Updates to readlas

Kevin W. Hall and Gary F. Margrave

ABSTRACT

The CREWES Matlab toolbox contains a script called *readlas* that has long had the ability to read version 1.2 and version 2.0 Log ASCII Standard (LAS) files. A new Matlab class called *las* that can handle all LAS versions has been written. The constructor reads an entire LAS file into memory and splits it into a Matlab cell array using regular expressions. The *readlas* script is now a wrapper that makes a new *las* object and creates version 2.0 inputs suitable for *logedit* and *syngram* regardless of the LAS version of the input file. Since *logedit* and *syngram* cannot handle the multiple log data sections that are allowed in LAS 3 files, or more than ten logs effectively, or logs containing non-numeric data (eg. lithology), *readlas* provides graphical user interface (GUI) windows that prompt users to pick a section, decide which logs they want to edit, and replaces any character data with log nulls.

INTRODUCTION

The Log ASCII Standard (LAS) format is defined in documents provided by the Canadian Well Logging Society (CWLS). These documents, as well as software to validate existing LAS files, are provided online (CWLS, 2013). In general, LAS files are organized into named parameter or data sections, where the section name appears on a line that starts with a '~'. Parameter sections contain mnemonics with information about the well or about the data sections, with one mnemonic per line Versions 1.2 and 2.0 define well log data using three sections that can be minimally named: *~P, ~C, ~A*. Some LAS files extend these names to *~Parameter, ~Curve, ~Ascii* for clarity. Version 3.0 allows these names, but also extends the naming system to allow *~*_Parameter, ~*_Definition* and *~*_Data*, where * could be *Log, Tops, Core*, etc. This version also allows multiple data sections, For example *~Log_Definition[1], ~Log_Definition[2], ~Log_Data[1], ~Log_Data[2],* which was not possible in earlier versions. The data format can be optionally defined in the *Definition* sections in curly brackets for each data column, for example, *{F}* for float, *{E}* for exponent, *{S}* for string. Finally, the separator between data columns in the LAS file can be defined in the ~Well section as SPACE, COMMA or TAB. Versions 1.2 and 2.0 specified that the separator must be spaces, and *{F}* was implied to be the only allowed data type. Finally, long data rows are allowed to wrap in version 2.0, but not in versions 1.2 and 3.0.

Legacy CREWES Matlab code to read LAS files for use in programs such as *LOGEDIT, LOGSEC, SYNGRAM* was able to read version 1.2 and 2.0 (wrapped and unwrapped) LAS files, but could not handle version 3.0. A project was undertaken to write generalized code that makes as few assumptions about the LAS file as possible in order to read the file into Matlab.

READLAS

Within the CREWES Matlab toolbox, *crewes\z_legacy\io\readlas.m* is now a wrapper that calls *crewes\z_legacy\io\@las\las.m* to read a LAS file from disk and create a *las* object. The *las.readlas* method then interrogates the contents of the *las* object, and returns the outputs expected from the original *readlas* to the calling program. LAS version 2.0 information is returned regardless of the version of the input file. The original code is still present in the CREWES toolbox, but has been renamed *readlas_old.* Keeping this code appears to be important, as we have had issues where we were told the new code does not work, but the issue turned out to be users running old versions of Matlab.

The new *readlas* has been tested on many LAS files, and is now officially being used by *SYNGRAM* and *LOGEDIT.* If you have a LAS file that our code fails to read, we'd like to request that you send a copy of it to support@crewes.org. These files are editable in any text editor, such as notepad – so for the sake of confidentiality, please feel free to delete any location or company information. We would, however, ask that you provide at least three data lines.

