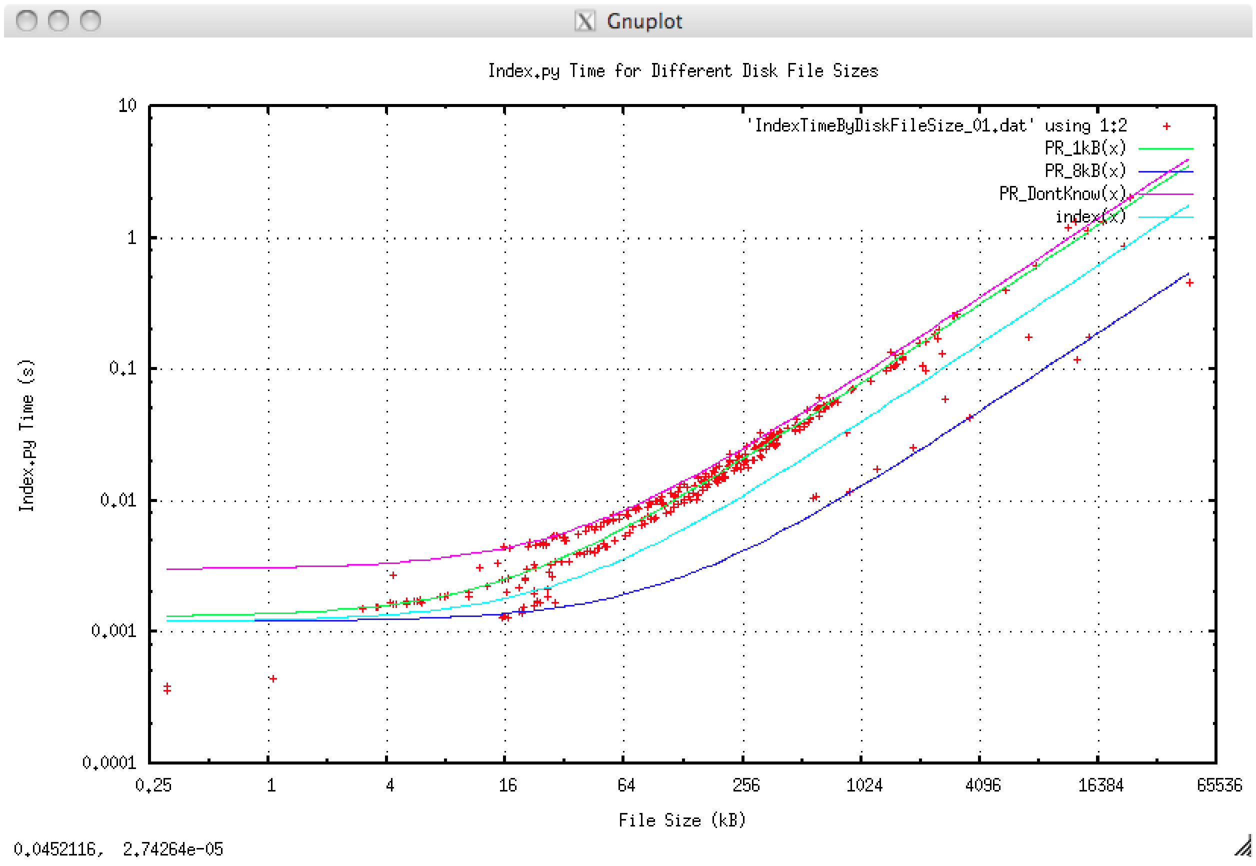
The original targets for the cost of TotalDepth LIS indexing were:

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

[2] Well not that simple, there is indirect X-axis to take into account and several other things.

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:

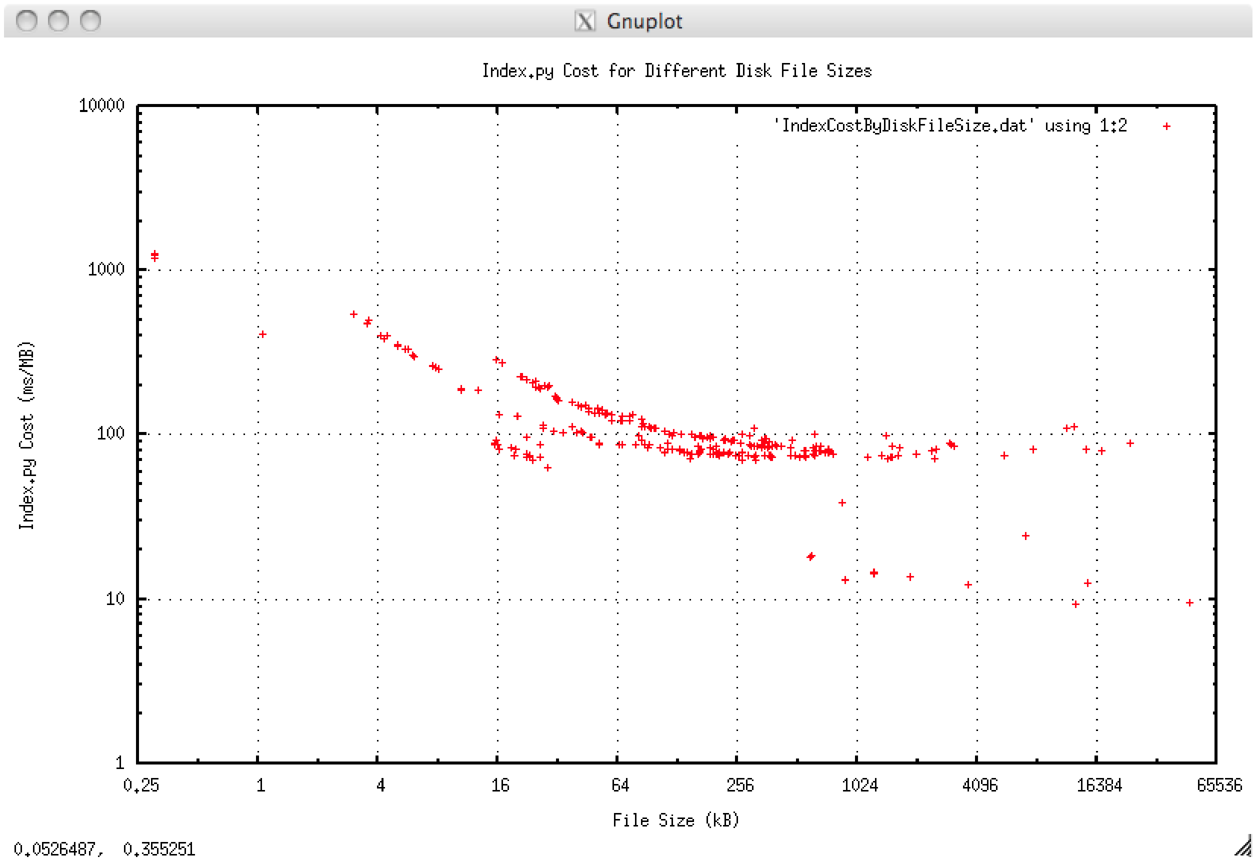
Line	Description	Colour	Latency (ms)	Cost (ms/Mb)	Result
PR_1kB(x)	Physical Record length = 1kb	Green	1.3	80	Satisfactory
PR_8kB(x)	Physical Record length = 8kb	Blue	1.2	12	Excellent
PR_DontKnow	Upper bound, worst case	Magenta	3.0	90	Satisfactory
index(x)	Regression fit on all points	Cyan	1.2	39.9	Good

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:

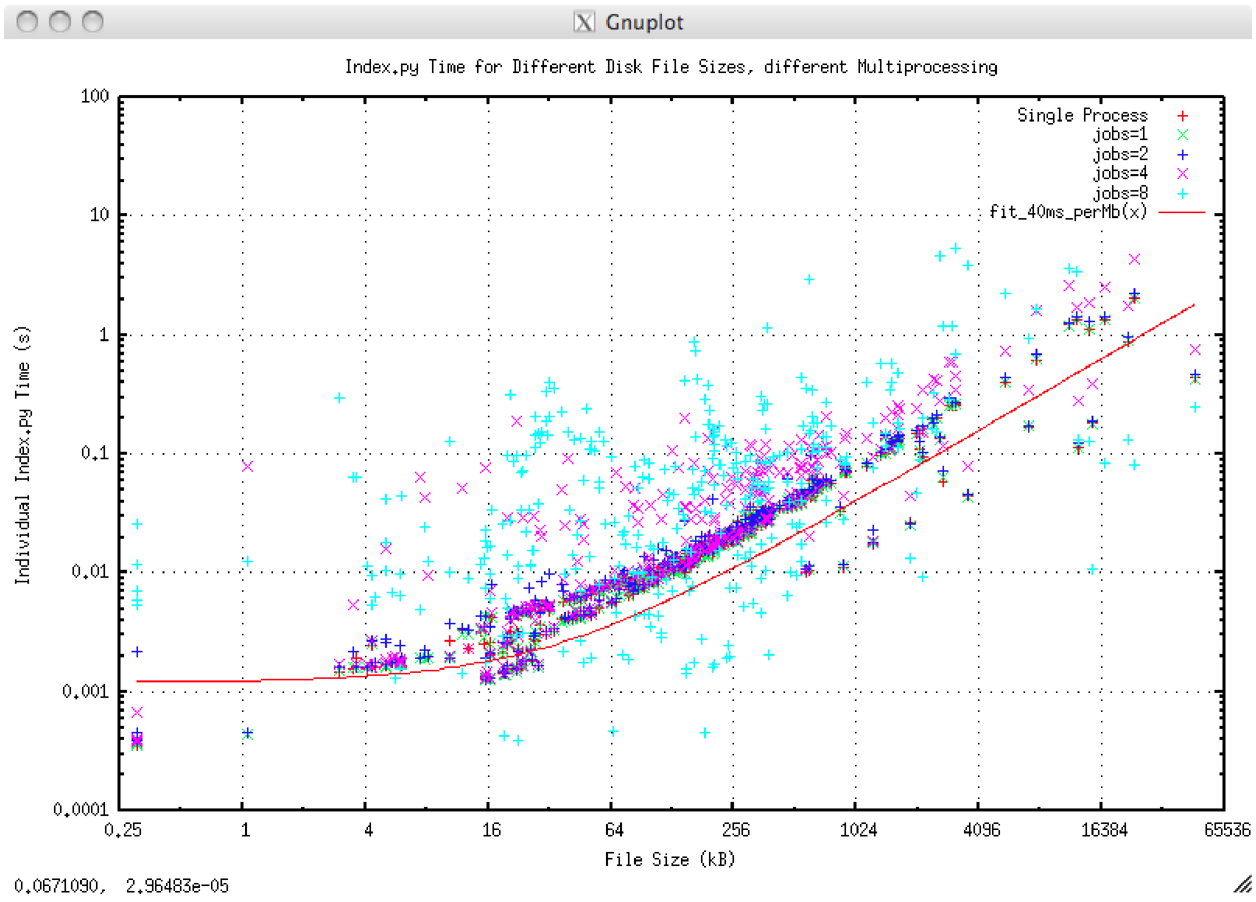


The rising costs for smaller files is no real cause for concern, this represents the startup cost of indexing and reading of around 1 to 2 ms. This graph clearly demonstrates O(N), or better, behaviour.

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.

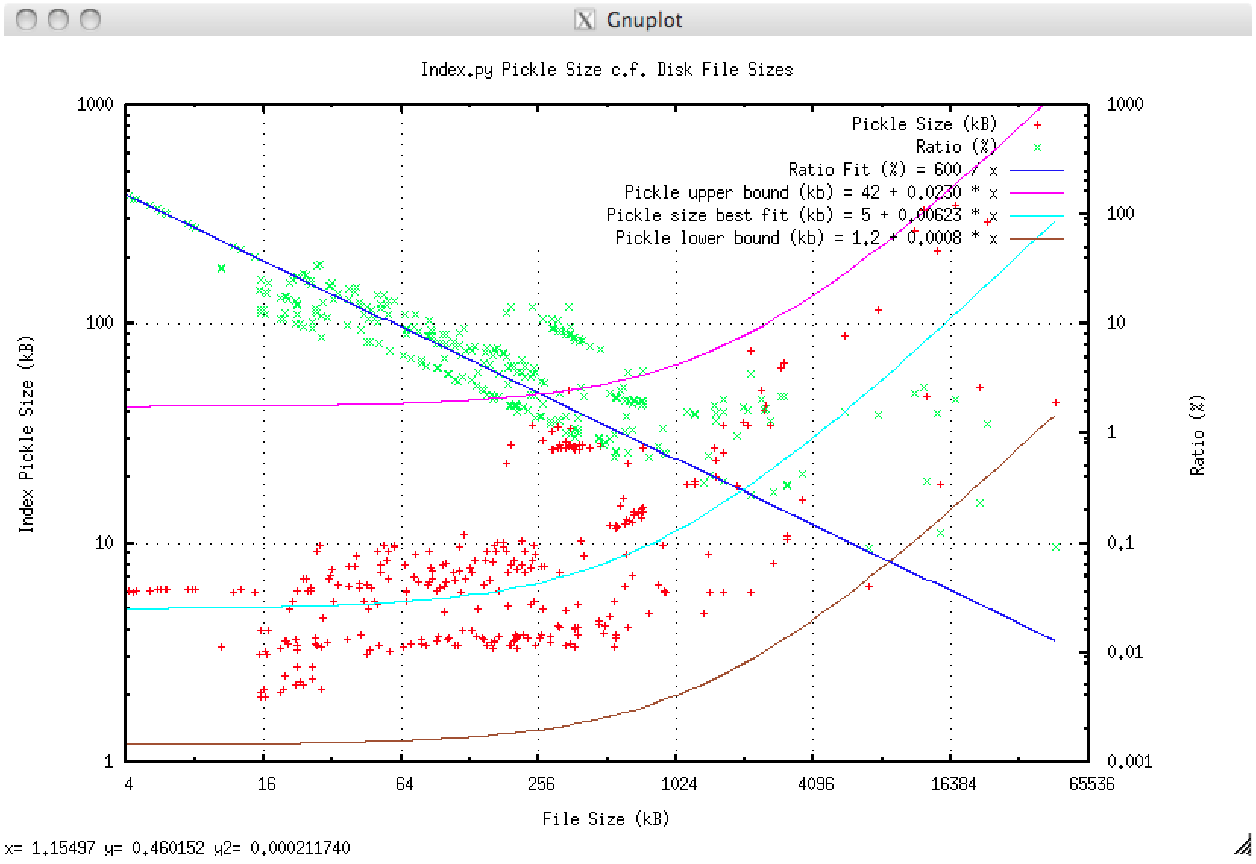
- Serialised in human readable form, such as XML, onto the file system or database[3].
- Serialised in binary form and attached as a Logical Record to the LIS file it refers to.

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):



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.

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

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 *Performance of TotalDepth*

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.

