GridLog

Logging is sometimes of only incidental importance. One should never make that assumption when it comes to security or system health, but let's face it. There are no log files on the New York Times bestseller list.

That's both a shame and an opportunity. A dry and dusty text file is not going to tell a story to applications people or to business management. Everybody likes spreadsheets, though. Almost no one uses them for logging, but they can be a great fit.

Sort your logging or data analysis across multiple tabs, and users can navigate your logs. What might have been buried in a mind-numbing text file can jump out. It's also a nice chance to show above-and-beyond initiative.

Getting a spreadsheet extracted from log files can be tedious. GridLog can either be incorporated into a larger application, or it can be used with code to scan conventional logs, sorting entries by regular expression or other methods.

GridLog is an example of a class that seems simple in concept, ends up being a little less simple to implement, and offers a chance to focus an existing package on a common use case. There are many reasons to write spreadsheets. Logging is a special case, and GridLog does the job with a simple YAML config, one line to create and initialize log files and a log spreadsheet, and one line to close all the files.

A single line of code puts log data into an appropriate spreadsheet tab.

GridLog theory of operation

GridLog has a single dependency:

- openpyxl - https://foss.heptapod.net/openpyxl/openpyxl

Logging is routed by specifying a destination Excel workbook and sheet (tab). Each tab optionally has a companion text log file.

Logging to text files adds a time stamp and the column title (if one exists) for the column of each item being logged. A separate log line is written for each item. Of interest, the log file names should not include a file type or extension. The name "table_one" will be written to disk with the name "table_one_yyyy-mm-dd_hh-mm-ss.log".

An "item" in this case is a cell in a spreadsheet row. You get a clean text log file ready for grepping or any other technique a system admin would apply, and you also

get a Excel spreadsheet ready for consumption by non-technical audiences. In most cases, the spreadsheet is ready to go with column widths and row heights already set to reasonable values.

There are four basic operations to consider.

Configuration is done in YAML. The root layer of the YAML structure is spreadsheets. A section called “tabs” contains a dictionary of tab (sheet) names.

Each sheet can specify a log file, a description, row and column headers, and a cols section.

Inside cols, there is a list of column addresses (A, B, C, etc.), each with a title attribute. If more than one column header is specified with the column header specification, the rows in a column header are delimited by pipe (“|”) characters.

The description attribute is used to populate a summary tab, which lists the tabs in the spreadsheet with a description of each. The summary tab may not conveys a form of soft information that’s very important.

Data without context is not as useful as information with an explanation. The table descriptions are a place to document what each tab in the workbook represents.

YAML specification

Here is a sample configuration:

```
cfg = """
--
test.xlsx:
  tabs:
    Sample Tab:
      logfile: sample_tab
      description: >
        This is a description for the
        sample_tab sheet in text.xlsx.

        This description will appear in the Summary
        tab, identified as the description for Sample Tab.
      rowheaders: 0
      colheaders: 2
      cols:
        A:
          title: Sample source|test data
        C:
          title: Extended info
```

This YAML specification defines a single spreadsheet called text.xlsx with a single tab (in addition to the automatically created Summary tab) called Sample Tab.

Everything written to Sample Tab is also written to a text log file called sample_tab_yyyy-mm-dd_hh-mm-ss.log. The time at the beginning of the run will be used for the time fields in the log file name.

If test.xlsx exists, it will be overwritten.

Sample Tab uses the first two rows for column titles. Columns A and C have titles. Column A uses both available rows for Sample source followed by test data.

Column C just uses the top row for a title, Extended info.

The first two rows will be frozen.

GridLog.__init__

GridLog's constructor takes a single argument, a string containing YAML specifications as described above.

A new workbook object is created and the Summary tab is created from tab names and descriptions in the YAML. Header rows and columns are left un-frozen at this point, and row and column widths are left at default since the content to be logged isn't known at this time.

A new GLworkbook variable is added to each workbook (top level) section of the YAML dictionary. This variable is the openpyxl.Workbook object used for the spreadsheet data.

Each tab using a secondary text log file will get a file handle stored as a GLlogfile variable in the section of the YAML for the tab.

GLworkbook and GLlogfile variables can generally be ignored, but it's also OK to use them directly.

GridLog.addRow

The addRow class method takes three arguments, a workbook name such as test.xlsx, a tab name such as Sample Tab, and a list or tuple of cells to add to a new row in the indicated spreadsheet tab.

If the tab has a secondary text log file, one new line per cell in the row data will appear in the tab's log.

GridLog.close

The close method requires no arguments, and does some housekeeping as well as closing files.

The cells in all rows across all workbooks and tabs are scanned for maximum height, and the cells in all columns are scanned for maximum width.

Determining width of variable fonts is not an exact thing, so an estimation of font size is made. It's probably OK, but in some cases cell sizes may need adjustment based on any font changes made in the spreadsheet.

Row and column headers are frozen at this time.

The workbook is saved and all log files are closed.

Typical usage

```
# build the specification
cfg = """
--
test.xlsx:
  tabs:
    Sample Tab:
      logfile: sample_tab
      description: >
        This is a description for the
        sample_tab sheet in text.xlsx.

        This description will appear in the Summary
        tab, identified as the description for Sample Tab.
      rowheaders: 0
      colheaders: 2
      cols:
        A:
          title: Sample source|test data
        C:
          title: Extended info

spreadsheetLog = gridlog.GridLog(cfg)
# add a row to the spreadsheet
spreadsheetLog.addRow('test.xlsx', 'Sample Tab', ('Column A data here',
                                                  'Column B data here',
```

```
                                                        'Column C data
here'))
# add more data with additional calls to addRow
# now close the logs and write the spreadsheet to disk
spreadsheetLog.close()
```