Public methods of primary importance

w = las(filename)

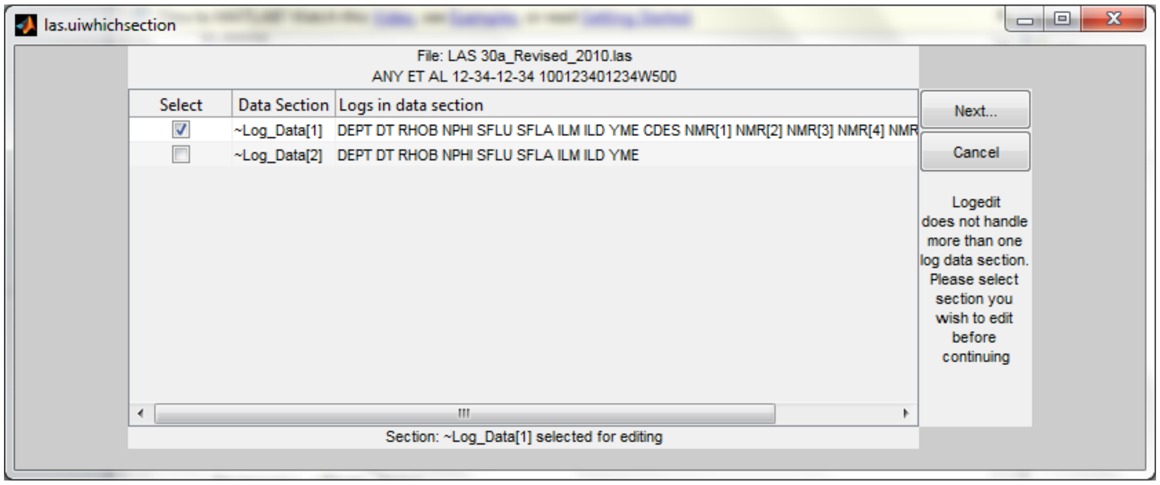
las() is the constructor for the *las* object, *w*. Inputs are either void or a filename as a character string. Output is a single *las* object. An entire LAS file is read into the *las* object after being split into individual character strings by the regular expression documented in Appendix A.

[logmat,mnem,desc,name,id,loc,null,units,kb,tops,ztops,lash]=w.readlas()

w.readlas() interrogates the *las* object and returns the variables (Figure 1) expected by other programs in the CREWES toolbox in LAS version 2.0 format. If multiple data sections (LAS version 3.0) exist, *las.uiSelectSection()* is called (Figure 2). If more than ten logs exist in the selected data section, *las.uiDeleteLogs()* is called (Figure 3), as *LOGEDIT* cannot be effectively used on more than ten logs.

```
%   logmat ... matrix containing the logs in the file. Each log is in a
%              column and the first column contains the depths. Thus,
%              if logmat has n columns, then there are n-1 logs
%   mnem   ... matrix of standard 4 letter mnemonics of the n columns
%              of logmat
%   desc   ... matrix of string descriptors of the n columns of the
%              logmat
%   name   ... string with the well name
%   id     ... the unique well id
%   loc    ... string containg the well location
%   null   ... the null value used in the logs
%   units  ... string indicating the depth units of the log
%   kb     ... elevation of the kelly bushing
%   tops   ... names of tops found in the file
%   ztops  ... depths of tops
%   lash   ... matlab string matrix containing the entire las header
```

FIG. 1. Description of variables returned by *readlas*.