1.9.2 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)
```

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.

Actual LIS Performance

Actual performance results of some LIS file operations.

File Read Cost

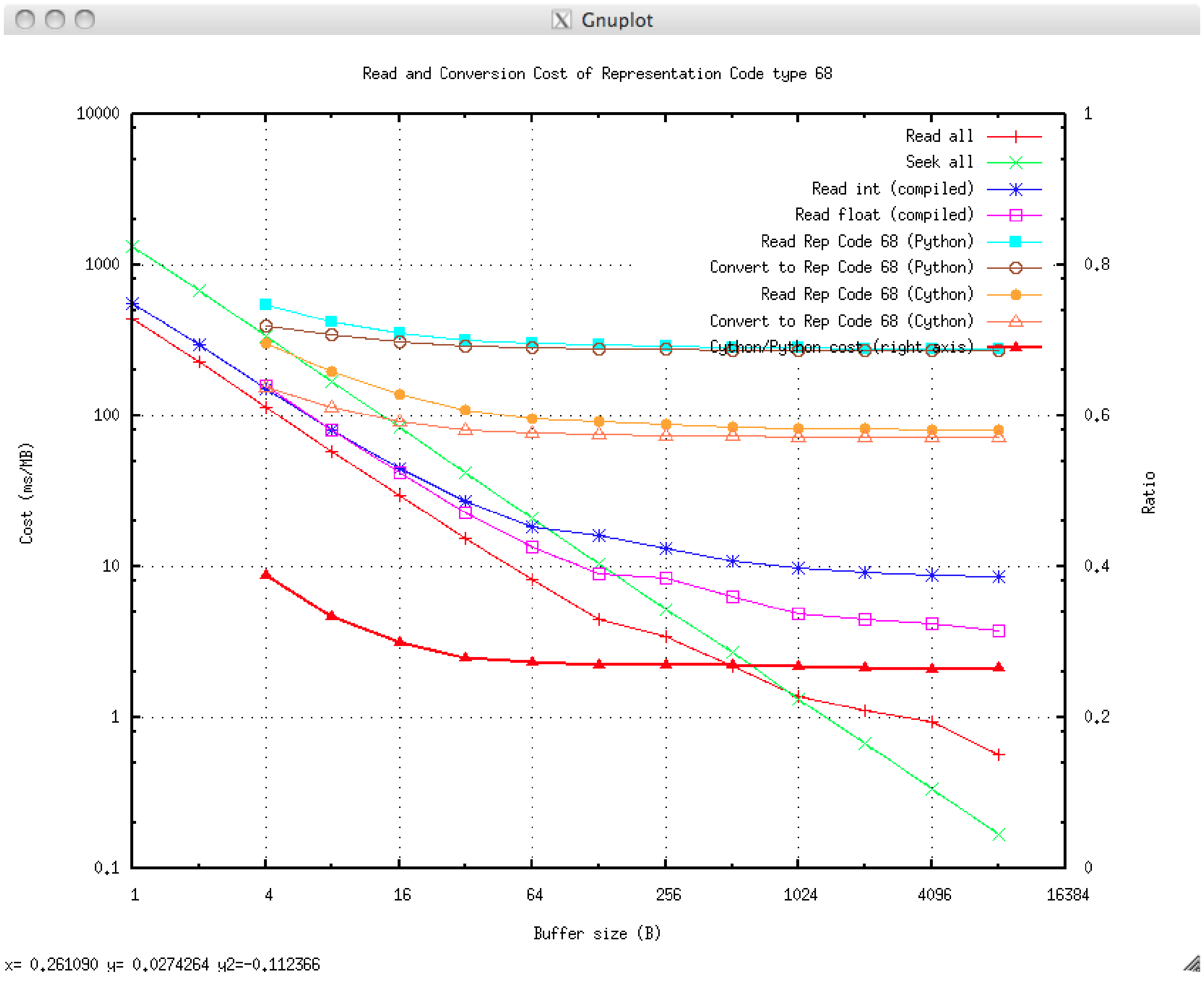
The measure here is low level reading and conversion from binary words to internal number formats. Three operations are being measured:

- 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.

[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.



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:

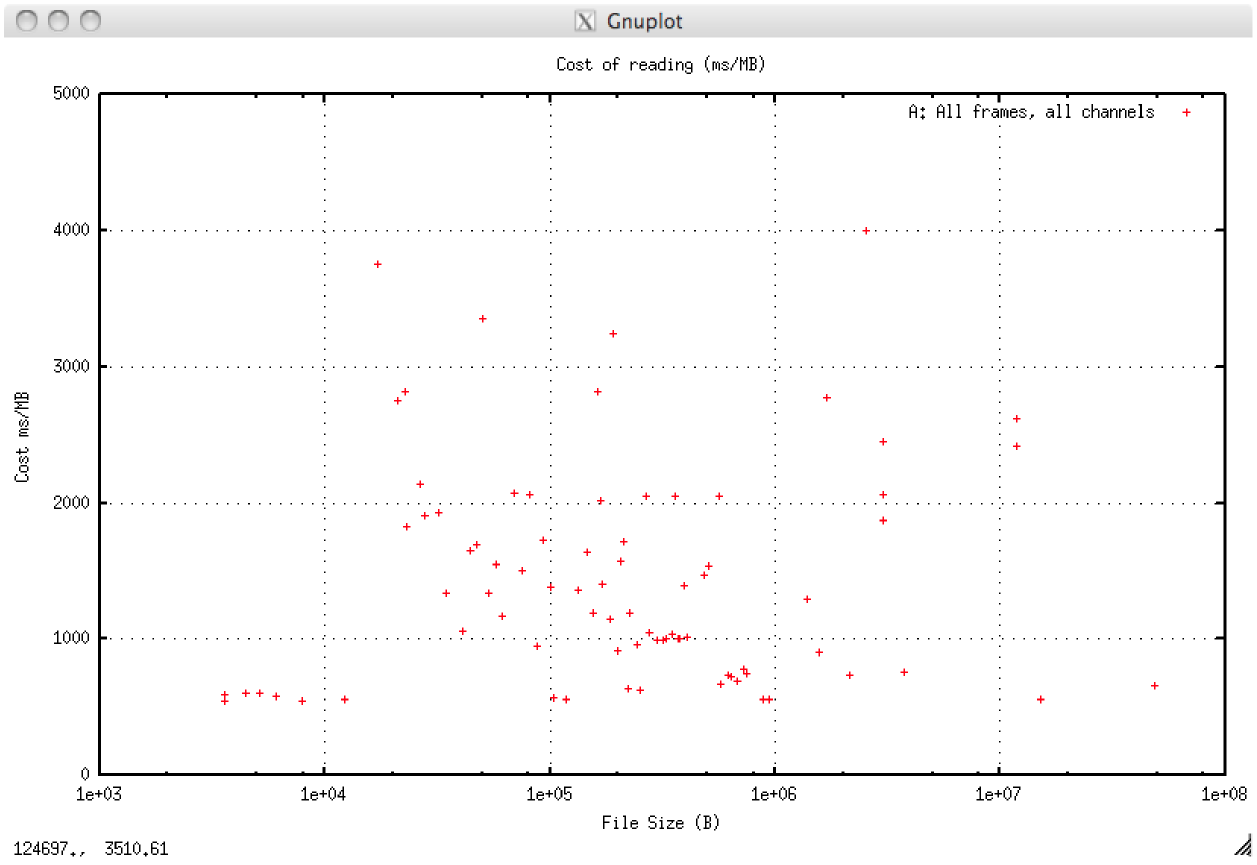
- Method A: All frames all channels. Here indexing provides no optimisation as it is a simple sequential read operation of everything.
- Method C: A sequential subset of the frames and a sequential subset of the channels. This requires a logical seek operation to the start of the frames then, within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



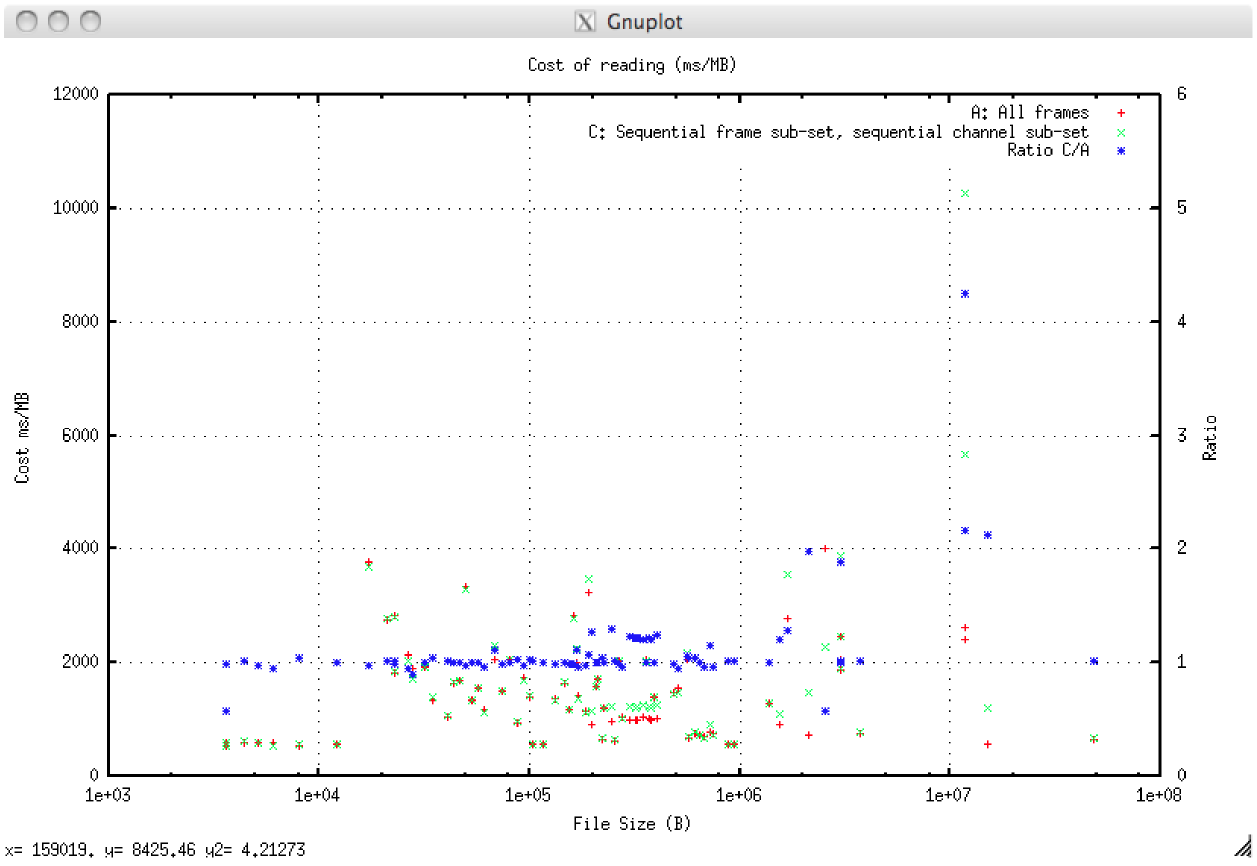
Although there is quite a spread there is, gratifyingly, no obvious trend for the cost to increase as the file size does which would indicate worse than O(N) behaviour. The behaviour is clearly O(N). The cost in ms/Mb is:

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

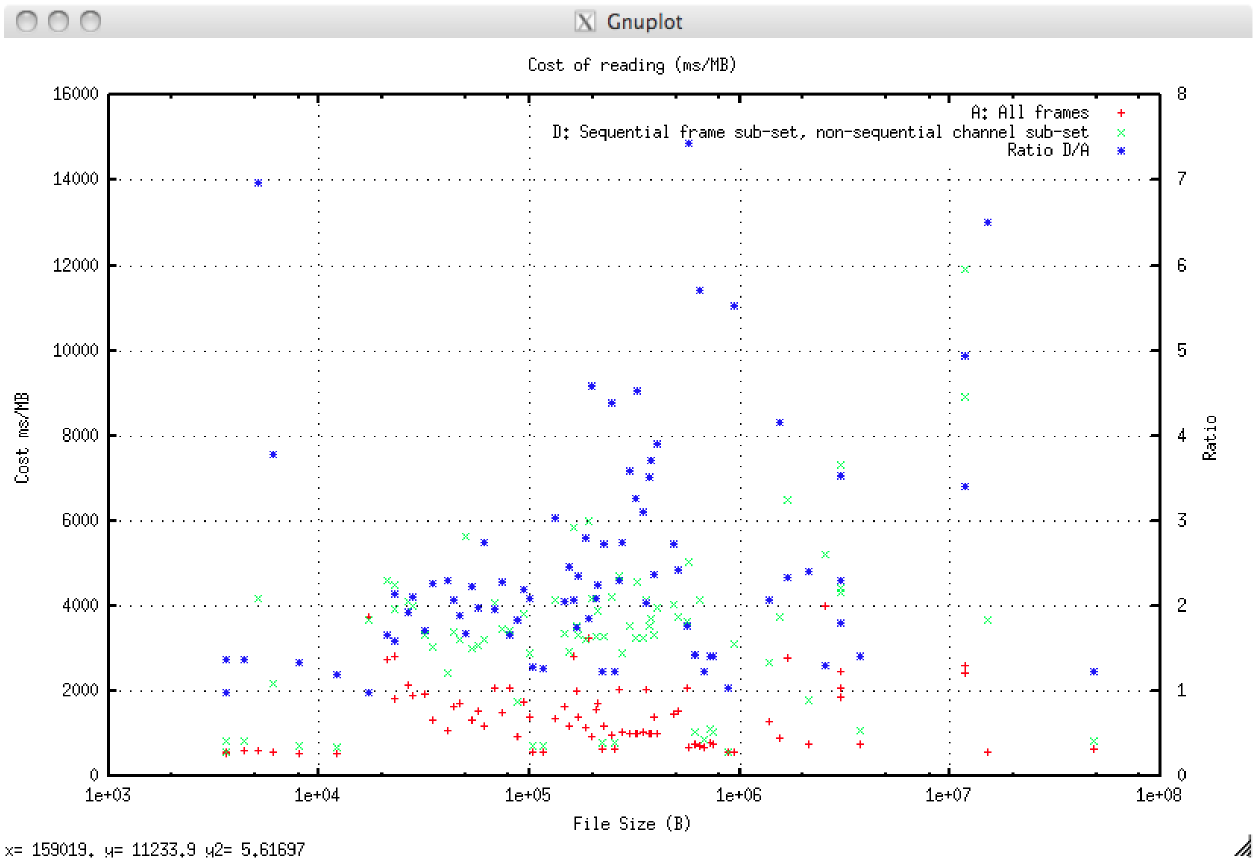
This means that a small increase in cost (13%) is the price of a hugely reduced data set.

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).

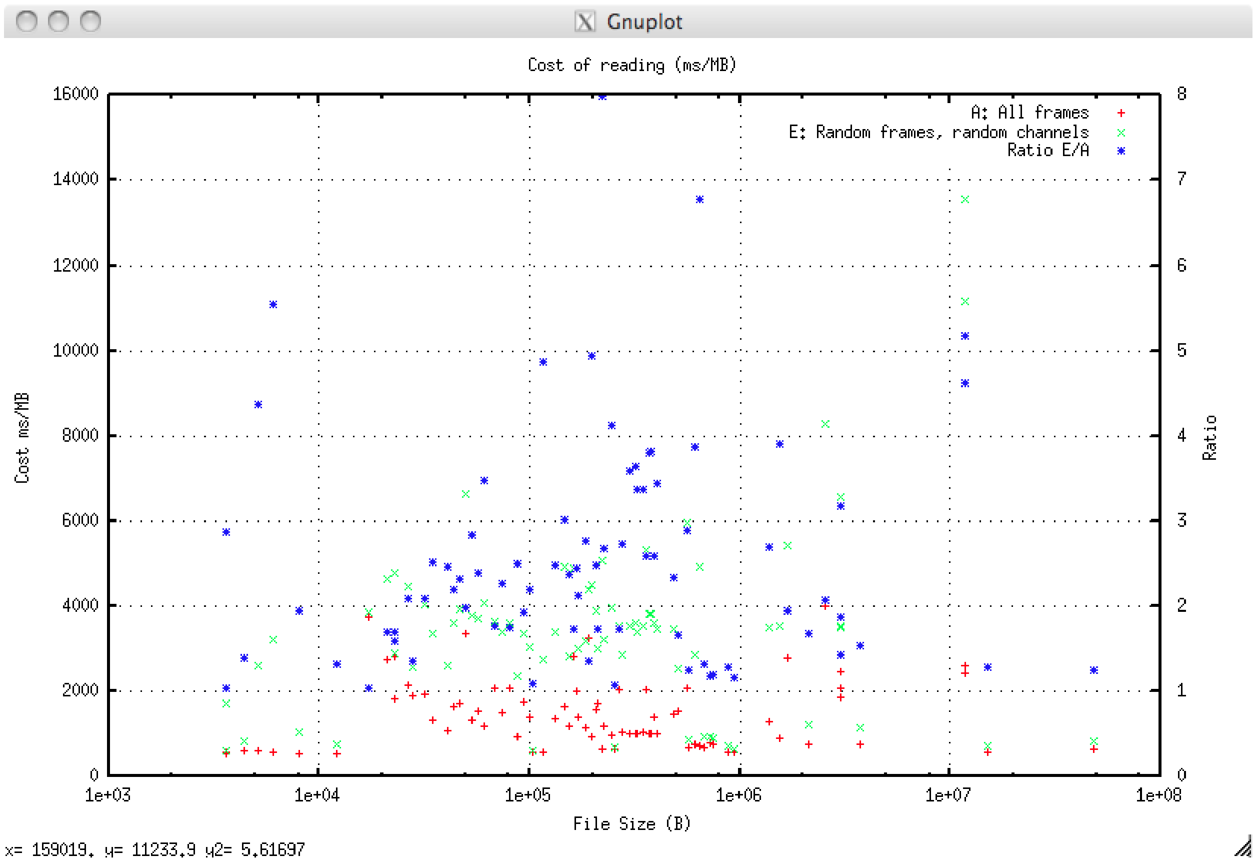
Evens so provided the number of channels of interest is less than 40% of the total indexing this way provides a real speed improvement. The important point being this may not produce an improvement (and may actually be slower) for small frame sizes; they are not the problem precisely because they are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



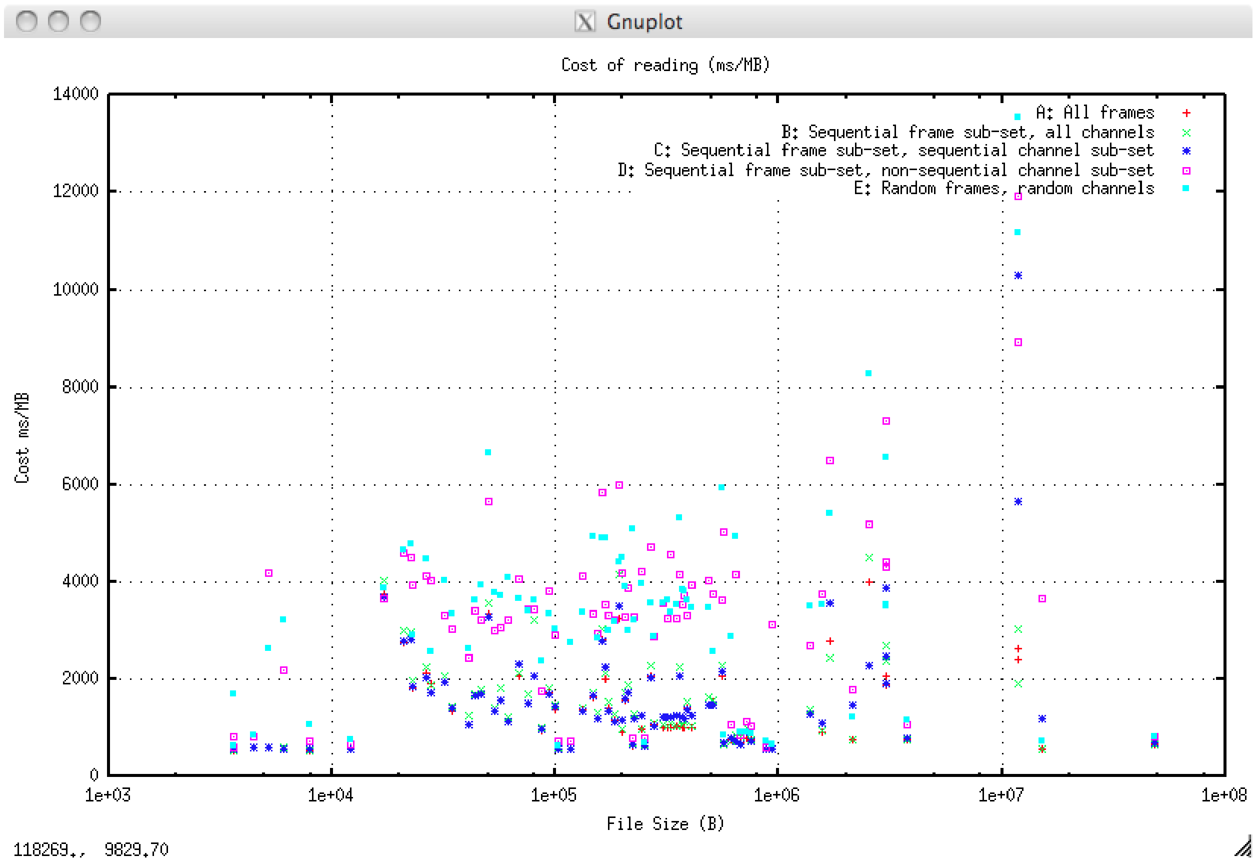
The cost in ms/Mb is:

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

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.

1.9.3 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.

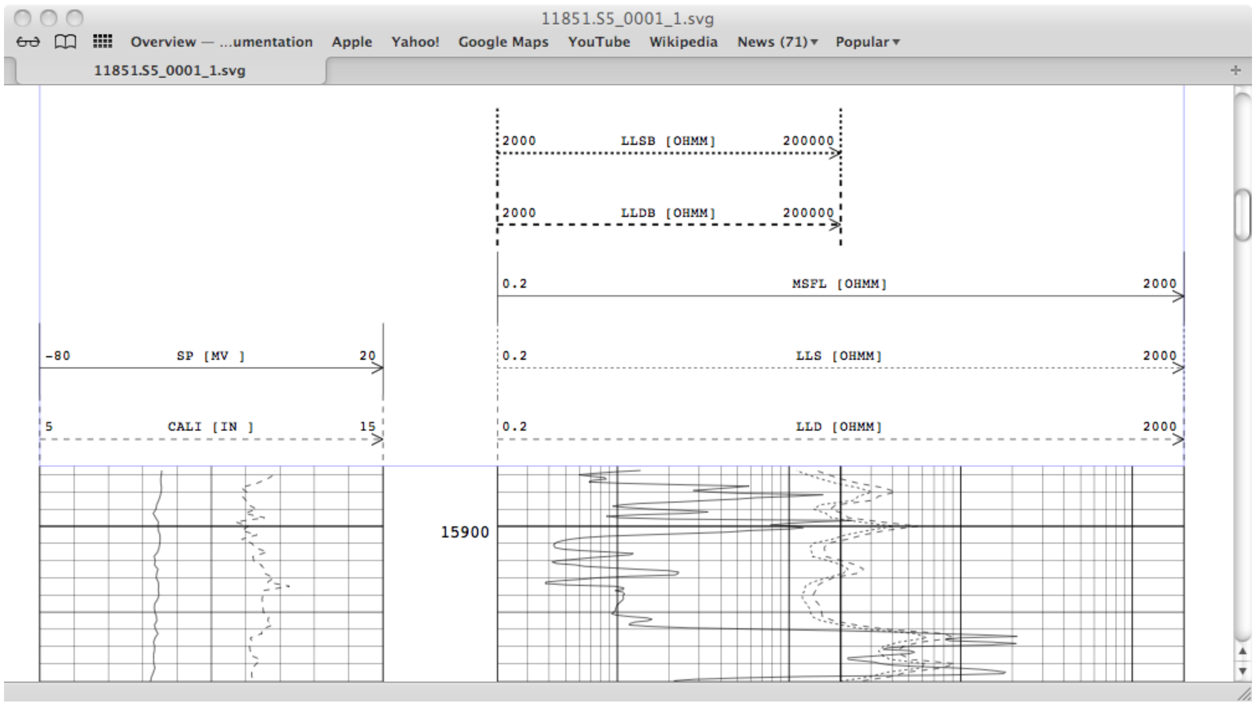
Internal Plot Algorithms

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

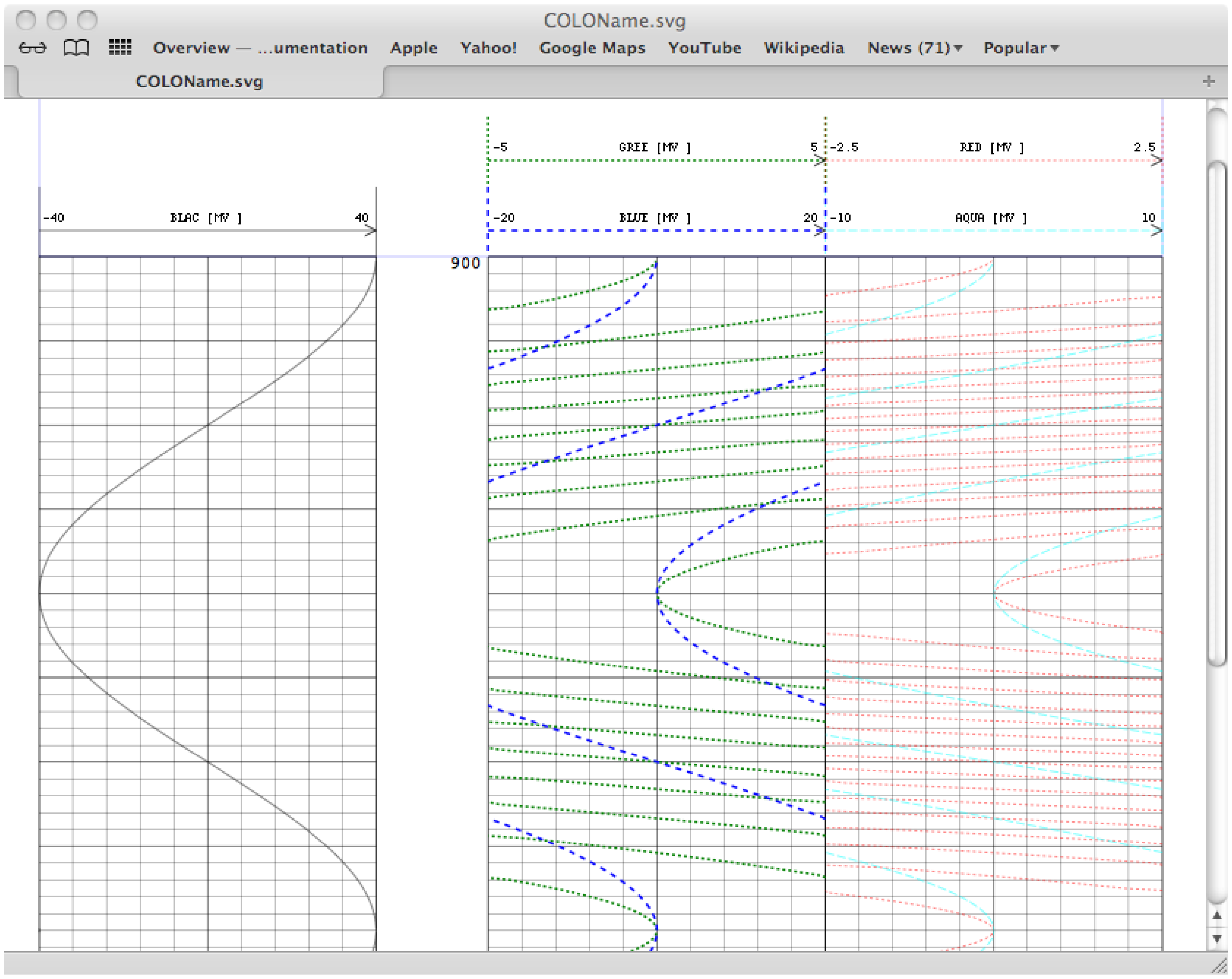
Industry standard wireline plots have the notion of curve backups where if the the trace for a particular curve goes off scale on one side of the track it might well reappear from the alternate side.

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:

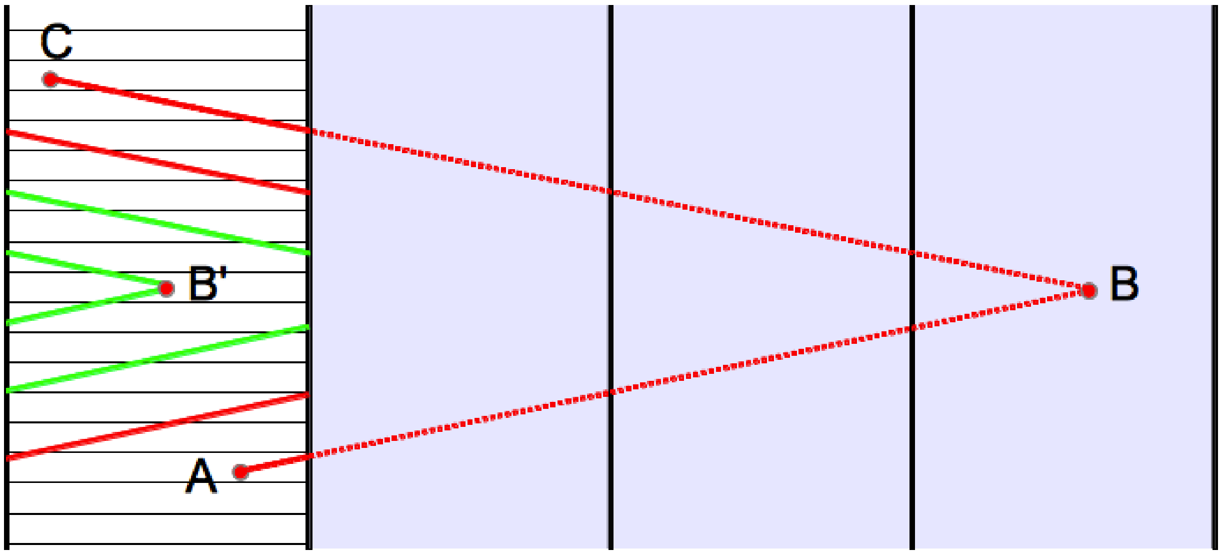
- In green on track two with a scale of +/- 5

[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.

- 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_ONCE	Single backup to left or right	'SHIF'	'1'
(-2, 2)	BACKUP_TWICE	Two backups to left or right		'2'
(-1, 1)	BACKUP_LEFT	Single backup to left only		'LG_LEFT_WRAPPED'
(1, 1)	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).

The method that does this is The `Plot._filterCrossLineList()` method

1.9.4 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

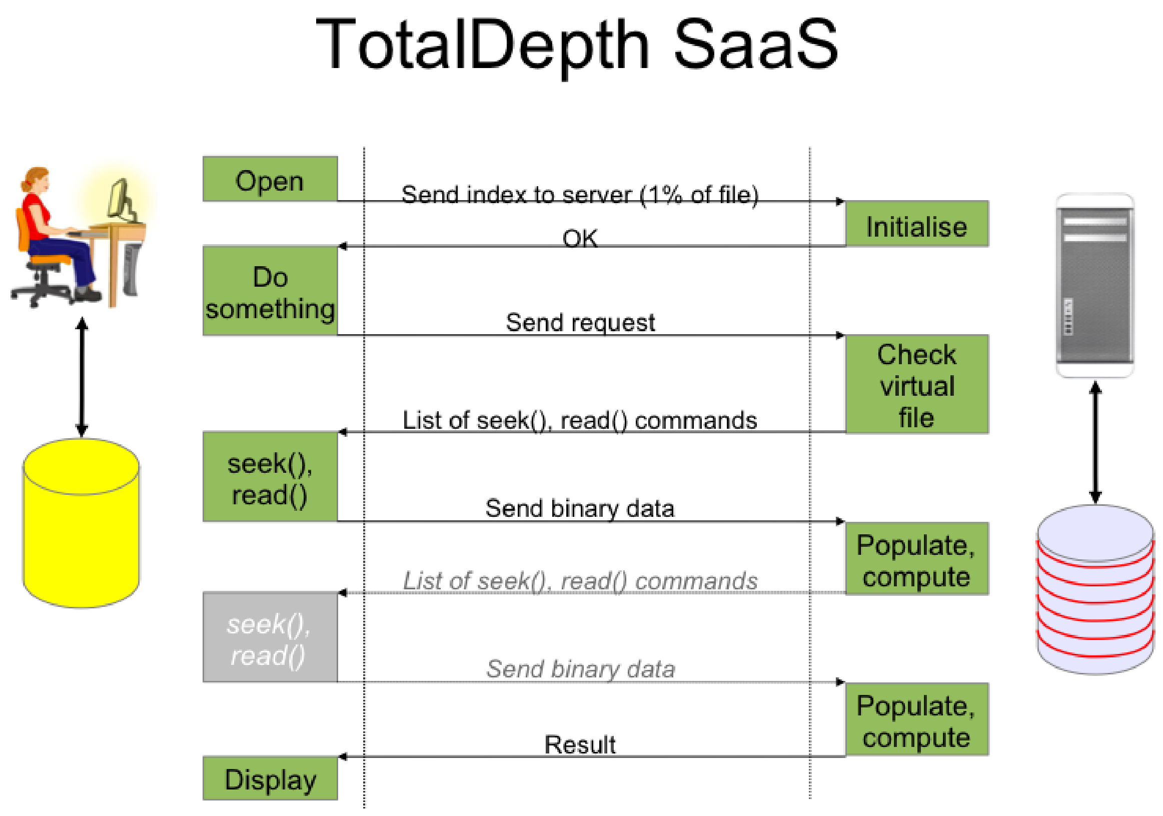
There is no particular challenge here as the server has all it needs to process any client instruction. The big disadvantage is that the client has to upload the file to the server before they can do anything. For small jobs on large files that can impose an disproportionate cost.

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.

1.10 TotalDepth and Software Test

Contents:

1.10.1 Unit Tests

This describes how unit testing is done in TotalDepth.

Module Unit Testing

Most modules have their unit tests in the `test/` directory in the package the module is in. The test code is usually in `Test`*`Module`*`.py`.

Example

For the module `TotalDepth.LIS.core.LogPass` the unit tests are in `TotalDepth/LIS/core/test/TestLogPass.Py` and they can be run thus:

```
$ cd <TOTALDEPTH>/src/TotalDepth/LIS/core
$ python3 test/TestLogPass.py
```

The output is:

```
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)

... 8<-------- Many lines snipped -------->8

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 0.271s

OK
CPU time =    0.275 (S)
Bye, bye!
```

Package Unit Testing

Most packages have their unit tests in the `test/UnitTests.py` in the package the module is in.

Example

For the module `TotalDepth.LAS.core` the unit tests are in `TotalDepth/LAS/core/test/UnitTests.Py` and they can be run thus:

```
$ python3 test/UnitTests.py
```

And the result is:

```
UnitTests.py script version "0.8.0", dated 2009-09-15
Author: Paul Ross
Copyright (c) Paul Ross

Command line:
test/UnitTests.py

('File:', 'test/UnitTests.py')
Testing TestEngVal
----------------------------------------------------------------------
Ran 55 tests in 0.002s

OK

8<-------- Many lines snipped -------->8

OK
Testing TestUnits
Time:    0.952 rate    102.547 k/S Time:    0.389 rate    250.863 k/S Time:    0.175␣
↪rate    556.827 k/S ----------------------------------------------------------------
↪------
Ran 8 tests in 1.517s

OK
Results
   Tests: 803
  Errors: 0
Failures: 0

CPU time =  215.986 (S)
Bye, bye!
```

Unit Testing and `coverage`

Ned Batchelor's *excellent* coverage tool can be used with these unit tests:

```
$ coverage run test/TestMnem.py
```

And the result is:

```
TestMnem.py script version "0.8.0", dated May 26, 2011
Author: Paul Ross
Copyright (c) 2011 Paul Ross.

test_00 (__main__.TestMnem)
TestMnem.test_00(): Tests setUp() and tearDown(). ... ok
test_01 (__main__.TestMnem)
TestMnem.test_01(): Basic constructor. ... ok
test_02 (__main__.TestMnem)
TestMnem.test_02(): Truncation of >4 chars. ... ok
test_03 (__main__.TestMnem)
TestMnem.test_03(): Padding of <4 chars. ... ok
...
test_70 (__main__.TestMnem)
TestMnem.test_70(): Construction with a string. ... ok
```

```
----------------------------------------------------------------------
Ran 21 tests in 0.002s

OK
CPU time =    0.003 (S)
Bye, bye!
```

Reporting coverage:

```
$ coverage report -m
```

And the result is:

```
Name                                   Stmts   Miss  Cover   Missing
---------------------------------------------------------------------------------------
↪-----
TotalDepth/src/TotalDepth/LIS/__init__      7      0   100%
TotalDepth/src/TotalDepth/__init__          6      0   100%
Mnem                                       52      0   100%
__init__                                    0      0   100%
test/TestMnem                             161     13    92%   259, 286-289, 292-296,␣
↪298-300
---------------------------------------------------------------------------------------
↪-----
TOTAL                                     226     13    94%
```

1.10.2 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)
 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!
```

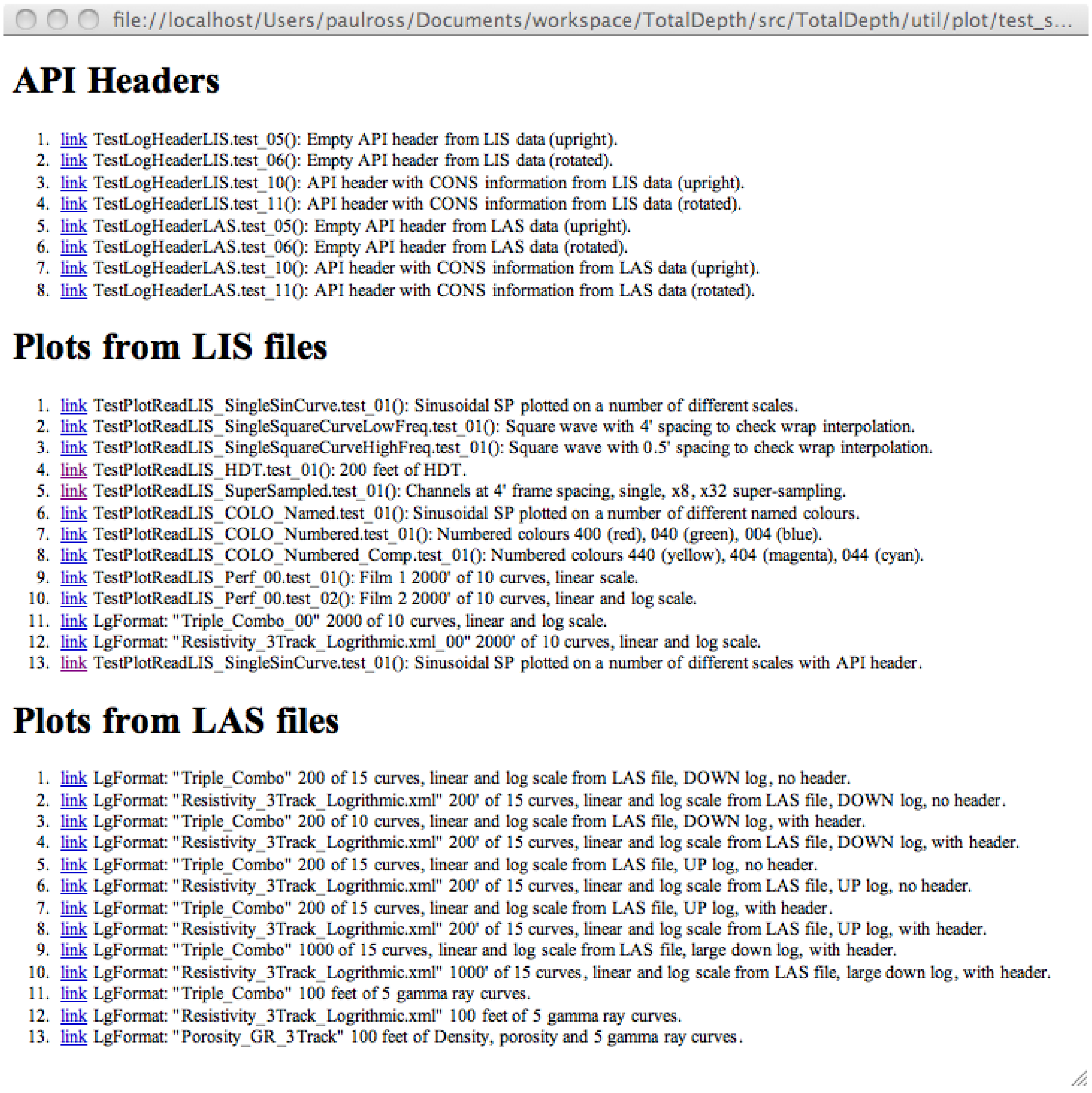
The important thing is that there should be no reported failures.

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

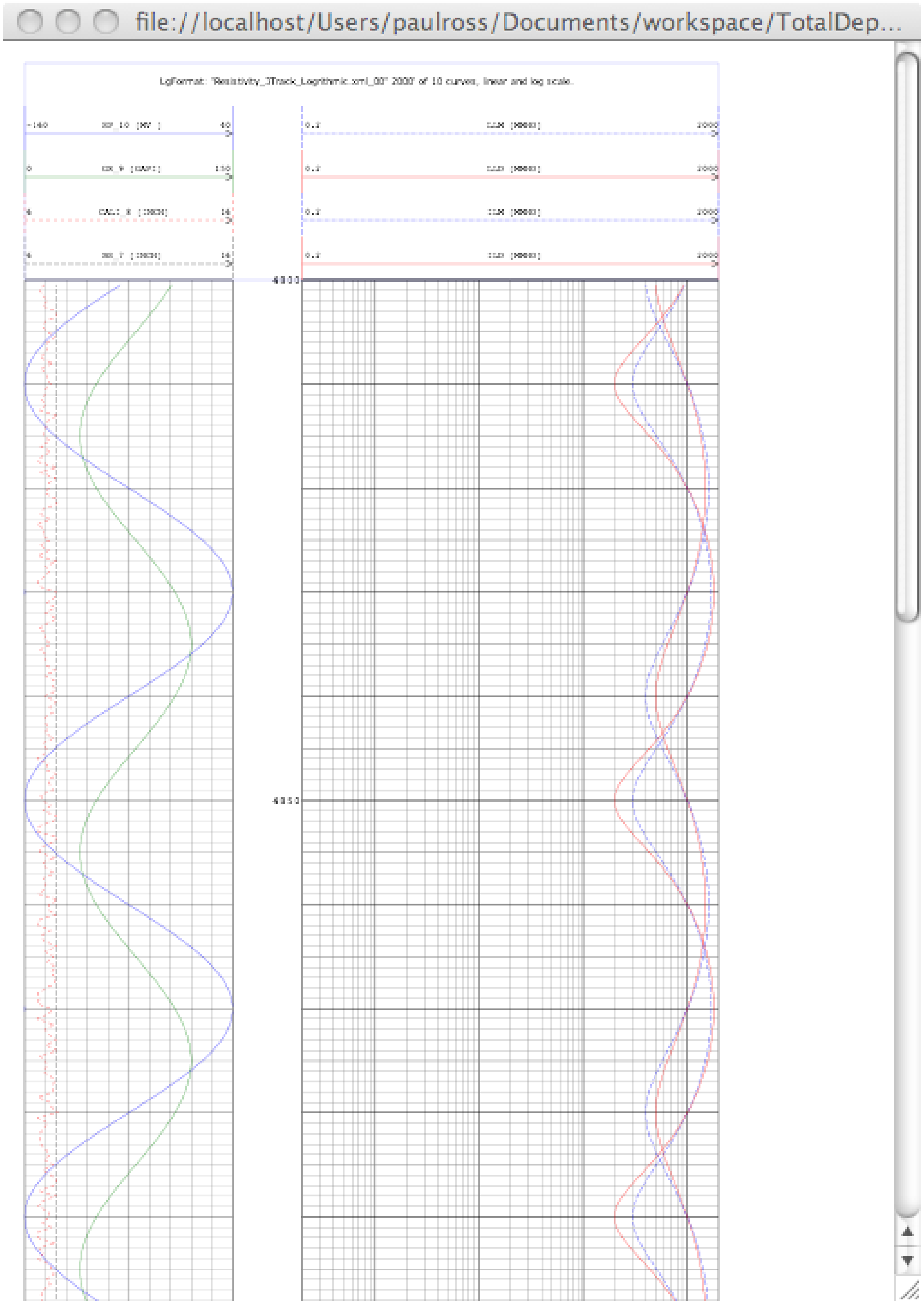
1. link TestPlotReadLIS_SingleSinCurve.test_01(): Sinusoidal SP plotted on a number of different scales.
2. link TestPlotReadLIS_SingleSquareCurveLowFreq.test_01(): Square wave with 4' spacing to check wrap interpolation.
3. link TestPlotReadLIS_SingleSquareCurveHighFreq.test_01(): Square wave with 0.5' 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.
-160
SP_10 [MV]
40
0
GR_9 [GAPI]
150
6
CALI_8 [INCH]
16
6
BS_7 [INCH]
16
0.2
ILM [OHMM]
2000
0.2
ILD [OHMM]
2000
0.2
ILM [OHMM]
2000
0.2
ILD [OHMM]
2000
4800
4850

Cool. If you see that then your TotalDepth installation is pretty good!

1.11 API Reference

Contents:

1.11.1 TotalDepth.LIS API Reference

Contents:

LIS Handling of 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.`**`ExceptionRepCodeNoLength`**
    Exception for indeterminate length when using Rep Code 65.

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.

`TotalDepth.LIS.core.RepCode.`**read49**(*theFile*)
Returns a Representation Code 49 value from a File object.

`TotalDepth.LIS.core.RepCode.`**read50**(*theFile*)
Returns a Representation Code 50 value from a File object.

`TotalDepth.LIS.core.RepCode.`**read56**(*theFile*)
Returns a Representation Code 56 value from a File object.

`TotalDepth.LIS.core.RepCode.`**read66**(*theFile*)
Returns a Representation Code 66 value from a File object.

`TotalDepth.LIS.core.RepCode.`**read68**(*theFile*)
Returns a Representation Code 68 value from a File object.

`TotalDepth.LIS.core.RepCode.`**read70**(*theFile*)
Returns a Representation Code 70 value from a File object.

`TotalDepth.LIS.core.RepCode.`**read73**(*theFile*)
Returns a Representation Code 73 value from a File object.

`TotalDepth.LIS.core.RepCode.`**read77**(*theFile*)
Returns a Representation Code 77 value from a File object.

`TotalDepth.LIS.core.RepCode.`**read79**(*theFile*)
Returns a Representation Code 79 value from a File object.

`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.`**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.`**readBytes49**(*arg*)
Returns a Representation Code 49 value from a bytes object.

`TotalDepth.LIS.core.RepCode.`**readBytes50**(*arg*)
Returns a Representation Code 50 value from a bytes object.

`TotalDepth.LIS.core.RepCode.`**readBytes56**(*arg*)
Returns a Representation Code 56 value from a bytes object.

`TotalDepth.LIS.core.RepCode.`**readBytes66**(*arg*)
Returns a Representation Code 66 value from a bytes object.

`TotalDepth.LIS.core.RepCode.`**readBytes68**(*arg*)
Returns a Representation Code 68 value from a bytes object.

`TotalDepth.LIS.core.RepCode.`**readBytes70**(*arg*)
Returns a Representation Code 70 value from a bytes object.

`TotalDepth.LIS.core.RepCode.`**readBytes73**(*arg*)
Returns a Representation Code 73 value from a bytes object.

TotalDepth.LIS.core.RepCode.**readBytes77**(*arg*)
Returns a Representation Code 77 value from a bytes object.

TotalDepth.LIS.core.RepCode.**readBytes79**(*arg*)
Returns a Representation Code 79 value from a bytes object.

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.**writeBytes**(*v*, *r*)
Takes a value v and a Representation Code r and converts this to 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.**writeBytes68**(*v*)
Converts a value to a Rep Code 68 and returns the bytes.

TotalDepth.LIS.core.RepCode.**writeBytes73**(*v*)
Converts a value to a Rep Code 73 and returns the bytes.

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.**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.**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.**from56**(*theWord*)
Returns an integer from Rep code 0x38, signed char representation.

`TotalDepth.LIS.core.pRepCode.`**`from66`**`(`*theWord*`)`
Returns a Rep code 0x42, unsigned byte from theWord, expected to be an integer.

`TotalDepth.LIS.core.pRepCode.`**`from68`**`(`*theWord*`)`
Returns a double from a Rep code 68 word (a 32 bit integer).

`TotalDepth.LIS.core.pRepCode.`**`from70`**`(`*theWord*`)`
Returns a double from a Rep code 0x46, 32bit fixed point.

`TotalDepth.LIS.core.pRepCode.`**`from73`**`(`*theWord*`)`
Returns a integer from a Rep code 0x49, 32bit signed integer.

`TotalDepth.LIS.core.pRepCode.`**`from77`**`(`*theWord*`)`
Returns a Rep code 0x4D, 8bit mask from theWord, expected to be an integer.

`TotalDepth.LIS.core.pRepCode.`**`from79`**`(`*theWord*`)`
Returns an integer from Rep code 0x4F, 16bit signed integer.

`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.`**`maxValue`**`(`*r*`)`
Returns the maximum 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.`**`minValue`**`(`*r*`)`
Returns the minimum value for various representation codes. May raise ExceptionRepCodeUnknown.

`TotalDepth.LIS.core.pRepCode.`**`read49`**`(`*theFile*`)`
Returns a Representation Code 49 value from a File object.

`TotalDepth.LIS.core.pRepCode.`**`read50`**`(`*theFile*`)`
Returns a Representation Code 50 value from a File object.

`TotalDepth.LIS.core.pRepCode.`**`read56`**`(`*theFile*`)`
Returns a Representation Code 56 value from a File object.

`TotalDepth.LIS.core.pRepCode.`**`read66`**`(`*theFile*`)`
Returns a Representation Code 66 value from a File object.

`TotalDepth.LIS.core.pRepCode.`**`read68`**`(`*theFile*`)`
Returns a Representation Code 68 value from a File object.

`TotalDepth.LIS.core.pRepCode.`**`read70`**`(`*theFile*`)`
Returns a Representation Code 70 value from a File object.

`TotalDepth.LIS.core.pRepCode.`**`read73`**`(`*theFile*`)`
Returns a Representation Code 73 value from a File object.

`TotalDepth.LIS.core.pRepCode.`**`read77`**`(`*theFile*`)`
Returns a Representation Code 77 value from a File object.

`TotalDepth.LIS.core.pRepCode.`**`read79`**`(`*theFile*`)`
Returns a Representation Code 79 value from a File object.

`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.`**`to50`**`(`*theVal*`)`
: Converts a double to Rep code 0x32, 32bit floating point representation.

`TotalDepth.LIS.core.pRepCode.`**`to56`**`(`*theVal*`)`
: Converts a double to Rep code 0x38, signed char representation.

`TotalDepth.LIS.core.pRepCode.`**`to66`**`(`*theVal*`)`
: Converts theVal to representation code 66 integer from theWord.

`TotalDepth.LIS.core.pRepCode.`**`to68`**`(`*v*`)`
: Returns Representation code 68 as a 32 bit integer from a double.

`TotalDepth.LIS.core.pRepCode.`**`to70`**`(`*v*`)`
: Returns Rep code 0x46, 32bit fixed point from an int or double.

`TotalDepth.LIS.core.pRepCode.`**`to73`**`(`*v*`)`
: Returns Rep code 0x49, 32bit signed integer from an int or double.

`TotalDepth.LIS.core.pRepCode.`**`to77`**`(`*theVal*`)`
: Converts theVal to Rep code 0x4D, 8bit mask from theVal.

`TotalDepth.LIS.core.pRepCode.`**`to79`**`(`*theVal*`)`
: Converts a double to Rep code 0x4F, 16bit signed integer.

`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.

`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.

Testing

The unit tests are in `test/TestRepCode.py` and `test/TestRepCode68.py`.

Units

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.`**`ExceptionUnitsMissmatchedCategory`**
When a two units do not exist in the same category.

exception `TotalDepth.LIS.core.Units.`**`ExceptionUnitsNoUnitInCategory`**
When a unit does not exist in a category.

exception `TotalDepth.LIS.core.Units.`**`ExceptionUnitsUnknownCategory`**
When a unit category does not exist.

exception `TotalDepth.LIS.core.Units.`**`ExceptionUnitsUnknownUnit`**
When a unit does not exist.

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.

`convert`(*val*, *other*)
Convert a value from me to the other where other is a UnitConvert object.

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.

convert(*v*, *u_1*, *u_2*)

Returns a value converted from one units to another. e.g. convert(1.2, "FEET", "INCH")

unitConvertor(*u*)

Returns a UnitConvert object corresponding to the name u. Will raise a ExceptionUnitsNoUnitInCategory if not found.

units()

Reuturns a list of unit names for this category.

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.**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.**hasUnit**(*u*)

Returns True if I have that unit e,g, b"FEET".

TotalDepth.LIS.core.Units.**hasUnitCategory**(*c*)

Returns True if I have that unit category e.g. b"TIME".

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.

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.**retUnitConvert**(*u*)

Returns a UnitConvert object for the unit. May raise a ExceptionUnits or descendent.

TotalDepth.LIS.core.Units.**retUnitConvertCategory**(*c*)

Returns a UnitConvertCategory object for the category. May raise a ExceptionUnits or descendent.

TotalDepth.LIS.core.Units.**unitCategories**()

Returns a list of the unit categories.

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.**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.

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`.

EngVal

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

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).

`__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.

`__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.

`__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.

`__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.

`__iadd__`(*other*)
Addition 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.

`__isub__`(*other*)
Subtraction in place, other must be a real number or an EngVal where it is converted to my units.

`__itruediv__`(*other*)
Overload self /= other. other must be a real number or a dimensionless EngVal.

`__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.

`__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.

`__mul__`(*other*)
Overload self*other. other must be a real number or a dimensionless EngVal.

`__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.

`__radd__`(*other*)
Right value addition, see __add__().

`__rmul__`(*other*)
Right value multiplication, see __mul__().

`__rsub__`(*other*)
Right value subtraction, see __sub__().

`__rtruediv__`(*other*)
Right value true division, see __truediv__().

`__str__`()
String representation.

`__sub__`(*other*)
Overload self-other, returned result has the sum of self and other. The units chosen are self's.

`__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.

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

`convert`(*theUnits*)
Convert my value to the supplied units in-place. May raise an ExceptionUnits.

`dimensionless`()
Returns True if the measure is dimensionless.

`getInUnits`(*theUnits*)
Returns my value 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'.

`newEngValInUnits`(*theUnits*)
Returns a new EngVal converting me to the supplied units. May raise an ExceptionUnits.

`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.

class `TotalDepth.LIS.core.EngVal.`**`EngValRc`**(*theVal*, *theUom*, *theRc=None*)
An engineering value with a integer Representation Code.

`__str__`()
String representation.

`encode`()
Encode my value to my RepCode returning a bytes object. May raise an ExceptionEngVal.

Mnem

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.

`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.

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

`__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.

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.

`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.

`__weakref__`

list of weak references to the object (if defined)

`write`(*theB*)

Writes theB bytes.

`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.

LIS 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).

class `TotalDepth.LIS.core.File.`**`FileRead`**(*theFile*, *theFileId=None*, *keepGoing=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.

`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.

`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.

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()).

write(*theLr*)
Writes the Logical Record to the file. Returns the tell() of the start of the LR.

close()
Closes the file.

TIF Marker Handling

TIF Markers

These are 3x32bit big-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 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
...
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_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.

`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_ALL_FORMAT = <Struct object>`**

struct.Struct() format for a TIF marker

`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_WORD_BYTES = 4`**

Number of bytes in a TIF word

`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

`class` `TotalDepth.LIS.core.TifMarker.`**`TifMarkerBase`**(*raiseOnError=True*)
Base class for TIF markers.

`eof`
True if I have encountered a EOF marker.

`markers`()
Current values of markers as a tuple of three integers.

`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.

`reset`()
Resets the TIF markers to all zero, this means hasPrevious is False.

`strHeader`()
Header string for an ASCII dump.

`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.

`hasPrevious`
True if a Physical Record has been read, cleared on reset().

`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.

`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.

`class` `TotalDepth.LIS.core.TifMarker.`**`TifMarkerWrite`**
Class for writing TIF markers.

`close`(*theStream*)
Write TIF EOF markers.

`write`(*theStream*, *theLen*)
Write TIF markers to a RawStream object. theLen must be the length of the Physical Record including the PRH and PRT.

Physical Record Handler

Physical Record Attributes

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

`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 = -32768`**
The minimum record number

`TotalDepth.LIS.core.PhysRec.`**`PR_PRT_REC_NUM_MAX = 32767`**
The maximum record number

`TotalDepth.LIS.core.PhysRec.`**`PR_PRT_FILE_NUM_FORMAT = <Struct object>`**
The struct.Struct() format for the Physical Record Trailer file number

`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 = -32768`**
The minimum file number

`TotalDepth.LIS.core.PhysRec.`**`PR_PRT_FILE_NUM_MAX = 32767`**
The maximum file number

`TotalDepth.LIS.core.PhysRec.`**`PR_PRT_CHECKSUM_FORMAT = <Struct object>`**
The struct.Struct() format for the Physical Record Trailer checksum

`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*, *keepGoing*)

Base class for physical record read and write. TODO: Checksum reading, writing and testing.

close()

Close the underlying stream, further operations will raise a ValueError.

strHeader()

Returns the header string to go at the top of a list of __str__().

class TotalDepth.LIS.core.PhysRec.**PhysRecRead**(*theFile*, *theFileId=None*, *keepGoing=False*)

Specialisation of PhysRecBase for reading streams.

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.

class TotalDepth.LIS.core.PhysRec.**PhysRecTail**(*hasRecNum=False*, *fileNum=None*, *hasCheckSum=False*)

Represents Physical Record Tail fields.

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.

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.

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.

Logical Record Handler

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**
Normal data record containing log data

TotalDepth.LIS.core.LogiRec.**LR_TYPE_ALTERNATE_DATA = 1**
Alternate data.

TotalDepth.LIS.core.LogiRec.**LR_TYPE_JOB_ID = 32**
Job identification

TotalDepth.LIS.core.LogiRec.**LR_TYPE_WELL_DATA = 34**
Well site data

TotalDepth.LIS.core.LogiRec.**LR_TYPE_TOOL_INFO = 39**
Tool string info

TotalDepth.LIS.core.LogiRec.**LR_TYPE_ENCRYPTED_TABLE = 42**
Encrypted table dump

TotalDepth.LIS.core.LogiRec.**LR_TYPE_TABLE_DUMP = 47**
Table dump

TotalDepth.LIS.core.LogiRec.**LR_TYPE_DATA_FORMAT = 64**
Data format specification record

TotalDepth.LIS.core.LogiRec.**LR_TYPE_DATA_DESCRIPTOR = 65**
Data descriptor (not defined in the LIS79 Description Reference Manual)

TotalDepth.LIS.core.LogiRec.**LR_TYPE_TU10_BOOT = 95**
TU10 software boot

TotalDepth.LIS.core.LogiRec.**LR_TYPE_BOOTSTRAP_LOADER = 96**
Bootstrap loader

TotalDepth.LIS.core.LogiRec.**LR_TYPE_CP_KERNEL = 97**
CP-kernel loader boot

TotalDepth.LIS.core.LogiRec.**LR_TYPE_PROGRAM_FILE_HEAD = 100**
Program file header

TotalDepth.LIS.core.LogiRec.**LR_TYPE_PROGRAM_OVER_HEAD = 101**
Program overlay header

TotalDepth.LIS.core.LogiRec.**LR_TYPE_PROGRAM_OVER_LOAD = 102**
Program overlay load

TotalDepth.LIS.core.LogiRec.**LR_TYPE_FILE_HEAD = 128**
File header

TotalDepth.LIS.core.LogiRec.**LR_TYPE_FILE_TAIL = 129**
File trailer

TotalDepth.LIS.core.LogiRec.**LR_TYPE_TAPE_HEAD = 130**
Tape header

TotalDepth.LIS.core.LogiRec.**LR_TYPE_TAPE_TAIL = 131**
Tape trailer

TotalDepth.LIS.core.LogiRec.**LR_TYPE_REEL_HEAD = 132**
Reel header

TotalDepth.LIS.core.LogiRec.**LR_TYPE_REEL_TAIL = 133**
Reel trailer

TotalDepth.LIS.core.LogiRec.**LR_TYPE_EOF = 137**
Logical EOF (end of file)

TotalDepth.LIS.core.LogiRec.**LR_TYPE_BOT = 138**
Logical BOT (beginning of tape)

TotalDepth.LIS.core.LogiRec.**LR_TYPE_EOT = 139**
Logical EOT (end of tape)

TotalDepth.LIS.core.LogiRec.**LR_TYPE_EOM = 141**
Logical EOM (end of medium)

TotalDepth.LIS.core.LogiRec.**LR_TYPE_OPERATOR_INPUT = 224**
Operator command inputs

TotalDepth.LIS.core.LogiRec.**LR_TYPE_OPERATOR_RESPONSE = 225**
Operator response inputs

TotalDepth.LIS.core.LogiRec.**LR_TYPE_SYSTEM_OUTPUT = 227**
System outputs to operator

TotalDepth.LIS.core.LogiRec.**LR_TYPE_FLIC_COMMENT = 232**
FLIC comment

TotalDepth.LIS.core.LogiRec.**LR_TYPE_BLANK_RECORD = 234**
Blank record/CSU comment

TotalDepth.LIS.core.LogiRec.**LR_TYPE_PICTURE = 85**
Picture

`TotalDepth.LIS.core.LogiRec.`**`LR_TYPE_IMAGE = 86`**
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.`**`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 d`**
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.

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.

class `TotalDepth.LIS.core.LogiRec.`**LrEOF**(*attr=0*)
A EOF marker record.

class `TotalDepth.LIS.core.LogiRec.`**LrEOFRead**(*theFile*)
A EOF marker record read from a file.

class `TotalDepth.LIS.core.LogiRec.`**LrBOT**(*attr=0*)
A BOT marker record.

class `TotalDepth.LIS.core.LogiRec.`**LrBOTRead**(*theFile*)
A BOT marker record read from a file.

class `TotalDepth.LIS.core.LogiRec.`**LrEOT**(*attr=0*)
A EOT marker record.

class `TotalDepth.LIS.core.LogiRec.`**LrEOTRead**(*theFile*)
A EOT marker record read from a file.

class `TotalDepth.LIS.core.LogiRec.`**LrEOM**(*attr=0*)
A EOM marker record.

class `TotalDepth.LIS.core.LogiRec.`**LrEOMRead**(*theFile*)
A EOM marker record read from a file.

class `TotalDepth.LIS.core.LogiRec.`**LrWithDateField**(*theType*, *theAttr*)
ABC for classes that have the YY/MM/DD date field.

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.

read(*theFile*)
Read from a LIS physical file.

contFileName
Continuation file name.

class `TotalDepth.LIS.core.LogiRec.`**LrFileHead**(*theType*, *theAttr*)
Specific class of File Head.

prevFileName
Previous file name.

class `TotalDepth.LIS.core.LogiRec.`**LrFileHeadRead**(*theFile*)
Specific class of File head read from a file.

class `TotalDepth.LIS.core.LogiRec.`**`LrFileTail`**(*theType*, *theAttr*)
Specific class of File Tail.