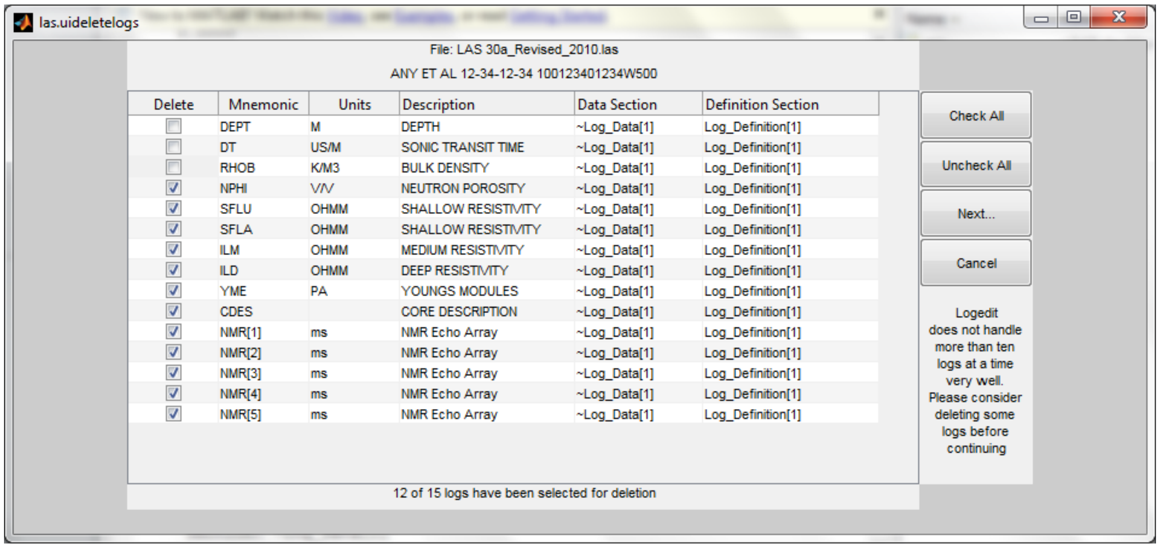
File: LAS 30a_Revised_2010.las
ANY ET AL 12-34-12-34 100123401234W500

Select	Data Section	Logs in data section
✓	~Log_Data[1]	DEPT DT RHOB NPHI SFLU SFLA ILM ILD YME CDES NMR[1] NMR[2] NMR[3] NMR[4] NMR
	~Log_Data[2]	DEPT DT RHOB NPHI SFLU SFLA ILM ILD YME

Logedit does not handle more than one log data section. Please select section you wish to edit before continuing

Section: ~Log_Data[1] selected for editing

FIG. 2. Example of *las.uiSelectSection()* forcing user selection of a single data section.



File: LAS 30a_Revised_2010.las
ANY ET AL 12-34-12-34 100123401234W500

Delete	Mnemonic	Units	Description	Data Section	Definition Section
	DEPT	M	DEPTH	~Log_Data[1]	Log_Definition[1]
	DT	US/M	SONIC TRANSIT TIME	~Log_Data[1]	Log_Definition[1]
	RHOB	K/M3	BULK DENSITY	~Log_Data[1]	Log_Definition[1]
✓	NPHI	V/V	NEUTRON POROSITY	~Log_Data[1]	Log_Definition[1]
✓	SFLU	OHMM	SHALLOW RESISTIVITY	~Log_Data[1]	Log_Definition[1]
✓	SFLA	OHMM	SHALLOW RESISTIVITY	~Log_Data[1]	Log_Definition[1]
✓	ILM	OHMM	MEDIUM RESISTIVITY	~Log_Data[1]	Log_Definition[1]
✓	ILD	OHMM	DEEP RESISTIVITY	~Log_Data[1]	Log_Definition[1]
✓	YME	PA	YOUNGS MODULES	~Log_Data[1]	Log_Definition[1]
✓	CDES		CORE DESCRIPTION	~Log_Data[1]	Log_Definition[1]
✓	NMR[1]	ms	NMR Echo Array	~Log_Data[1]	Log_Definition[1]
✓	NMR[2]	ms	NMR Echo Array	~Log_Data[1]	Log_Definition[1]
✓	NMR[3]	ms	NMR Echo Array	~Log_Data[1]	Log_Definition[1]
✓	NMR[4]	ms	NMR Echo Array	~Log_Data[1]	Log_Definition[1]
✓	NMR[5]	ms	NMR Echo Array	~Log_Data[1]	Log_Definition[1]

Logedit does not handle more than ten logs at a time very well. Please consider deleting some logs before continuing

12 of 15 logs have been selected for deletion

FIG. 3. Example of *las.uiDeleteLogs()* asking for user selection of fewer than ten logs.

Public methods of secondary importance

In the following, *w* is a *las* object, *ca* is a cell array, *st* is a character string, *db* is a double, *sn* is character string containing a full or partial section name, *mn* is character string containing a full mnemonic, and *w* is a *las* object

ca = getSectionNames(sn)

w.getSectionNames(sn) returns a cell array of character strings containing all of the section names in the *las* object that match *sn*. Example: *w.getSectionNames*(‘~w’) might return {‘~Well’}, but *w.getSectionNames*(‘~log_data’) might return {‘~Log_Data[1]’, ’~Log_Data[2]}.

ca = getSectionAssociation(sn)

w.getSectionAssociation(sn) returns a cell array containing all of the section names containing definitions that are associated with section names that match *sn.* This method will be renamed *getSectionDefinitions* in a future release. For example: *w.getSectionAssociation('~log_data')* might return {'~Log_Definition[1]', '~Log_Definition[2]'}.

ca = getSection(sn)

w.getSection(sn) returns a cell array containing all of the contents associated with all section names in the *las* object that match *sn*.

ca = getSectionData(sn)

w.getSectionData(sn) returns a cell array of character strings containing all data for all section names in the *las* object that match *sn*.

db = getSectionDataAsDouble(sn)

w.getSectionDataAsDouble(sn) returns an array of doubles or a cell array of doubles containing all data for all section names in the *las* object that match *sn*, depending on whether there are single or multiple matches to *sn*. Any numeric data stored as character strings are converted to doubles, and any non-numeric data are converted to the log null value. This method is particularly useful if you want to use math on well log curves.

ca = getParameterSectionMnemonics(sn)

w.getParameterSectionMnemonics(sn) returns a cell array of character strings containing all mnemonics for all section names in the *las* object that match *sn*. If the section is a data section rather than a parameter section, this method will return {''}.

ca = getParameterSectionUnits(sn)

w.getParameterSectionUnits(sn) returns a cell array of character strings containing all mnemonic units for all section names in the *las* object that match *sn*. If the section is a data section rather than a parameter section, this method will return {''}.

ca = getParameterSectionValues(sn)

w.getParameterSectionValues(sn) returns a cell array of character strings containing all mnemonic values for all section names in the *las* object that match *sn*. If the section is a data section rather than a parameter section, this method will return {''}.

ca = getParameterSectionValuesAsDouble(sn)

w.getParameterSectionValuesAsDouble(sn) returns a cell array of doubles containing all mnemonic values for all section names in the *las* object that match *sn*. If the section is a data section rather than a parameter section, this method will return {''}. If the data is non-numeric, this method will return NaN. This method will return log nulls instead of NaN in a future release.

ca = getParameterSectionDescriptions(sn)

w.getParameterSectionDescriptions(sn) returns a cell array of character strings containing all mnemonic descriptions (comments) for all section names in the *las* object that match *sn*. If the section is a data section rather than a parameter section, this method will return {''}. This will be renamed *getParameterSectionComments* in a future release.

ca = getParameterSectionFormats(sn)

w.getParameterSectionFormats(sn) returns a cell array of character strings containing all mnemonic formats for all section names in the *las* object that match *sn*. If the section is a data section rather than a parameter section, this method will return {''}.

ca = getParameterSectionAssociations(sn)

w.getParameterSectionAssociations(sn) returns a cell array of character strings containing all mnemonic associations (definitions) for all section names in the *las* object that match *sn*. If the section is a data section rather than a parameter section, this method will return {''}. This will be called *getParameterSectionDefinitions* in a future release.

st = getMnemonicUnit(sn,mn)

w.getMnemonicUnit(sn,mn) returns a character string containing the unit for the mnemonic *mn* contained in the parameter section *sn*. This method will return '' if there are no units associated with *mn,*For example, *w.getMnemonicUnit ('~log_definition[1]','dept')* might return 'M'

st = getMnemonicValue(sn,mn)

w.getMnemonicValue(sn,mn) returns a character string containing the value for the mnemonic *mn* contained in the parameter section *sn*. This method will return '' if there are no values associated with *mn,* For example, *w.getMnemonicValue('~w','lati')* might return '34.56789'.

st = getMnemonicDescription(sn,mn)

*w.getMnemonic*Description*(sn,mn)* returns a character string containing the description (comment) for the mnemonic *mn* contained in the parameter section *sn*. This method will return '' if there are no values associated with *mn,* For example, *w.getMnemonic*Description *('~w','API')* might return 'API NUMBER'. This method will be renamed *getMnemonicComment* in a future release.

st = getMnemonicFormat(sn,mn)

w.getMnemonicFormat (sn,mn) returns a character string containing the value for the mnemonic *mn* contained in the parameter section *sn*. This method will return '' if there are no values associated with *mn,* For example *w.getMnemonicFormat ('~log_definition[1]', 'dt')* might return 'F'.

v = getMnemonicAssociation(sn,mn)

*w.getMnemonic*Association*(sn,mn)* returns a character string containing the association (definition) for the mnemonic *mn* contained in the parameter section *sn*. This method will return ‘’ if there are no values associated with *mn,* For example, *w.getMnemonicAssociation('~log_parameter','dref')* might return ‘RUN[1]’. This method will be renamed *getMnemonicDefinition* in a future release.

SUMMARY AND FUTURE WORK

While well tested, the code needs to be optimized, better commented, and extended. Set methods need to be written that would allow creation of LAS file. Also, it would be desirable to have a *writelas* method. Finally, while we haven’t seen such a file yet, it may be desirable to be able to merge multiple data sections before editing in *LOGEDIT*.

REFERENCES

Canadian Well Logging Society (CWLS), 2013, http://www.cwls.org/las_info.php

APPENDIX A

The following code (Figure A.1) splits a LAS file into a Matlab array of structures containing the variables named *section, comment, mnemonic, munits, mvalues, mcomment, mformat, mdefinition* and *data,* each of which may be empty or contain character strings, depending on which line from the LAS file was stored. The data lines produced by the code shown will need to be further split into arrays, where columns of data are separated by spaces (LAS versions 1.2, 2.0) or commas, spaces or tabs (LAS version 3.0).

```
%Linetype1: ~SECTION NAME
%Linetype2: ddd.ddd [comma, space or tab] ccccc cccc cc [comma,
%           space or tab] ...
%Linetype3: ccc cc ccc [comma, space or tab] ddd.ddd ....
%           *** NOT HANDLED ***
%Linetype4: MNEMONIC .UNITS  VALUE1 VALUE2  : COMMENT {FORMAT} |
%           DEFINITION
% Pattern definitions used for regexp
%   ?<variablename> = variablename to assign a match
%   ^               = match beginning of string
%   [ ]             = match any characters listed in the square
%                     brackets
%   [^]             = match any characters not listed in the square
%                     brackets
%   \s              = white space characters
%   \d              = digit
%   \S              = non-white space characters
%   ()              = group
%   ?               = match zero or one character
%   *               = match zero or more characters
%   +               = match one or more characters
%   .               = wildcard, stands in for any character
%   |               = logical or
%   \|                 = '|'
%   ~                  = '~'
pattern = [                                     ...
  %Linetype 1 if first character is '~', *OR*
  '^\s*(?<section>^~.*)|'                       ...
  %Linetype 2 if first char is '#', *OR*
  '^\s*(?<comment>^#.*)|'                       ...
   %Linetype4 if the first '.' is not followed by a digit
  '^\s*(?<mnemonic>[^.]*)\.(?=[^\d])'           ...
  %   mnemonic units
  '(?<munits>\S*)'                               ...
  %   mnemonic values
  '\s*(?<mvalues>[^:]*)'                          ...
   %   a single ':' followed by zero or more spaces
  '\:?\s*'                                      ...
  %   mnemonic comment, all characters to the left of a '{' or '|'
  '(?<mcomment>[^|{]*)'                          ...
  %   log data format; eg. {F},{E},{S} for float, exponent, string
  '{?(?<mformat>[^|}]*)}?'                       ...
  %   zero or more spaces, single '|', zero or more spaces
  '\s*\|?\s*'                                   ...
  %   associated section name or mnemomic, *OR*
  '(?<mdefinition>.*)|'                          ...
  %Linetype3 contains data; This line will need to be split
  '^\s*(?<data>.*)'
];

las_struct = regexp(las_strings, pattern, 'names');
```

Figure A.1. Code to split a LAS file into a Matlab array of structures