`nextFileName`
Next file name.

class `TotalDepth.LIS.core.LogiRec.`**`LrFileTailRead`**(*theFile*)
Specific class of File tail read from a file.

class `TotalDepth.LIS.core.LogiRec.`**`LrReelTapeHeadTail`**(*theType*, *theAttr*)
Parent class of Reel/Tape Head/Tail that have identical structure.

class `TotalDepth.LIS.core.LogiRec.`**`LrTapeHeadTail`**(*theType*, *theAttr*)
Tape head or tail Logical Record.

class `TotalDepth.LIS.core.LogiRec.`**`LrTapeHead`**(*theType*, *theAttr*)
Tape head Logical Record.

`prevTapeName`
Previous tape name.

class `TotalDepth.LIS.core.LogiRec.`**`LrTapeHeadRead`**(*theFile*)
Specific class of Tape head read from a file.

class `TotalDepth.LIS.core.LogiRec.`**`LrTapeTail`**(*theType*, *theAttr*)
Tape tail Logical Record.

`nextTapeName`
Next tape name.

class `TotalDepth.LIS.core.LogiRec.`**`LrTapeTailRead`**(*theFile*)
Specific class of Tape tail read from a file.

class `TotalDepth.LIS.core.LogiRec.`**`LrReelHeadTail`**(*theType*, *theAttr*)
Reel head or tail Logical Record.

class `TotalDepth.LIS.core.LogiRec.`**`LrReelHead`**(*theType*, *theAttr*)
Reel head Logical Record.

`prevReelName`
Previous reel name.

class `TotalDepth.LIS.core.LogiRec.`**`LrReelHeadRead`**(*theFile*)
Specific class of Reel head read from a file.

class `TotalDepth.LIS.core.LogiRec.`**`LrReelTail`**(*theType*, *theAttr*)
Reel tail Logical Record.

`nextReelName`
Next reel name.

class `TotalDepth.LIS.core.LogiRec.`**`LrReelTailRead`**(*theFile*)
Specific class of Reel tail read from a file.

class `TotalDepth.LIS.core.LogiRec.`**`LrMisc`**(*theType*, *theAttr*)
Miscellaneous Logical Record.

class `TotalDepth.LIS.core.LogiRec.`**`LrMiscRead`**(*theFile*)
Miscellaneous Logical Record read from a LIS file.

`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

setValue(*v*)
Sets the value to v.

lisBytes()
Return a bytes() array from the internal representation.

value
The value of the Component Block or None.

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.

genCells()
yields each CbEngVal.

__len__()
The number of cells in the row.

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*)
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.

isSingleParam
True it this table is a list of single parameters (i.e. type 0 blocks).

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.

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.**EntryBlockRead**
An entry block read from a LIS file.

class TotalDepth.LIS.core.LogiRec.**EntryBlockSet**
Represents the set of Entry Blocks in a DFSR.

ATTR_MAP = {'dataType': 1, 'dsbType': 2, 'upDown': 4, 'optLogScale': 5, 'frameSpac
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.

__getattr__(*name*)
Returns the Entry Block corresponding to the name.

__getitem__(*key*)
This returns an Entry block by integer index.

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.

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.

xInc
True if the logging X increases (down or time log).

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.

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.

class TotalDepth.LIS.core.LogiRec.**LrDFSR**(*theType*, *theAttr*)
Data Format Specification Record.

class TotalDepth.LIS.core.LogiRec.**LrDFSRRead**(*theFile*)
Data Format Specification Record read from a file.

class TotalDepth.LIS.core.LogiRec.**LrNormalAlternateData**(*theType*, *theAttr*)
Class for Normal and Alternate data i.e. curve data.

class `TotalDepth.LIS.core.LogiRec.`**`LrNormalAlternateDataRead`**(*theFile*)
Class for Normal and Alternate data i.e. curve data.

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)

class `TotalDepth.LIS.core.LogiRec.`**`LrFactoryRead`**
A factory for generating complete Logical Records from a file.

`retLrFromFile`(*theFile*)
Given a LIS file this reads one Logical Record, and returns the appropriate Logical Record object or None.

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.`**`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.

`genAll`()
Generates each index object (child class of IndexObjBase).

`genLogPasses`()
Yields all IndexLogPass objects in the index.

`genPlotRecords`(*fromInternalRecords=True*)
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.

`jsonObject`()
Return an Python object that can be JSON encoded.

`longDesc`()
Returns a string that is the long description of this object.

`lrTypeS`
Returns a list of Logical Record types (integers) that is in the index.

`numLogPasses`()
Returns the number of IndexLogPass objects in the index.

class `TotalDepth.LIS.core.FileIndexer.`**`IndexFileHead`**(*tell*, *lrType*, *theF*)
Indexes a File header. The full Logical record is retained.

class TotalDepth.LIS.core.FileIndexer.**IndexFileHeadTail**(*tell*, *lrType*, *theF*, *theClass*)

Indexes a File header or trailer. The full Logical record is retained.

tocStr()

Returns a 'pretty' string suitable for a table of contents.

class TotalDepth.LIS.core.FileIndexer.**IndexFileTail**(*tell*, *lrType*, *theF*)

Indexes a File trailer. The full Logical record is retained.

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.

add(*tell*, *lrType*, *theF*)

Add an IFLR.

canAdd(*iflrType*)

Returns the IFLR type that this EFLR can describe.

iflrType()

Returns the IFLR type that this EFLR can describe.

jsonObject()

Return an Python object that can be JSON encoded.

logPass

The LogPass object.

tocStr()

Returns a 'pretty' string suitable for a table of contents.

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.

setLogicalRecord(*theFile*)

Sets the logicalRecord property to an LogiRec object from theFile.

class TotalDepth.LIS.core.FileIndexer.**IndexNone**(*tell*, *lrType*, *theF*)

NULL class just takes the LR information and skips to next LR.

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.

canAdd(*iflrType*)

Returns True if this can accumulate another IFLR.

iflrType()

Returns the IFLR type that this EFLR can describe.

isDelimiter

True if this represents a delimiter record e.g. File Head/Tail.

jsonObject()

Return an Python object that can be JSON encoded.

logicalRecord
The underlying LogiRec object or None.

lrType
The Logical Record type, in integer.

setLogicalRecord(*theFile*)
Sets the logicalRecord property to an LogiRec object from theFile.

tell
The file offset of the Logical Record.

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.

class TotalDepth.LIS.core.FileIndexer.**IndexReelHeadTail**(*tell*, *lrType*, *theF*, *theClass*)
Indexes a Reel header or trailer. The full Logical record is retained.

tocStr()
Returns a 'pretty' string suitable for a table of contents.

class TotalDepth.LIS.core.FileIndexer.**IndexReelTail**(*tell*, *lrType*, *theF*)
Indexes a Reel trailer. The full Logical record is retained.

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.

jsonObject()
Return an Python object that can be JSON encoded.

name
The name of the table from the first component block.

setLogicalRecord(*theFile*)
Sets the logicalRecord property to an LogiRec.LrTable() object.

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.

class TotalDepth.LIS.core.FileIndexer.**IndexTapeHeadTail**(*tell*, *lrType*, *theF*, *theClass*)
Indexes a Tape header or trailer. The full Logical record is retained.

tocStr()
Returns a 'pretty' string suitable for a table of contents.

class TotalDepth.LIS.core.FileIndexer.**IndexTapeTail**(*tell*, *lrType*, *theF*)
Indexes a Tape trailer. The full Logical record is retained.

class TotalDepth.LIS.core.FileIndexer.**IndexUnknownInternalFormat**(*tell*, *lrType*, *theF*)
Binary verbatim class for things like encrypted records, images, raw table dumps and so on.

setLogicalRecord(*theFile*)
Sets the logicalRecord property to an LogiRec.LrTable() object.

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.

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.

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.

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

The LIS Log Pass

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).

`TotalDepth.LIS.core.LogPass.`**`EVENT_EXTRAPOLATE = 'extrapolate'`**
Event to extrapolate X axis

`TotalDepth.LIS.core.LogPass.`**`EVENT_READ = 'read'`**
Event to read bytes from frame

`TotalDepth.LIS.core.LogPass.`**`EVENT_SEEK_LR = 'seekLr'`**
Event to seek to the start of a new Logical Record

`TotalDepth.LIS.core.LogPass.`**`EVENT_SKIP = 'skip'`**
Event to skip bytes in frame

`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.`**`ExceptionLogPassKeyError`**
Raised when and internal KeyError is raised in a FrameSet access.

`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.`**`ExceptionLogPassNoType01Data`**
Raised on access when there is no frame data loaded by addType01Data().

`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.

`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.

curveUnits(*chMnem*)
Given a curve as a Mnem.Mnem() this returns the units as a bytes object.

curveUnitsAsStr(*chMnem*)
Given a curve as a Mnem.Mnem() this returns the units as a string.

dfsr
The DFSR used for construction.

frameFromX(*theEv*)
Returns the estimated frame number from the X axis, and EngValue.

frameSet
The Frame Set as a FrameSet.FrameSet() object or None if not initialised.

frameSetLongStr()
Returns a long (multiline) descriptive string of the Frame Set or N/A if not initialised.

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.

genOutpPoints(*theMnem*)
Wrapper around the frameset generator, in fact this returns exactly that generator.

hasOutpMnem(*theMnem*)
Returns True is theMnem is in this LogPass (i.e. is in the DFSR).

iflrType
Returns the IFLR type that this LogPass describes.

isIndirectX
True if indirect X axis, False if explicit X axis channel.

jsonObject()
Return an Python object that can be JSON encoded.

longStr()
Returns a long (multiline) descriptive string.

nullValue
The NULL or absent value as specified in the DFSR.

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.

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 ExceptionLogPassKeyError.

rle
The Logical Record Run Length Encoding as a Rle.RLEType01() object.

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).

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.

totalFrames
The total number of frames.

type01Plan
The Frame plan, a Type01Plan.FrameSetPlan() object.

xAxisFirstEngVal
The EngVal (value, units) of the X axis of the first frame.

xAxisFirstVal
The numerical value of the X axis of the first frame.

xAxisFirstValOptical
The numerical value of the X axis of the first frame in 'optical' units.

xAxisIndex
The channel index that corresponds to the X axis.

xAxisLastEngVal
The EngVal (value, units) of the X axis of the first frame.

xAxisLastVal
The numerical value of the X axis of the last frame.

xAxisLastValOptical
The numerical value of the X axis of the last frame in 'optical' units.

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.

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.

xAxisUnits
The units of the X axis.

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".

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')
```

Resources

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.

Resources

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])
```

Resources

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 *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)
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%
```

The LIS Frame Set

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.

class `TotalDepth.LIS.core.FrameSet.`**`SuChArTe`**
Sub-channel Array Template.

`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.

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.

`numSubChannels`
Number of sub-channels.

`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.`**`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.

`longStr`()
Returns a long string that describes me.

`dumpFrames`(*theS=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)
Dump the frames to the stream.

`nbytes`
Returns the number of bytes in the underlying array implementation.

`lisSize`
The number of LIS bytes that make up this FrameSet.

`numFrames`
The number of frames currently in this FrameSet.

`valuesPerFrame`
The number of values in each frame currently in this FrameSet.

`numValues`
The total number of values in the array.

`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.

`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.

`isIndirectX`
True if there is an indirect X axis, False otherwise.

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.

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.

class `TotalDepth.LIS.core.FrameSet.`**AccMin**

Accumulates the minimum value.

add(*v*)

Add a new value.

value()

Return the result.

class `TotalDepth.LIS.core.FrameSet.`**AccMax**

Accumulates the maximum value.

add(*v*)

Add a new value.

value()

Return the result.

class `TotalDepth.LIS.core.FrameSet.`**AccMean**

Accumulates the mean value.

add(*v*)

Add a new value.

value()

Return the result.

class `TotalDepth.LIS.core.FrameSet.`**AccStDev**

Accumulates the standard deviation.

add(*v*)

Add a new value.

value()

Return the result.

class `TotalDepth.LIS.core.FrameSet.`**AccCount**

Accumulates the number of values.

add(*v*)

Add a new value.

value()

Return the result.

class `TotalDepth.LIS.core.FrameSet.`**AccDelta**

Base class for acumulating a count of first order differences.

add(*v*)

Add a new value.

value()

Return the result.

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.

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.

add(*v*)
Add a new value.

value()
Return the result.

class TotalDepth.LIS.core.FrameSet.**AccActivity**
Measures curve activity.

add(*v*)
Add a new value.

value()
Return the result.

FrameSet Usage

Run Length Encoding Module

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.`**`RLEItem`**(*v*)

Class that represents a single entry in a Run Length Encoding set. v - The datum value.

`__str__`()

String representation.

`datum`

The initial datum value.

`stride`

The stride as a number or None if there is only one entry.

`repeat`

The repeat count.

`numValues`()

Total number of record values.

`add`(*v*)

Returns True if v has been absorbed in this entry. False means a new entry is required.

`values`()

Generates all values.

`value`(*i*)

Returns a particular value.

`range`()

Returns a range object that has (start, stop, step) or None if a single entry.

`first`()

Returns the first value.

`last`()

Returns the last value.

`__weakref__`

list of weak references to the object (if defined)

class `TotalDepth.LIS.core.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__`()

String representation.

`__len__`()

The number of RLEItem(s).

`__getitem__`(*key*)

Returns a RLEItem.

numValues()
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.

rangeList()
Returns a list of range() or None objects. A None object means that the RLEItem has a single value.

first()
Returns the first value or None if no values added.

last()
Returns the last value or None if no values added.

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

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.

__str__()
String representation.

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.

__str__()
String representation.

xAxisUnits
X axis units.

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).

Type01Plan

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.

indirectSize
The size, in bytes of the indirect X axis value. 0 for explicit X axis.

frameSize
Frame size in bytes.

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.

1.11.2 TotalDepth.LAS API Reference

Contents:

LAS Read

Reads LAS files.

Created on Jan 11, 2012

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.`**`hasLASExtension`**(*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

`TotalDepth.LAS.core.LASRead.`**`DEBUG_LINE_BY_LINE = False`**
logging.debug call here can add about 50% of the processing time

`TotalDepth.LAS.core.LASRead.`**`genLines`**(*f*)
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 heasd

`TotalDepth.LAS.core.LASRead.`**`SECT_DESCRIPTION_MAP = {'V': 'Version Information Section', 'W`**
: 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

`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.`**`RE_LINE_FIELD_2 = re.compile("\n ^ # Start of line\n ([^{|]+)*`**
: Data field 2 is after the last ':' THREE groups

class `TotalDepth.LAS.core.LASRead.`**`SectLine`**(*mnem*, *unit*, *valu*, *desc*, *format*, *assoc*)
: Captures the 6 fields: mnem unit valu desc format assoc

  `__getnewargs__`()
  : Return self as a plain tuple. Used by copy and pickle.

  static `__new__`(*_cls*, *mnem*, *unit*, *valu*, *desc*, *format*, *assoc*)
  : Create new instance of SectLine(mnem, unit, valu, desc, format, assoc)

  `__repr__`()
  : Return a nicely formatted representation string

  `assoc`
  : Alias for field number 5

  `desc`
  : Alias for field number 3

  `format`
  : Alias for field number 4

  `mnem`
  : Alias for field number 0

  `unit`
  : Alias for field number 1

  `valu`
  : Alias for field number 2

class `TotalDepth.LAS.core.LASRead.`**`LASSection`**(*sectType*)
: Contains data on a section.

  `__len__`()
  : Number of members.

  `__contains__`(*theMnem*)
  : Membership test.

  `addMemberLine`(*i*, *l*)
  : Given a line this decomposes it to its _members. i is the position of the line l in the file.

  `finalise`()
  : Finalisation of section, this updates the internal representation.

  `__getitem__`(*key*)
  : Returns an entry, key can be int or str.

mnemS()
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**(*sectType*, *wrap*, *curvSect*, *null=-999.25*)
Contains data on an array section.

addMemberLine(*i*, *l*)
Process a line in an array section.

finalise()
Finalisation. TODO: Make a numpy array and get rid of the members. Need to overload __getitem__.

frameSize()
Returns the number of data points in a frame.

class TotalDepth.LAS.core.LASRead.**LASBase**(*id*)
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

__len__()
Number of sections.

__getitem__(*key*)
Returns a section, key can be int or str.

numFrames()
Returns the number of frames of data in an 'A' record if I have one.

numDataPoints()
Returns the number of frame data points in an 'A' record if I have one.

genSects()
Yields up each section.

nullValue
The NULL value, defaults to -999.25.

xAxisUnits
The X axis units.

xAxisStart
The Xaxis start value as an EngVal.

xAxisStop
The Xaxis end value as an EngVal.

xAxisStep
The Xaxis step value as an EngVal.

logDown()
Returns True if X axis is increasing i.e. for time or down log.

getWsdMnem(*m*)
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.

getAllWsdMnemonics()
Returns a set of mnemonics from the Well section and the Parameter section.

hasOutpMnem(*theMnem*)
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.

curveMnems(*ordered=False*)
Returns list of curve names actually declared in the Curve section. List will be unordered if ordered is False.

curveUnitsAsStr(*m*)
Given a curve as a Mnem.Mnem() this returns the units as a string. May raise KeyError.

genOutpPoints(*theMnem*)
Generates curve values for theMnem, a Mnem.Mnem() object.

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

class `TotalDepth.LAS.core.LASRead.`**LASRead**(*theFp*, *theFileID=None*)
Reads a LAS file.

__init__(*theFp*, *theFileID=None*)
Reads a LAS file from theFp that is either a string (file path) or a file like object.

1.11.3 Utility Package Reference

Contents:

Common Command Line Options

Common command line options, this is to try and present some degree of interface consistency among command line applications.

Copyright (c) 2010-2011 Paul Ross. All rights reserved.

`TotalDepth.util.CmnCmdOpts.`**argParser**(*desc*, *prog=None*, *version=None*)
Return an command line parser with the standard pre-set options.

Standard options are `-h, --version` and:

`-j`: Multiprocessing job control.

`-k`: Flag to indicate that we should keep going as far as sensible.

`-l`: Log level.

`TotalDepth.util.CmnCmdOpts.`**argParserIn**(**args*, ***kwargs*)
Return an command line parser with the standard pre-set options plus an input path as an argument.

`TotalDepth.util.CmnCmdOpts.`**argParserInOut**(**args*, ***kwargs*)
Return an command line parser with the standard pre-set options plus an input and output paths as an arguments.

DictTree

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.

class `TotalDepth.util.DictTree.`**`DictTree`**(*valIterable=None*)
A dictionary that takes a list of hashables as a key and behaves like a tree

`add`(*k*, *v*)
Add a key/value. k is a list of hashables.

`remove`(*k*, *v=None*)
Remove a key/value. k is a list of hashables.

`value`(*k*)
Value corresponding to a key or None. k is a list of hashables.

`values`()
Returns a list of all values.

`keys`()
Return a list of keys where each key is a list of hashables.

`__len__`()
Returns the number of keys.

`depth`()
Returns the maximum tree depth as an integer.

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

class `TotalDepth.util.DictTree.`**`DictTreeHtmlTable`**(**args*)
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%">
for anEvent in myTree.genColRowEvents():
    if anEvent == myTree.ROW_OPEN:
        # Write out the '<tr>' element
    elif anEvent == myTree.ROW_CLOSE:
        # Write out the '</tr>' element
    else:
        k, v, r, c = anEvent
        # Write '<td rowspan="%d" colspan="%d">%s</td>' % (r, c, v)
# 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>
```

setColRowSpan()

Top level call that sets colspan and rowspan attributes.

genColRowEvents()
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

Directory Walking

Provides various ways of walking a directory tree

Created on Jun 9, 2011

class TotalDepth.util.DirWalk.**FileInOut**(*filePathIn*, *filePathOut*)
A pair of (in, out) file paths

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__(*_cls*, *filePathIn*, *filePathOut*)
Create new instance of FileInOut(filePathIn, filePathOut)

__repr__()
Return a nicely formatted representation string

filePathIn
Alias for field number 0

filePathOut
Alias for field number 1

exception TotalDepth.util.DirWalk.**ExceptionDirWalk**
Exception class for this module.

TotalDepth.util.DirWalk.**genBigFirst**(*d*)
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*, *theOut=None*, *theFnMatch=None*, *recursive=False*, *bigFirst=False*)
Walks a directory tree generating file paths.

theIn - The input directory.

theOut - The output directory. If None then input file paths as strings will be generated If non-None this function will yield FileInOut(in, out) objects. 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.

Timing Code Execution

Has classes for timing execution

exception TotalDepth.util.ExecTimer.**ExceptionExecTimer**
Specialisation of exception for this module.

class `TotalDepth.util.ExecTimer.`**`ExecEvent`**(*desc*)

Records the timing of a single event.

`stop`(*bytes=0*)

Stop the timer.

`hasCompleted`

True if the timer has been stopped.

`tExec`

Executions time in seconds.

`lenDesc`()

Length of the task description string.

`writeToStderr`(*descWidth=0*)

Write self to sys.stderr.

`writeToStream`(*theS*, *descWidth=0*)

Write self to provided stream.

`__weakref__`

list of weak references to the object (if defined)

class `TotalDepth.util.ExecTimer.`**`ExecTimerList`**

Maintains a list of execution time objects

`__init__`()

Constructor

`__len__`()

Number of task timers.

`startNewTimer`(*theDesc*)

Load a new task timer starting right now.

`timer`

The current timer.

`hasActiveTimer`

True if there is a running timer, False if there are either no timers or the latest timer is halted.

`stopTimer`(*bytes=0*)

Stop current timer.

`writeToStderr`()

Write self to sys.stderr.

`writeToStream`(*theS*)

Write self to provided stream.

`__weakref__`

list of weak references to the object (if defined)

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.

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)

Histogram

Produces histograms.

Created on Nov 29, 2011

class `TotalDepth.util.Histogram.`**Histogram**
A histogram class.

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 tha tacture 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 | ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++"""
```

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.

XML Writer Module

Introduction

The important classes here are:

- `XmlWrite.XMLStream` (and derived classes): these manage the low level stream output.
- `XmlWrite.XMLElement`: these handles opening and closing elements via the Context Manager. They require an initialised XMLStream in their constructor.

Together, if used correctly, these make writing output to a stream fluent in the code. They also make strong guarantees about the encoding and well-formedness of the result.

Reference

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.

`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.**Element**(*theXmlStream*, *theElemName*, *theAttrs=None*)
Represents an element in a markup stream.

__enter__()
Context manager support.

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

__exit__(*excType*, *excValue*, *tb*)
Context manager support.

Examples

All these examples assume these imports:

```
import io
from TotalDepth.util import XmlWrite
```

Creating an XML Stream

Create a file-like object and write to it using an XmlWrite.XmlStream:

```
myF = io.StringIO()
with XmlWrite.XmlStream(myF):
    pass
print(myF.getvalue())
# Results in:
<?xml version='1.0' encoding="utf-8"?>\n
```

Simple Elements

Create a file-like object and write elements to it using an `XmlWrite.XmlElement`:

```
myF = io.StringIO()
with XmlWrite.XmlStream(myF) as xS:
    with XmlWrite.Element(xS, 'Root', {'version' : '12.0'}):
        with XmlWrite.Element(xS, 'A', {'attr_1' : '1'}):
            pass
print(myF.getvalue())
# Results in:
<?xml version='1.0' encoding="utf-8"?>
<Root version="12.0">
    <A attr_1="1"/>
</Root>
```

Mixed Content

As above but writing text as well:

```
myF = io.StringIO()
with XmlWrite.XmlStream(myF) as xS:
    with XmlWrite.Element(xS, 'Root', {'version' : '12.0'}):
        with XmlWrite.Element(xS, 'A', {'attr_1' : '1'}):
            xS.characters('<&>')
print(myF.getvalue())
# Results in:
<?xml version='1.0' encoding="utf-8"?>
<Root version="12.0">
    <A attr_1="1">&lt;&amp;&gt;</A>
</Root>
```

Testing

The unit tests are in `test/TestWMLWrite.py`.

Plot Package Reference

Contents:

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.`**`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.

`__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.

`__lt__``(`*other*`)`
: Returns true if self value < other value after unit conversion.

`__le__``(`*other*`)`
: Returns true if self value <= other value after unit conversion.

`__eq__`(*other*)
Returns true if self value == other value after unit conversion.

`__ne__`(*other*)
Returns true if self value != other value after unit conversion.

`__gt__`(*other*)
Returns true if self value > other value after unit conversion.

`__ge__`(*other*)
Returns true if self value >= other value after unit conversion.

`TotalDepth.util.plot.Coord.`**`dimIn`**(*v*)
Returns a Dim object with the value in inches.

class `TotalDepth.util.plot.Coord.`**`Box`**
A named tuple that describes a box with width and depth as Dim() objects.

`__str__`()
Stringifying.

class `TotalDepth.util.plot.Coord.`**`Pad`**
Padding around another object that forms the Bounding Box. All 4 attributes are Dim() objects

`__str__`()
Stringifying.

class `TotalDepth.util.plot.Coord.`**`Margin`**
Margin padding around another object. All 4 attributes are Coord.Dim() objects.

`__str__`()
Stringifying.

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__`()
Stringifying.

`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.

`TotalDepth.util.plot.Coord.`**`baseUnitsDim`**(*theLen*)
Returns a Coord.Dim() of length and units BASE_UNITS.

`TotalDepth.util.plot.Coord.`**`zeroBaseUnitsDim`**()
Returns a Coord.Dim() of zero length and units BASE_UNITS.

`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.

`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.

`TotalDepth.util.plot.Coord.`**`convertPt`**(*theP*, *theUnits*)
    Returns a new point with the dimensions of theP converted to theUnits.

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`.

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

`coding`
Alias for field number 2

`colour`
Alias for field number 1

`opacity`
Alias for field number 3

`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.

SVG Writer Module

An SVG writer.

TODO: Add a float format to reduce the size of the SVG file.

`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

exception `TotalDepth.util.plot.SVGWriter.`**`ExceptionSVGWriter`**
Exception class for SVGWriter.

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.`**`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.`**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.`**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.`**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.`**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.`**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.

`TotalDepth.util.plot.SVGWriter.`**dimToTxt**(*theDim*)
Converts a Coord.Dim() object to text for SVG units.

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())
# 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>
```

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(value=0.25, units='in')`**
    Definition of a quarter inch margin

`TotalDepth.util.plot.PlotConstants.`**`STANDARD_PAPER_WIDTH = Dim(value=8.5, units='in')`**
    Standard wireline log fan paper width i.e. long side.

`TotalDepth.util.plot.PlotConstants.`**`STANDARD_PAPER_DEPTH = Dim(value=6.25, units='in')`**
    Standard wireline log fan paper depth i.e. short side.

`TotalDepth.util.plot.PlotConstants.`**`STANDARD_PAPER_MARGIN = Margin(left=Dim(value=0.25, uni`**
    Standard wireline log fan paper margin.

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
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

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
-------------------------------------------

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 = functools.partial(<function genLog10>, cycles=`**
Generator for 2 decades of log base 10

`TotalDepth.util.plot.Track.`**`genLog10Decade3 = functools.partial(<function genLog10>, cycles=`**
Generator for 3 decades of log base 10

`TotalDepth.util.plot.Track.`**`genLog10Decade4 = functools.partial(<function genLog10>, cycles=`**
Generator for 4 decades of log base 10

`TotalDepth.util.plot.Track.`**`genLog10Decade5 = functools.partial(<function genLog10>, cycles=`**
Generator for 5 decades of log base 10

`TotalDepth.util.plot.Track.`**`genLog10Decade1Start2 = functools.partial(<function genLog10>, c`**
Generator for 1 decade of log base 10 starting at 2

`TotalDepth.util.plot.Track.`**`genLog10Decade2Start2 = functools.partial(<function genLog10>, c`**
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(value=0.25, units='in')`
Space for plotting the scale for each curve

`left`
The left edge as a Coord.Dim().

`right`
The left edge as a Coord.Dim().

`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.

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)

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.

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.

AREA Plotting Configuration

Module for plotting areas of particular patterns wirth well log data.

Created on 1 Apr 2011

Note on patterns in XML file in: `formats\IWWPatterns.xml`

There are elements thus:

```
<Bits>/+7/7v/u/+7/7gAA7/7v/u/+7/7v/gAA</Bits>
<PatternHeight>12</PatternHeight>
<PatternWidth>15</PatternWidth>
```

Frequency analysis shows that the characters used are '+', '/', A-Z, a-z, 0-9 so this looks like base64 encoded. Length is always 32 chars (256 bits). Pattern size suggests 12*15=180 bits.

Examples:

```
>>> base64.b64decode(b'/+7/7v/u/+7/7gAA7/7v/u/+7/7v/gAA')
b'ÿîÿîÿîÿî\\ïþïþïþïþ\\'

<Description>Coal-LigNite</Description>
>>> base64.b64decode(b'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA')
b'\\\\\\\\\\\\\\\\\\\\\\\\'
```

```
<Description>Void</Description>
>>> base64.b64decode(b'//7//v/+//7//v/+//7//v/+//7//v/+')
b'ÿþÿþÿþÿþÿþÿþÿþÿþÿþÿþÿþÿþ'

>>> base64.b64decode(b'2HzfkrnudnzvstO2n67gmZ83Z7V5iYa')
b'Ø|ß¹îv|ï²ÝÎÚ~»f|ÝÕæ&'
```

Length is always 24 bytes (192 bits) 12x16 ? This looks promising:

```
>>> l = list(b'Ø|ß¹îv|ï²ÝÎÚ~»f|ÝÕæ&')
>>> s = ['{:08b}'.format(v) for v in l]
>>> st = ''.join(s)
>>> patS = [st[i:i+12] for i in range(16)]
>>> print('\n'.join(patS))
110110000111
101100001111
011000011111
110000111110
100001111100
000011111001
000111110011
001111100110
011111001101
111110011011
111100110111
111001101111
110011011111
100110111111
001101111110
011011111100
```

Or shortened:

```
st = ''.join(['{:08b}'.format(v) for v in b'...'])
print('\n'.join([st[i:i+12] for i in range(16)]))

<Description>Void</Description>
>>> base64.b64decode(b'//7//v/+//7//v/+//7//v/+//7//v/+')
b'ÿþÿþÿþÿþÿþÿþÿþÿþÿþÿþÿþÿþ'
111111111111
111111111111
111111111111
111111111111
111111111110
111111111101
111111111011
111111110111
111111101111
111111011111
111110111111
111101111111
111011111111
110111111111
101111111111
011111111111
```

Hmmm 12 zeros, perhaps it is 12x15 after all:

```
>>> base64.b64decode(b'/+7/7v/u/+7/7gAA7/7v/u/+7/7v/gAA')
b'ÿîÿîÿîÿîÿî\\ïþïþïþïþïþ\\'
111111111110
111111111101
111111111011
111111110111
111111101110
111111011101
111110111011
111101110111
111011101111
110111011111
101110111111
011101111111
111011111111
110111111111
101111111111
011111111111
```

Full range (sorted):

```
//4zAP/6//oA9v/+//4zAP/8//oA+v/2
//6/AIP+//4Avv+C//6/AIP+//4Avv+C
//60ks/+//6ZaP+e//60ks/+//6ZaP+e
//69fIEA//4QENfW//69fIEA//4QENf2
//6N5r/mj76/urm+ub7v8u7+774t8v/y
//6v1vt+rOr//lq2/87//rrq//7XXu++
//7//v/+//7//v/+//7//v/+//7//v/+
//7/3t/e397fwsP+//7/3t/e397fwsP+
//7/AP/+//4A/v/+//7/AP/+//4A/v/+
//7/AP/+/9YA7v/W1/7vANf+/9YA7v/W
//7/AP/+Wv4A/v/+//7/AP/+Wv4A/v/+
//7/APv+9/4A/v/6//b/AP3+8/4A/v/+
//73/uv+wf7//v/+//7/3v+u/wb//v/+
//732gAAvfbvfAAAu97vegAAve73fAAA
//7bAP/+//4A1v/+//7bAP/+//4A1v/+
//7bAPv+9/4A1v/6//bbAP3+8/4A2v/+
//7HHoLugu7HHv/+//6PxneCd4KPxv/+
//7HHrruuu7HHv/+//6Pxne6d7qPxv/+
//7nAP/+/+4A7v/u//7nAP/+/+4A7v/u
//7nOOc4//45zjnO//7nOOc4//45zjnO
//7nOOc4//45zv/O//7nOP8+//45zvn+
//7vAP/+//4A7v/+//7vAP/+//4A7v/+
//b/9v/2+7b5Ngqg+777vv/+9/73/gAA
//b/9v/2+7b7tgqg+T77vv/+9/73/gAA
//b/9v/2+Db79gog+774Pv/+9/73/gAA
//b/9v/2+Hb7tghg+774fv/+9/73/gAA
//b/9v/2+Pb7dgjA+/77/v/+9/73/gAA
/+4zLv/uzOj/7gcA3/7TMt/+XMzf/gBy
/+6B7n7uge7/7gAA7/7vAu787wLv/gAA
/+7/7rvux+7/7gAA7/7v/u+678bv/gAA
/+7/7sHu/+7/7gAA797vvu/+7/7v/gAA
/+7/7uDu/+7/7gAA7/7v/u8G7/7v/gAA
/+7/7v/u/+7/7gAA7/7v/u/+7/7v/gAA
/+7/7v/u/+7/7oOC7/7v/u/+7/7v/oOC
/+7/7v/u/+7bbAAA7/7v/u/+7/5ttgAA
/+7/7v/u/+r/5gAA7/7v/u/+6/7n/gAA
/+7/7vPu/+7/7gAA7/7v/u8+7/7v/gAA
```

```
/+7/7vPu/+7/7gAA7/7v/u8G7/7v/gAA
/+7/7vPu/+7/7gAA797vvu/+7/7v/gAA
/+7/AP/+7/4A/v/+/+7/AP/+7/4A/v/+
/+717vvu9e7/7gAA7/7v1u/u79bv/gAA
/+73xuPG44LB/v/+/+73xuPG44LB/v/+
/+7bAP/+7/4A1v/+/+7bAP/+7/4A1v/+
/+7n7oHu5+7/7gAA7/7vzu8C787v/gAA
/+7v7sfu7+7/7gAA7/7v7u/G7+7v/gAA
/+7v7tfu7+7/7gAA7/7v7u/W7+7v/gAA
/+7vAP/+7/4A7v/+/+7vAP/+7/4A7v/+
/+7X7oPu1+7/7gAA7/7v1u+C79bv/gAA
/+bP2ofah+bP/v8+/h7+Hs8+t/63/s/+
/+bP2rfat+bP/v8+/t7+3s8+t/63/s/+
/34a4Pn+AAD/+sHW/84AAO/+Xg4//gAA
/34ekP3+AAD/+gc2/+4AAO/+3MC//gAA
/34O4P3+AAD/+uD2/+4AAO/+3wa//gAA
/36a8vn+AAD/+ufW/84AAO/+Xz4//gAA
/37+/v3+AAD/+v/2/+4AAO/+3/6//gAA
/376/vn+AAD/+v/W/84AAO/+X/4//gAA
/37Ozv3+AAD/+s/2/+4AAO/+356//gAA
/3b/jv+u/47/dv/+u/7H/tf+x/67/v/+
/97/wP/e+N77Xvje+/77/v/g//4AHv/+
/d773vvuAAC+/r7+f34AAPfu7/bv9gAA
/wKBer0Cgf7//v/+///7/AoF6vQKB/v/+
+/73MgH+//Yz7v4C+/73MgP+//Yz7v4C
2/7nMu/+3/75mP/+/7Yzzv/e/76Z8v/+
2Hzfkrnudnzvst3O2n67gmZ83Z7V5iYa
3/4HMv/+/+4zgv/+3/4HMv/+/+4zgv/+
3/6s+N/+/+451v/u3/6s+N/+/+451v/u
3d7u7vd2e7q93N7u73b3unvcve7e9u96
3d7urvZ2ebq53Nbqz3a3inucvW7e9u16
3dz//v/+d3b//v/+3dz//v/+d3b//v/+
4er//tde//764P/+117//uHq//7XXv/+
4er//tdO/+764P/+117f/sHq//7XXv/+
5/6BAOf+/+aAgP/m5/6BAOf+/+aAgP/m
5+aB/uf+/87/Ao/OvH6lfrT+hX7/5v+A
5mb//szm//6ZmP/+//4zOP/+//45zv/+
5zL//pnO//7nMv/+mc7//ucy//45zv/+
7/6rMu/+AADu7qqq7u4AAP7uMqr+7gAA
7/7vAO/+/+4A7v/u7/7vAO/+/+4A7v/u
8e7u7u7u7u7x7gAA7x7u7u7u7u7vHgAA
9/7nANf+//YA5v/W9/7nANf+//YA5v/W
9/7nMtf+/+4zzv+u9/7nMtf+/+4zzv+u
9/7vAN/+//YA7v/e9/7vAN/+//YA7v/e
9fT//l9e//719P/+X17//vX0//5fXv/+
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAD//gAA//4AAP/+AAD//gAA//4AAP/+
ff67/td877r/1v/uff67/td877r/1v/u
JJK22pJIJJK22pJIJJK22pJIJJK22pJI
m/73Mu/+//bM7v/eO/73Mu/+//aZ7v/e
MzL//szO//6DwP/+zM7//v/+ODj//s/O
u/7WeO/+/7o81v/uu/7WeO/+/7r/1jzu
u/7WeO/+/7o81v/uu/7WeO/+/7r/1rvu
u/7XAO/+/7oA1v/uu/7XAO/+/7oA1v/u
uur//tdc//666v/+11z//rrq//7XXP/+
w+6Z7mbuWu7/7gAA74bvMu7M7rTv/gAA
x+677qvuu+7H7gAA78bvuu+q77rvxgAA
z/4DMs/+/84zAv/Oz/4DMs/+/84zAv/O
```

```
zM7MzszOzM7MzszOzM7MzszOzM7MzszO
zv4/PvnOx/I+/P9+/b7z3s/uvfb+fvu+
```

`TotalDepth.util.plot.AREACfg.`**`PATTERN_MAP = {'Void':  b'//7//v/+//7//v/+//7//v/+//7//v/+',`**

A map of {pattern name : bytes, . . . }

`TotalDepth.util.plot.AREACfg.`**`pprintPattern`**`(`*theName*`)`

Pretty print the pattern.

`TotalDepth.util.plot.AREACfg.`**`pprintAll`**`()`

Pretty print all patterns.

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

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
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
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'GDEC', b'----')": 227,
"(34, b'FILM', b'GDEC', b'-1- ')": 2,
"(34, b'FILM', b'GDEC', b'-2- ')": 6,
"(34, b'FILM', b'GDEC', b'-2--')": 32,
"(34, b'FILM', b'GDEC', b'-3- ')": 3,
"(34, b'FILM', b'GDEC', b'-4- ')": 10,
"(34, b'FILM', b'GDEC', b'-4--')": 26,
"(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.

`name`
: Name of the FILM.

`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'D40 ': 40, b'D200': 200, b'D240 ': 240, 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.

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.

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 = {'track12': 4, 'track23': 4, 'TrackFC1': 1, 'T`
Map of track IDs to start track location in half-tracks

`TRAC_UNIQUEID_TO_NON_STANDARD_START = {'track12': 2, 'track23': 4, 'TrackFC1': 4, '`
Number of half-tracks of the start of the track, this is a fudge around deciding this from the order of the LgTrack elements

`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.

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
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,
 "(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,
 "(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
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.

leftL
The left value of the curve scale as a number.

rightL
The right value of the curve scale as a number.

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

halfTrackStart
Alias for field number 2

halfTracks
Alias for field number 3

leftP
Alias for field number 0

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.`**PresCfg**
Contains the configuration equivalent to a complete PRES table.

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.

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

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.

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
```

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', co**

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 ', 'DepthTrack': b'TD ', 'track1':

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>
</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.

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()

d
Alias for field number 3

font
Alias for field number 5

mnem
Alias for field number 9

rAttr
Alias for field number 8

size
Alias for field number 6

tAttr
Alias for field number 7

text
Alias for field number 4

w
Alias for field number 2

x
Alias for field number 0

xMnem
Alias for field number 10

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 = {'text-anchor': 'start', 'font-weight':`**
SVG attributes for a bold text

`TotalDepth.util.plot.LogHeader.`**`TEXT_ATTRS_LARGE_WOB = {'text-anchor': 'start', 'font-weig`**
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.

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.

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.

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', 'TLI': 'STOP', 'CN': 'COMP', 'WN': 'WELL', 'FN'
Some conversions from LIS standard to LAS standard All as truncated ascii strings i.e. using pStr(strip=True)

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.

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
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

`curveName`
Alias for field number 1

`slice`
Alias for field number 0

`span`
Alias for field number 3

`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(value=0, units='in')*, *theTailDepth=Dim(value=0, units='in')*, *plotUp=True*, *theWidth=Dim(value=8.5, units='in')*, *theMargin=Margin(left=Dim(value=0.25, units='in')*, *right=Dim(value=0.25*, *units='in')*, *top=Dim(value=0.25*, *units='in')*, *bottom=Dim(value=0.25*, *units='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) -- ^^^^^^^^^^^^^^^^^^^.   |
|   .^                                                          ^.   |
|   .^                                                          ^.   |
|   .^                                                          ^.   |
|   .^                  Main log plotted here                   ^.   |
|   .^                                                          ^.   |
|   .^                                                          ^.   |
|   .^                                                          ^.   |
|   .^^^^^^^^^^^^^^^^ -- X Start (if up log) -- ^^^^^^^^^^^^^^^^^.   |
|   .++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++.   |
|   .+                                                          +.   |
|   .+       Scales (legends) at the bottom of the plot.        +.   |
|    .+                                                          +.    |
|    .++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++.    |
|    .************************************************************.    |
|    .*                                                          *.    |
|    .*         Optional trailer e.g. calibration record         *.    |
|    .*                                                          *.    |
|    .************************************************************.    |
|    ..............................................................    |
|                             Plot margin                               |
|----------------------------------------------------------------------|
```

`__init__`(*theXStart*, *theXStop*, *theScale*, *theLegendDepth*, *theHeadDepth=Dim(value=0*, *units='in')*, *theTailDepth=Dim(value=0*, *units='in')*, *plotUp=True*, *theWidth=Dim(value=8.5*, *units='in')*, *theMargin=Margin(left=Dim(value=0.25*, *units='in')*, *right=Dim(value=0.25*, *units='in')*, *top=Dim(value=0.25*, *units='in')*, *bottom=Dim(value=0.25*, *units='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.

`viewBox`
The overall size of the plot.

`width`
The overall width as a number in PlotConstants.DEFAULT_PLOT_UNITS.

`depth`
The overall width as a number in PlotConstants.DEFAULT_PLOT_UNITS.

`widthDim`
The overall width as a Coord.Dim().

`depthDim`
The overall width as a Coord.Dim().

`trackTopLeft`
"A Coord.Pt() that is the top left of the pane that tracks are plotted within.

`mainPanePlotDepth`
The depth of the plot of the main pane as a Coord.Dim().

`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(value=0.5, units='in')
: How much depth to give each legend (curve scale) at the top and bottom of the log

LEGEND_DEPTH_SPARE = Dim(value=0.5, units='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(value=0.08, units='in')
: Arrow head width on legend scales

LEGEND_ARROW_DEPTH = Dim(value=0.04, units='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(value=0.04, units='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.

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.

class `TotalDepth.util.plot.Plot.`**`PlotReadXML`**(*uniqueId*, *theScale=0*)
A subclass of Plot that is configured from XML file(s) using LgFormat.

1.12 A Bag Full of TODO's

This is just a gathering area for work items that should be done at some point.

TotalDepth is currently at **Alpha**, release version 0.2.0.

1.12.1 Current Release

This lists known work (and area) that are known lacuna in the Alpha release.

LIS Format Support

General

- XNAM direct support for LIS-A

LIS Index

- Various forms of persistence: XML, Pickle, JSON (no).
- Insert or append binary representation of the index to the LIS file.

LIS FrameSet/LogPass

- Provide a numpy view on each sub-channel.
- X Axis quality i.e. duplicate, missing frames.
- Change up log to down log (needs to rearrange multi-sampled channels).
- Clearer view on what 'frame spacing' actually is.

Representation Codes

- Check overflow/underflow on write.

LAS Format Support

- Read directly from .zip files.
- Version 3.0 support.
- Merge `~O` section int `~P` if correct format.
- Consistency checking of mutual data such as STRT/STOP/STEP

Plotting

- There is quite a lot of technical debt built up since we added LgFormat support, this area needs a review.
- Header: Some mud parameters being dropped.
- Benchmarks to characterise execution time and profiling.

1.12.2 Future Releases

This lists known work (and area) that are would be nice to have in future releases.

Format Support

Wireline Formats

- RP66v1
- RP66v2
- WellLogML?
- ATLAS BIT?

Seismic Formats

- SEG-Y
- SEG-D?
- Other SEG formats

Miscellaneous Formats

- Position

1.13 Glossary

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 however this rarely happens and it is debatable whether any or all LIS processing implementations support 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 *RP66*.

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.

Frame An array of values for each channel at a particular depth (or time).

Frame Set A set of frames representing multi-channel data that is typically depth or time series based.

IFLR See *Indirectly Formatted Logical Record*.

Indirectly Formatted Logical Record This is a Logical Record whose format is only completely 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".

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 continuos body of logging data such as "Repeat Section" or "Main Log". In the *LIS* format this is defined by a single Logical Record, the *DFSR*, plus multiple type 0 or type 1 Logical Records that the DFSR describes.

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 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*.

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 *RP66* 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 TotalDepth Licence

1.14.1 OpenSource Licence

TotalDepth is released under GPL 2.:

1.14.2 Other Licences

For other licences contact: apaulross@gmail.com

1.15 Credits

1.15.1 Development Lead

- Paul Ross <apaulross@gmail.com>

1.15.2 Contributors

None yet. Why not be the first?

1.16 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:

1.16.1 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 :)

1.16.2 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.

1.16.3 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.

1.16.4 Tips

To run a subset of tests:

```
$ pytest tests.test_TotalDepth
```

1.17 History

1.17.1 0.2.0 (2017-09-25)

- Moved to Github: https://github.com/paulross/TotalDepth
- First release on PyPI.

1.17.2 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

1.18 TotalDepth

Petrophysical software capable of processing wireline logs.

- Free software: GPL 2.0

- Documentation: https://TotalDepth.readthedocs.io

1.18.1 Features

- Reads LIS, LAS (1.2, 2.0) file formats for analysis or conversion to other formats.
- Plots log data as SVG viewable in any modern browser.
- Plots can be made with a wide variety of plot formats.
- TotalDepth can generate HTML summaries of log data.
- TotalDepth is written in Python so it is fast to develop with.
- Special indexing techniques are used to be able to randomly access sequential files.

Here is an example of a LAS file plotted with the Tripple Combo plot format as seen in a browser, this includes the API header:

images/TrippleCombo.png

An example of a High Resolution Dipmeter plotted at 1:25 scale:

images/HDT_25_no_hdr.png

1.18.2 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

For many year this project was hosted by Sourceforge. Thank you Sourceforge!

CHAPTER TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX