CSV Profiler
User Documentation

Version 1.2.0
March 29, 2021

Copyright Notice

Copyright (C) 2021 Larry Kuhn.
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Table of Contents

Revision History

Version 1.2.0

Current Release (March 24, 2021)

1. Added configuration filename to output report.
2. csvprofiler.py will now “return rc” when called or sys.exit(rc) when it is the main script.
3. Column numbers are added to the end of column headers in the error CSV output file.
4. Profile options allowed are y/n/p/u/s – ‘p’ for the default column profile (for occurrence counts, y/n for backward compatibility), ‘u’ for unique profile, ‘s’ for statistical profile.
5. Nine (9) new built-in named tests (regular expression tests for different latitude/longitude formats).
6. Minor documentation edits and corrections.

Version 1.1.3

Original Release (May 25, 2020)

Introduction

This open-source project was developed to provide a feature-rich CSV file validation and profiling utility without the need for writing any code. It was also designed to allow quite a bit of extensibility for those understanding Python and regular expressions, and with the expectation that the main testing module – profmod.py – be used in the future to validate other file types. It should serve as a helpful tool for those working with extremely large files or who work with relatively large files of the same type on a routine basis where validation would be useful.

Extensive documentation is provided to ensure a deep understanding but the process is pretty straight forward knowing just the basics. A simple file layout and validation setup should yield a simple parameter set. The more complicated the file and testing needs, the more complicated the parameter file. If the data is well understood and there is a fair grasp of the parameter options, it may only take a few minutes to setup and start a validation run against a CSV data file. More complex files or specialized tests will take longer to develop especially if "xcheck" testing (cross-check testing), which is interrelated column testing, is being performed.

Modules

This project is made up of the following Python modules:

csvpcg.py: This is the CSV analysis and configuration generator tool for setting up new profiling jobs.

csvprofiler.py: This is the CSV Profiler tool that uses profmod.py to execute the tests and generate column (field) report data.

profmod.py: This is the main test processing engine that is utilized by the other modules. It can be run directly from the command line with no arguments to exercise the built-in testing routine.

wrapdemo.py: The wrapper demo module can be repurposed for adding new test functions external to csvprofiler and profmod including capabilities like database look-ups.

Prerequisites

The software was developed and tested using Python 3.8.2 on Windows 10 and requires a recent version of Python 3 installed on the target system. As of version 1.2.0, pandas is required as it is used for the statistical profile option. If statistical profiling is not required or there are occasions where pandas will not be available on the target system, the import statement can be moved below code line 900 where it is used during reporting only when statistical profiling was chosen. The code has been tested on Ubuntu with Python 3.6.9 and CentOS with Python 3.6.8. When run with a version prior to Python 3.7, two of the Python built-in tests are not available – the strings and bytes isascii functions. A spreadsheet editor is also recommended as editing the parameter set in a text editor is error prone.

Quick Start

Once Python is installed and the script files have been downloaded, they can be run either from a user executable library location, from the directory of the CSV data file or by using path(s) and file name(s) on the command line. It is recommended to keep this documentation available for reference while editing the configuration and column parameters files. For simple setups, use of "Named Tests", the "profile" option and "lookup" tables is likely sufficient. Here is the procedure for producing a validation run by executing the scripts from the command line (note that you may need to precede the script name with "python" or "python3" depending on your environment and how the software is installed):

1. **Run → `csvpcg.py inputfile.csv [codec]`**
   a) This will run the configuration generator script which will analyze the input CSV file and output a small test file, a complete configuration file, and a parameter file template constructed specifically for the input CSV file. The parameter file contains recommended Column Tests and field length validations based on the file analysis. You may need to specify a codec if csvpcg throws a UnicodeDecodeError.
2. **Edit the configuration text file to verify or change input/output file locations and other job level options.**
   a) Do not delete or comment out any of the options.
   b) Strongly consider leaving the "csv_file = " parameter pointing to the small test file until all parameters have been tested.
3. **Edit the parameter file in a spreadsheet editor (keep it in CSV format and beware auto-corrections).**
   a) Column 1 contains labels for each row with the parameters expected for each column.
   b) Columns 2-n represent each column 1-n of the input CSV file. This is where the test specifications will go. All options can be left blank to disable the feature.
   c) Change or remove the parameter values as needed but do not remove or rearrange any of the labeled rows or change column 1. Do not remove columns; there must be, at minimum, a column header for every input column (csvpcg.py creates the column headers).
4. **Run → `csvprofiler.py ...csvp.cfg`** (using ...csvp_test.csv)
   a) Using the small test file is highly recommended as one small discrepancy could result in a large amount of undesired error output.
   b) Review the generated output files and report to make sure everything is running as expected.
   c) When satisfied with the parameters, edit the "csv_file = " configuration setting to specify the original CSV input file.
5. **Run → `csvprofiler.py ...csvp.cfg`** (using inputfile.csv)
   a) The console will display progress for every 100,000 records processed.
   b) The script will output a report file when the process has completed, and other optional output files depending on settings in the configuration file.

Quick Example

A CSV file is provided by a customer, vendor or partner and you have a general idea of what it contains but would like some more information before processing it into the system (loading into a database, converted, etc.). After running the configuration generator script, the parameter file template it creates looks like the following:

csvp_options	Customer	Region	Item Cat	Priority	Order ID	Order Date	Ship Date	Unit Price	Unit Cost	Units Sold
Column Test	Name	Name	Name	Name	ssn	mmddyyyy	mmddyyyy	decimal	decimal	digit
Column Length				1	9					
Max Length										
Profile (y/n/p/u/s)										
Blank is Error (y/n)										
Strip Surrounding Spaces (y/n)	y	y	y	y	y	y	y	y	y	y
Error Output Limit	50	50	50	50	50	50	50	50	50	50
Error Output Limit - Length Errors										
Error Output Limit - Blank Errors										
User Data										

The following observations and changes are made:

1. The Customer field in your system is limited to 65 characters so you place a max length check on the column. The “Name” Column Test is fine and if any of the “Name” tests encounter non-ASCII accented characters they will be reported.
2. You know the Regions are defined categorically, so you profile them. The “Name” test is fine.
3. The same with Item Category.
4. The Priority field is a code 1 character in length, so you leave that set to 1 and profile it as well. You know it contains an uppercase alpha character so you change the test to “ALPHA”.
5. The Order ID is not a social security number, but it appears to be consistently 9 digits, so you make the Column Test change to “digit” to avoid any misunderstanding.
6. The remaining Column Tests all appear correct so you leave them as is.

csvp_options	Customer	Region	Item Cat	Priority	Order ID	Order Date	Ship Date	Unit Price	Unit Cost	Units Sold
Column Test	Name	Name	Name	ALPHA	digit	mmddyyyy	mmddyyyy	decimal	decimal	digit
Column Length				1	9					
Max Length	65									
Profile (y/n/p/u/s)		y	y	y						
Blank is Error (y/n)										
Strip Surrounding Spaces (y/n)	y	y	y	y	y	y	y	y	y	y
Error Output Limit	50	50	50	50	50	50	50	50	50	50
Error Output Limit - Length Errors										
Error Output Limit - Blank Errors										
User Data										

You run the file and obtain a report on any deviation from the Column Tests reported as errors, deviation from the Customer, Priority and Order ID lengths reported as errors, and a look at the unique values for Region, Item Category and Priority, along with their quantities. Additional field and file counts are provided as well.

Configuration File

Paths

`file_path`	allows use of %(file_path)s for shorthand file locations
`csv_file`	path and name of input csv file to process
`param_file`	path and name of column parameters file
`report_file`	path and name of output report file (described below)
`error_path`	allows use of %(error_path)s for shorthand file locations
`error_csv_file`	path and name of output error CSV file (described below)
`error_log_file`	path and name of output error log file (described below)
`lookup_(name)`	(optional) path and name of a text file to use for a “lookup” option; there can be as many of these as required as long as the “name” portion is unique. For imported user functions see *User Functions & Lookup Providers*.
`regex_(name)`	(optional) path and name of a text file to use for a “regex” option; there can be as many of these as required as long as the “name” portion is unique.
`xcheck_(name)`	(optional) path and name of a text or CSV file to use for an “xcheck” option; there can be as many of these as required as long as the “name” portion is unique.

CSV Settings

These are set by the configuration generator script (csvpcg.py) based on information pulled from the Python built-in csv module “csv Sniffer”. It is unlikely that they need to be modified.

`has_header`	If set to True, the software will assume row 1 is a header row and skip over it during processing.
`delimiter`	Instructs the csv module to use the provided character as the column delimiter, typically a comma “,”. Do not use quotes in the file.
`escapechar`	Only specify if an escape character (e.g. “\”) is used in the file for escaping delimiters or quote characters within column data. Typically set to None.
`quotechar`	The character used when the csv module needs to enclose a column value in quotes due to special characters within the field, such as the delimiter. Typically set to – and defaults to – the doublequote character “"”.
`doublequote`	If set to True, the csv module will double the quote character when it appears within column data, as opposed to escaping it.

quoting	Determines the conditions under which the csv module writes the quotechar. It can also impact how the CSV file is read into the csvprofiler.py script. See the comments in the config file for additional information, or Python documentation at the link below. If it is set to either QUOTE_MINIMUM or QUOTE_ALL it is probably best not to be bothered by it.
encoding	Specifies the codec to use for decoding the input csv file. Python is designed to support Unicode using utf-8. The default codec used by csvpcg is utf-8.

For additional information, refer to:
https://docs.python.org/3.8/library/csv.html#dialects-and-formatting-parameters
https://docs.python.org/3/library/codecs.html

Output Settings

output_error_csv	If set to True, the software will write a copy of the input CSV record to the output CSV file when an error is detected. An additional column will precede each row which will contain a space-separated list of the column numbers where the errors were detected.
output_error_log	If set to True, the software will write a record to the log file for each column in the row found to be in error. Log file format details can be found below.
key_colnum	Not a misspelling, it represents the column number to be used as the key field to display in the error log. If 0, the record sequence number is used. The record sequence number ignores the header row if there is one.
error_limit	This error limit overrides all limits specified in the column parameters (i.e. this specifies the upper limit for all records written). If 0, no error records are written. If no value is specified, error output is only limited by column parameter limits, if any. Note that the log file may have more records written than this limit as the comparison is based on input records in error, not the number of records written to the log file, which could exceed one per input record. Consider using this limit during testing or in production when few errors are expected.
verbose	If set to True, the software will generate a dump of internal dictionaries from both csvprofiler and profmod at the end of parameter processing, and again at the end of the run to assist in debugging parameter settings. Not always helpful and very verbose, but viewing the named tests dictionary can sometimes uncover a test name typing error. Defaults to False.

Codecs and Decoding Errors

csvpcg

When csvpcg runs without a codec specified, it defaults to using the utf-8 codec and an error handler referred to as “strict”. In this mode, if Python encounters a character it cannot decode while reading the CSV file (such as those in the Windows 1252 Code Page in the range 128-159, or hex 80-9F), it will fail with a UnicodeDecodeError. In this case a decision must be made as to whether or not the character encountered is legitimate or not. If it is, the correct codec may very well be cp1252. Regardless of the decision, to proceed, a codec must be specified as the second parameter on the command line. Specifying a codec on the command line impacts csvpcg in a number of ways:

- It will use the codec specified when opening the CSV file and change the error handler from “strict” to “replace”, which will alleviate the codec error termination issue, even if an invalid character is encountered.
- By changing the error handler to “replace”, any character encountered that the codec cannot find a suitable utf-8 replacement for will be replaced with the standard Unicode replacement character “�”. The replacement character, by the way, will fail every profmod built-in test except “something”, “anything” and “isprintable”.
- The encoding parameter in the output configuration file will be set to the codec provided.
- The output test CSV file will be written using the codec provided.

csvprofiler and profmod

csvprofiler will read the configuration file and will open the input CSV file with the codec specified. It will also pass the codec over to profmod. Other than the configuration file, any files read or written by either program will use the codec specified along with the error handler “replace” as described above.

Parameter File

A parameter template file will be built specifically for the CSV input file, generated automatically by csvpcg to save time editing.

Column Header

The first row contains the header from the file, if it exists and was detected by the csv Sniffer. If there is no header detected in the input file, the software will generate a header for the parameter file to use for each column in the form Column1, Column2, etc. This is so that csvprofiler has a name to use and there is a column placeholder in the parameters so the software doesn’t fail on a missing column.

If the input has a duplicate header, the csv Sniffer will likely assume it is not a header which will cause the numbered column headers to appear. Also, the first row will be treated as data by the csvprofiler

script unless the configuration file is modified to indicate there is a header. The header values will have to be copied and pasted into the parameter file but the duplicate name should be changed. If there is a duplicate header name and the csv Sniffer accepts the header (unlikely), the software will modify the 2nd header by appending it with ($2), the 3rd with ($3), etc.

"Column 0"

The first column of the parameter file contains labels for each row, specifying the value belonging in that row; think of it as column 0. Each subsequent column represents the columns of the input file in strict order. The labels can be seen in the example below and are explained in the following documentation. The last labeled row, and those below it, are where additional User Data will go for a "*regex_(name)*" regular expression, "*lookup_(name)*" list parameters and "*xcheck_(name)*" test parameters.

Here is an example of what the start of a parameter file might look like in a spreadsheet with all of the available test types exercised in some form:

csvp_options	ID	Prefix	Name	Gender	Link_ID	Dept	Territory	T_State
Column Test	digit	Abbrev	Name	Alpha	regex_LID	lookup_Dept	xcheck_T	xcheck_T
Column Length	10				6			
Max Length		10	50	10		10	2	2
Profile (y/n/p/u/s)		y		y		y	y	y
Blank is Error (y/n)	y		y			y		
Strip Surrounding Spaces (y/n)		y	y	y		y	y	y
Error Output Limit		50	50		50		50	50
Error Output Limit - Length Tests		50	50	50	50		50	50
Error Output Limit - Blank Test			50					
User Data					(A\d{5}|B\d{5})	Admin	nothing	nothing
						Finance	E	(PA|NY)
						HR	SE	FL
						Training	MW	(IL|WI)
						Sales	S	TX
						Ops	NW	(WA|WY)
							W	CA
							G	range(1:99)

When editing the parameters, the key item will be specifying the value of the **Column Test** to use for each column of the input CSV file. It is important to understand the behavior of these tests and choose the most appropriate option combination for each column of the input CSV file.

Parameter Options

Here is description of the options that can be specified for each column of the input file (note that descriptions for each of the **Column Test** options will follow in more detail):

Column Header (csvp_options row): This will typically come from the header row if there is one, but it is important to provide a unique name for each column since it will be displayed along with the column number in the final report, with test results for that column.

Column Test: This will specify what type of True/False test to use for the column. It may be one of the following:

- **Named Test:** A name from the Named Tests table of built-in tests. The Named Tests table "remembers" any new test defined using the range, lookup or regex options.
- **The *range(from:to)* option**: (a.k.a. range()) Allows the user to define a range of integer, floating point or date values for the column. Each value in the column will pass the test if it falls within the defined range. These are stored in the Named Tests table for reuse.
- **The *lookup_(name)* option**: (a.k.a. lookup) Allows a user provided list of values from the User Data portion of the parameter file to be stored in an internal look-up table and the unique lookup name stored for use in the Named Tests table. It can also be used for accessing user provided functions in external Python modules.

  The *regex_(name)* option: (a.k.a. regex) Allows a regular expression test to be defined in the first User Data row, uniquely named and stored for use in the Named Tests table. This may be the most powerful tool as it allows precise column tests to be quickly developed for column values that should be following strict patterns.
- **The *xcheck_(name)* option**: (a.k.a. xcheck) Allows a multi-column cross check to be uniquely named for use by other interdependent columns. Each participating column will use the same "xcheck" name and the User Data rows will contain the tests to run for that column. This is useful for evaluating values in columns that cannot be effectively tested independently from values in other columns.

Column Length: When specified, will cause a column value to fail if it is not the specified length, regardless of the outcome of the Column Test. 0 or blank means no column length test is performed. Column length tests are only run with Named Tests, range() tests and regex tests (not with lookup or xcheck). Even if it fails, the Column Test is still performed. Note that multi-byte Unicode characters are counted as one (1) character. If a length range is required, use a regex.

Max Length: When specified, will cause a column value to fail if it is longer than the specified length, regardless of the outcome of the Named test. This test will always be run prior to any other tests and prior to **Strip Surrounding Spaces**. Even if it fails, the Column test is still performed. 0 or blank means no max length test is performed. This can be useful prior to loading the file into a database to ensure column size limits aren't exceeded.

Profile: Updated in v1.2.0, this option is now expanded to allow one of the values "**y/n/p/u/s**":

- **No Profiling:** When specified as no (**n**), or the cell is empty, no profiling is performed.

- **Profile option**: When specified as yes (**y**) or profile (**p**), the software collects counts for every unique value found in the column. It does not perform any validation but simply reports occurrence values and occurrence counts. It is helpful for doing demographic-like profiling of the contents of a column. The output report will contain a list of every unique occurrence found in the column along with the occurrence count, so it is not appropriate for columns with wide-ranging values (like street addresses), which could result in very large report files. This option is helpful if a validation list is not known or readily available ahead of time but the diversity is expected to be reasonable and a view into the file contents is desired. It is helpful in creating a validation list for future use and to uncover outliers that may need to be addressed. When Profile is used with a lookup, the output report will display valid *and* invalid occurrence values and counts. All occurrence values are output in ascending sorted order.

- **Unique Profile option:** When specified as unique (**u**), the column will be profiled looking to ensure that only unique values exist. This requires a copy of every input value to be stored and can result in the consumption of a large amount of memory. Duplicates found DO NOT result in an error for the record, but duplicated values are listed in the output report along with the record sequence numbers for the original and all following duplicates. Totals are provided for the number of field values inspected, the number of duplicated values (i.e. the number of original, unique values that were duplicated) and the total duplicates (i.e. the number of duplicate values found not including the original).

  For example, the following report excerpt indicates that 10 records were read, 2 values were duplicated ("a" and "b"), and 3 duplicates of those values were found (for sake of argument, assume the "invalid" duplicates are in record numbers 2, 6 and 7). A total of 5 (2+3) records would need to be interrogated:

```
                    a : 1, 2
                    b : 5, 6, 7

 (total field values) = 10
  (duplicated values) = 2
   (total duplicates) = 3
```

- **Statistical Profile option:** When specified as statistical (**s**), the column will be profiled using the pandas module to provide statistical analysis of the column values. This also requires a copy of every input value to be stored in a list in memory. At the end of the run, the list is converted into a pandas Series and the following statistics are run (either directly or using the "describe" method) and provided in the output report: count, mean, std, min, 25%, 50%, 75%, max, var, mad, median, skew, sem, and kurt. Fields that do not convert to float are dropped, as are NaN and Inf. All dropped values are counted and reported with the record sequence numbers where they were found. They DO NOT result in an error for the record.

Blank is Error: When specified as yes (y), will cause a column value to fail if it resolves to an empty field. When a field is empty, no other testing is performed on the field unless it is part of an xcheck. If the column is part of an xcheck test, no blank test will be performed automatically.

Strip Surrounding Spaces: Prior to any testing being performed (other than **Max Length**), this option is evaluated. When specified as yes (y), the Python strip() command will be run on the column which removes all whitespace from the leading and ending portions of the string until a non-whitespace character is encountered. This will impact the **Blank is Error** option if spaces exist as a column value and are not stripped (in this case the blank test would not fail).

Error Output Limit: This governs the number of output error records written for errors found in the column by a Column Test. This also defines the overall maximum for the column (overriding the Length Test and Blank Test limits). If 0, no error records are written due to errors encountered in this column. If no limit is specified, error records are written for all errors encountered in this column. This can be overridden at the record level (see *Configuration File Options*).

Error Output Limit - Length Tests: This governs the number of output error records written for errors encountered in the column by either the **Column Length** or **Max Length** tests. If neither of those options are specified, this parameter is ignored.

Error Output Limit - Blank Test: This governs the number of output error records written for errors encountered in the column by the **Blank is Error** test. If it was not set to 'y', this parameter is ignored.

User Data: This allows additional parameter data to be provided based on the **Column Test** specified. It is used only by regex, lookup and xcheck (see below for details on those options):

- **regex**: allows a single regex to be specified
- **lookup**: allows a list of comparison values to be specified (vertically)
- **xcheck**: allows a list of values, regular expressions, Named Tests, range()'s and lookup tests to be specified (vertically)

Named Tests

There are 5 different groups of Named Tests available:

- **Built-in Regular Expressions:** These tests leverage the regular expression capabilities of Python with custom patterns built to work with single-byte ASCII or ASCII extended text (except for latitude/longitude patterns that use Unicode symbols). They are designed to cover tests not explicitly addressed by the Python built-in tests, or in some cases to be a bit more restrictive about the characters expected in a typical CSV file (e.g. not including vertical tab or control codes as valid characters). Some tests duplicate the Python tests and are provided just for naming consistency.

- **Special Case Tests:** These hard coded tests are used to ensure blank and non-blank conditions can be handled in most instances, especially when not stripping whitespace (when whitespace only columns are expected as legal) or with xcheck test combinations. These should not be necessary in most cases.
- **Python String Tests**: These tests work with Unicode when multi-byte characters are expected and support a myriad of code points for uppercase, lowercase, etc. For example, code point 65313 is defined as Fullwidth Latin Capital A (`A`). When using the "istitle" test for instance, Python recognizes the lowercase version as Fullwidth Latin Small Letter A (`a`) as well as the other alphabetic characters in numerous alphabetic sets when evaluating the string for title upper/lower patterns. These tests will work with ASCII but may not report Unicode encountered in the text as an error, which may not be desirable if only ASCII or extended ASCII is expected.
- **Python Byte Tests:** These tests work with ASCII single-byte characters. To support these methods when specified in the parameters, the software needs to convert the Python "string types" to "bytes types", and uses UTF-8 encoding to do so prior to running the test. Using UTF-8 should prevent encoding errors (mojibake) since it can represent all Unicode values. When translated to bytes, a character such as 'À' which is a Latin Capital Letter A With Grave in the Windows-1252 code page is converted to a double-byte character. This character fails the `b.isupper` test, but the original character would pass the string `isupper` (Unicode) test, as well as the Windows (Windows-1252) built-in regular expression.
- **Python int() and float() Tests**: These hard coded tests were built to leverage the variety of numeric formats supported by these built-in Python conversion routines. Values that convert successfully are considered True and pass the test, all others are reported as errors.

Bottom line, it is probably best to avoid using the Python string tests if only ASCII is expected or the destination system only supports ASCII. Python string tests are best for Unicode files where Unicode code points above the ASCII range are expected, and the built-in regular expression tests may still be useful for columns that are limited to ASCII values. Python byte test are good for ASCII files but be aware of what they match. If the included tests are insufficient for the data, use the regex option, add new regular expressions or other tests to the source code (very easy), or use a lookup provider.

Built-in Regular Expressions

The table below describes the Named Tests developed for this project using regular expressions. They focus on character pattern validations for files expected to contain common, single-byte, printable characters in ASCII or ASCII extended code pages (as noted, except for latitude/longitude tests using Unicode symbols). Unicode data found in columns outside of the ranges specified will result in those data fields returned as errors. As a naming convention, the test names that contain only uppercase alpha characters have UPPERCASE NAMES, lowercase have lowercase names, and mixed have Title Case Names (except for ASCII). Examples are provided along with known issues, which are errors that might ***not*** be caught, not errors reported that are not errors (with the possible exception of creditcard and creditcard+ for obscure credit cards or newer numbers).

The built-in regular expressions can be modified to need, and custom regular expressions can be added into the source code (see section on *Customizing the Built-in Regular Expressions*). Most of the generic tests in this list do not limit the string length (use the length option), in others cases it is pattern specific.

Column Test Name	Description, Examples & Known Issues
Abbrev	one or more upper- or lowercase characters followed by a period '.': **M., Mr., Mrs., Ms., etc.**
Address	address text, upper- or lowercase letters and numbers plus space ' ', special characters parenthesis '()', periods '.', hyphen '-', comma ',' and forward slash '/', at least one: **123 1/2 S. 4th Street, Suite A(5)**
alpha	lowercase letters a-z, at least one: **abcdefg**
Alpha	upper- and lowercase letters a-z A-Z (the same as b.isalpha); at least one: **a, A, ABC, Cat, iPhone**
ALPHA	uppercase letters A-Z, at least one: **ABCDEFG**
alphanumeric	lowercase letters and numbers a-z 0-9, at least one: **a, 0, 5, a105, abc, 15a, a25d, 25a6**
Alphanumeric	upper- and lowercase letters and numbers a-z A-Z 0-9 (the same as b.isalnum); at least one: **a, A, a1, Title, ABCDEF, AB012345, 95N, 23y, o2**
ALPHANUMERIC	mixed uppercase letters and numbers A-Z 0-9; at least one: **1, A, 1A, A1, ABCDEF, AB012345, 95N, A1B2C3**
Alpha+	same as alpha with optional leading #, alpha char followed by optional mix of single separation characters '-_./' or enclosing parenthesis; at least one: **A, #A, #A-bc, AB_CDE, AB.CD.E-F(g), A(B)CD, Abc(a)** known issues: unbalanced parenthesis allowed, nested parenthesis may not work as expected
Alphanumeric+	same as Alpha+ with numbers allowed; at least one: **A, #A, 1, #123, #a0010203, A-101, Abc/1(a), A(101), A(b1)105, #AB-c.d(e15)** known issues: same as Alpha+
ALPHANUMERIC+	same as Alpha+ with numbers allowed; at least one: **A, #A, 1, #123, #a010203, A-101, ABC/1(A), A(101), A(b1)105, #AB-c.d(e15)** known issues: same as Alpha+
ASCII	keyboard characters in the ASCII range 32-126, at least one: **abcdefg ABCDEFG 0123456789 `~!@#$%^&*()-_=+[]{}|;:'",.<>/?** or see https://en.wikipedia.org/wiki/ASCII known issues: this does not include tab, vertical tab or newlines that may be inside of wrapping text fields (by design), **b.isascii** will accept these characters
ccnumber	possible credit card number, 12-19 numeric digits 0-9, no punctuation: **123456789012, 1234567890123456789**

Column Test Name	Description, Examples & Known Issues
	known issues: does not follow any published rules on financial provider number prefixes or formats
ccnumber+	numeric digit 0-9, followed by 11-22 numeric digits 0-9 or '-' and ' ' separators: **1234-5678-9012-3456, 1234 5678 9012 3456, 1234-5678-9012-3456-789** known issues: same as ccnumber, allows separators in illogical places
creditcard	numeric digits 0-9 only, formatted for published rules on financial provider number formats; no punctuation; should handle all US accepted credit cards; coded for Amex, Bankcard, CIBC, Dankort, Discover, Diners, Electron, HSBC, InstaPayment, InterPayment, JCB, Laser, Maestro, MasterCard, RBC, RuPay, Scotiabank, Solo, Switch, TD, Troy, UATP, Verve, Visa, VPay: **4532256932352187, 6011635431660585002, 3545359738319017003** known issues: number length variability limits validation strength in many cases, errors reported should be validated with an online tool to ensure accuracy
creditcard+	creditcard number plus allowance for '-' and ' ' separators; "should" handle all US accepted credit cards; coded for same institutions as creditcard: **6304-0477-8513-2854, 3671-363656-2189, 1324 35418 436821** known issues: same as creditcard, allows separators in illogical places
~~day~~	**use range(1:31)**; for strict rules where associated month is in another column, use xcheck to validate day range suitable for month list (attempting to validate leap year February days with year column would be difficult at best without a custom function)
decimal	wide variety of numeric patterns with punctuation: **1234, 0.123, 123.45, 12,345, 12,345.6789, +123, -123.45, +1,234.5, (1,234.567)** known issues: zero leading comma 0,123, unbalanced parenthesis, and .0 are allowed
digit	un-signed numbers; leading zero, other punctuation not allowed, at least one, or use **range()** for stronger validation: **0, 1, 10, 1000**
dollar	same as decimal except allows $ and restricts to 2 places after decimal point: **$1,000,000.00, $1, $1.23, ($5.00)** known issues: same as decimal
Email	unquoted email format, a mix of upper- and lowercase letters and numbers and allowed special characters !#$%&'*+-/=?^_`{|}~ , followed by @ and domain. **someone@example.com, someone-else+route@example.com** known issues: does not check formatting like double dots, for reserved domains, or compliance with ICANN approved top level domains.
integer	signed numbers; leading zero, other punctuation not allowed, at least one: **0, 15, -15, +15** see **int** test
ipaddress	4 groups of 1-3 characters, leading or no leading zeros, restricted to 0-255, separated by periods '.': **0.0.0.0, 255.255.255.255, 127.0.0.1**

Column Test Name	Description, Examples & Known Issues
ip+port	same as ipaddress but with optional port - a colon ':' followed by 1 to 5 numbers [0-9] **127.0.0.1:8080** known issues: largest port that passes is 99999 which is higher than 65535 limit
ip+cidr	same as ipaddress but with optional CIDR specification - a forward slash '/' followed by numbers in the range 0-32 **0.0.0.0/0, 127.0.0.0/24**
lat	latitude in sexagesimal with optional decimal minutes or seconds (similar to ISO 6709 Annex D) - sign: plus (+), hyphen (-), minus (\u2212), N or S (with optional space as prefix or suffix), followed by non-zero leading degrees [0-90], followed by degree sign (\u00B0) and optional space, followed by zero leading minutes [00-59] and optional minute decimal places (if so, no seconds), followed by single quote (') or single prime (\u2032) and optional space, followed by leading zero seconds [00-59] and optional second decimal places, followed by double quote (") or double prime (\u2033). **40° 26′ 46″ N, 15°24'15"N, 50°03′46.461″S, 40° 26.767′ N, N 38°53'23.3", 40°06.767′N, N 38° 53' 23.3", +90° 00' 00", −5° 03' 22.09"** known issues: doesn't allow decimal minutes or seconds for 90 degrees, doesn’t allow use of publishing quotation marks (‘’””), doesn’t enforce consistent use of punctuation
lat6709	ISO 6709 Annex H formatted latitude - supports ±DD.D, ±DDMM.M, ±DDMMSS.S formats; plus (+), hyphen (-) or minus (\u2212), followed by degree [00-90], minute [00-59], seconds [00-59] (as above), followed by (optional) appropriate decimal positions (up to 8 D, 6 M or 4 S); see https://en.wikipedia.org/wiki/ISO_6709 (excluding elevation). **+90, +90.0, -90.0, +900000, −0400.999999, −040101.0001**
latdec	latitude in decimal format - optional sign: plus (+), hyphen (-), minus (\u2212), N or S (with optional space as prefix or suffix), followed by zero or non-zero leading degrees [0-90], followed by optional 1-8 decimal places, followed by optional degree sign (\u00B0). **38.88980°, 32.30642° N, +32.30642, 09.00N, 0N, 90S, 8°S, +9, 0, N 90°**
Latin1	ISO-8859-1 html text standard, no code points removed; includes accented characters, currency symbols, superscript ¹²³ and symbols like ©®¼½¾, ranges 32-126, 160-255 see https://en.wikipedia.org/wiki/ISO/IEC_8859-1
latlon	latitude and longitude in sexagesimal with optional decimal minutes or seconds (similar to ISO 6709 Annex D) - lat format followed by space or comma (,) or semicolon (;) and optional space, followed by lon format. **40° 26′ 46″ N, 99° 58′ 56″ W | 15°24'15"N, 30°10'03"E | 50°03′46.461″S;125°48′26.533″E | 40° 26.767′ N 168° 59.0190′ W | N 38°53'23.3" E 179°10'3" | 40°06.767′N, 38°53.3′ E | +90° 00' 00", +180 00' 00"** known issues: enforces consistent prefix/suffix location but not consistent use of

Column Test Name	Description, Examples & Known Issues
	punctuation or depth of decimal place accuracy
latlon6709	ISO 6709 Annex H formatted latitude & longitude - lat6709 format followed directly by lon6709 format, no spaces or punctuation; see https://en.wikipedia.org/wiki/ISO_6709 (excluding elevation). **+89-079, +90.0+000, +890099-0010101, −0400.999999+12359.000000** known issues: allows any longitude at +/-90 latitude where +000 is probably the standard; enforces consistent formats but not depth of decimal place accuracy.
latlondec	latitude and longitude in decimal format - latdec format, followed by optional comma (,) or semicolon (;), followed by space and londec format. **38.88980°, -077.03654° | 32.30642° N 122.61458° W | +32.30642, -122.61458 | N 32.30642° W 122.61458° | N1, W3 | 1, 3 | 0 0** known issues: enforces consistent prefix/suffix but not consistent depth of decimal place accuracy or use of degree sign
lon	longitude in sexagesimal with optional decimal minutes or seconds (similar to ISO 6709 Annex D) - sign: plus (+), hyphen (-), minus (\u2212), E or W (with optional space as prefix or suffix), followed by non-zero leading degrees [0-180], followed by degree sign (\u00B0) and optional space, followed by zero leading minutes [00-59] and optional minute decimal places (if so, no seconds), followed by single quote (') or single prime (\u2032) and optional space, followed by leading zero seconds [00-59] and optional second decimal places, followed by double quote (") or double prime (\u2033). **79° 58′ 56″ W, 30°10'03"E, +179°10'3", 125°48′26.533″E, 129°08.933′W, 79°58.933′ W, W 79° 58.933′, +180 00' 00", −100° 43' 09.09"** known issues: doesn't allow decimal minutes or seconds for 180 degrees, doesn’t allow use of publishing quotation marks (‘’””), doesn’t enforce consistent use of punctuation
lon6709	ISO 6709 Annex H formatted longitude - supports ±DDD.D, ±DDDMM.M, ±DDDMMSS.S formats; plus (+), hyphen (-) or minus (\u2212), followed by degree [000-180], minute [00-59], seconds [00-59] (as above), followed by (optional) appropriate decimal positions (up to 8 D, 6 M or 4 S); see https://en.wikipedia.org/wiki/ISO_6709 (excluding elevation). **+180, +180.0, -180.0, +1800000, −09900.999999, −0990101.0001**
londec	longitude in decimal format - optional sign: plus (+), hyphen (-), minus (\u2212), E or W (with optional space as prefix or suffix), followed by zero or non-zero leading degrees [0-180], followed by optional 1-8 decimal places, followed by optional degree sign (\u00B0). **38.88980°, 32.30642° E, +32.30642, 09.00W, 0E, 180W, 8°E, +9, 0, W 90°**
mdyorymd	allows either **mmddyyyy** or **yyyymmdd**; see **range()** for date range support
mmddyyyy	1-12 or 01-12, hyphen '-' or forward slash '/', 1-31 or 01-31, '-' or '/', years 1900-2099 or 00-99 **1/1/99, 01/01/99, 12/25/2005, 02-02-1982** known issues: does not validate days by month or leap year, only 1-31 see **range()** for date range support

Column Test Name	Description, Examples & Known Issues
mmyyyy	1-12 or 01-12, hyphen '-' or forward slash '/', years 1900-2099 or 00-99 **1/99, 01/99, 12/2005, 2-1982** known issues: does not validate days by month or leap year, only 1-31
~~month~~	use range(1:12)
Name	upper- or lowercase letters, numbers, hyphen '-', apostrophe "'", double quote '"', period ".", comma ',' or space, at least one **Bob, 3M, Baba O'Riley, "Mr. John Smith-Anderson, MD"** known issues: no rules prevent punctuation location or punctuation only content
notation	optional sign (+-) followed by digit, followed by optional decimal and decimal place digits, followed by 'e' or 'E', followed by optional sign, followed by one or more digits **0e0, 1e5, 1.08E+9, -2.34e5, +1.0005e5, 9.1093822E-31, -2e-2** see **float** test using Python float() function
number	integers with optional thousand separators ',', no sign or decimal places (note that thousand separators will fail range() tests): **0, 1, 12345, 12,345, 1,234,567** known issues: zero leading comma 0,123
numeric	numbers 0-9, at least one, no punctuation, leading zeros allowed: **0, 000, 5, 05, 111111111, 00001111**
percent	0 or any number of digits (no leading zeros), followed by percent sign '%', no punctuation or decimal places: **0%, 1%, 100%, 200%, 1000%**
percent+	like **percent** but followed by optional period '.' and decimal places, or just decimal places, followed by percent: **0%, 0.1%, .01%, 100%, 200.345%, 1000%**
phone	optional (optional 1 and optional '-', '.' or ' ', followed by area code in or out of parenthesis, with optional ' ', or '.' or '-'), followed by 3 digits, optional '-' or '.', followed by 4 digits, followed by optional 1-5 digit extensions with optional separators and text formats (x, ext, ext.) as shown: **18001234567, 1 800 123-4567, 1(800)123-4567, 1 (800) 123-4567, 1-800-123-4567, 123-4567, 123-4567x1, "123.4567, ext1", 123-4567 x234, "123-4567, ext. 5"** known issues: no logic constraints over digit ranges like all 0's, mixed punctuation combinations that pass the test appear a bit odd (e.g. mix of hyphens and periods).
Sentence	sentence-like characters, ASCII less uncommon special characters '~`^_|\[]{}+=<>' known issues: not a grammar checker, simply a character class definition
ssn	9 digits; or 3 digits, hyphen '-', 2 digits, hyphen, 4 digits: **999999999, 999-99-9999** known issues: follows no rules other than those above for validity (e.g. area numbers)

Column Test Name	Description, Examples & Known Issues
time	digits 00-12 or 0-12, followed by optionally followed by colon ':' and followed by 00-59 (twice), optionally followed by optional space, a.m., p.m., AM or PM: **12:00PM, 11:00:00 a.m., 12:59:59 p.m., 1PM, 6 PM, 12a.m.**
time24	digits 00-23 or 0-23, followed by colon ':', followed by 00-59, optionally followed by colon and followed by 00-59: **00:00:00, 01:00, 1:23:45, 23:59:59**
@Twitter	mix of upper- and lowercase characters or numbers, or underscore '_', at least one up to 15: **@twitter, @twitter1, @twitter_1, @TWITTER_1, @1_Twitter, @_twitter_**
@Twitter+	same as **@Twitter** except allows a list separated by space ' ', comma ',' or ', ': **@twitter, @twitter1, @twitter_1, @TWITTER_1, @1_Twitter, @_twitter_**
#Twitter	hash/pound sign '#', followed by upper- and lowercase letters, numbers and underscores a-z A-Z 0-9 '_'; at least one: **#50isthenew30, #Python_Coding, #Quarantined, #GENESIS**
#Twitter+	same as #**Twitter** except allows a list separated by space ' ', comma ',' or ', ': **#50isthenew30, #Python_Coding, #Quarantined, #GENESIS**
Username	upper- or lowercase character, followed by mix of upper- and lowercase characters, numbers, hyphens and underscores a-z A-Z 0-9 '-' '_', at least one, up to 16: **Username, USERNAME, username, user_name, user-name, user5**
Website	optional http://, https:// or ftp:// followed by dotted website names with a-z A-Z 0-9 '-_', followed by optional port and optional path and parameters with alphanumerics and characters '-_/+=#%&.~?': **https://www.amazon.com/Come-Away-Me-Norah-Jones/dp/B00008WT49/ref=sr_1_5?dchild=1&keywords=sacd&qid=1588376982&refinements=p_n_binding_browse-bin%3A9536188011%7C9536192011&rnid=387643011&s=music&sr=1-5** known issues: no ip addresses, weak name logic, no logical enforcement beyond port#
Windows	Windows-1252 HTML5 text standard, includes ISO-8859-1 extended characters, ASCII control characters removed and additional characters in range 128-159 like ‘open / close quotes’ in MSWord, ligatures, graphemes, symbols like ™, en and em dashes, ligatures, ranges 32-126, 128, 130-140, 142, 145-156, 158-255: see https://en.wikipedia.org/wiki/Windows-1252 known issues: this does not include tab, vertical tab or newlines that may be inside of wrapping text fields (by design), a custom regex could be written if these are desired
year	yyyy 4 numbers in the range 1900-2099 **or use range(start year:end year) 1900, 1999, 2000, 2099**
yyyymmdd	years 1900-2099, hyphen '-' or forward slash '/', 1-12 or 01-12, '-' or '/', 1-31 or 01-31

Column Test Name	Description, Examples & Known Issues
	1925/01/01, 1925/1/1, 2005/12/31 known issues: does not validate days by month or leap year, only 1-31 see **range()** for date range support
zipcode+	5 digits followed by optional hyphen and 4 digits: **12345, 12345-0123** known issues: does not validate against inactive zip or +4 codes

Naming Collisions

There is a simple workaround if the Named Tests conflict with column values being tested. For example, consider the values 'alpha', 'beta', and so forth exist in an input column in the CSV file. A lookup list would work fine as they are just treated as values and not as regular expressions. If the preferred approach is an xcheck test (regex list) or standalone regular expression, specifying “alpha” as the test would result in the “alpha” Named Test to be used, which is not the desired result. To workaround this issue, simply use parenthesis (alpha) or (?:alpha) as the regex test. To test for '(alpha)', use the format \(alpha\). If name collisions are common, it is easy to go to the Named Tests dictionary in the profmod script and rename the tests to avoid collisions (e.g. nt.alpha) and then use that convention for named tests in the parameters.

Special Case Tests

The table below describes the built-in special case, hard coded tests that are listed in the Named Tests table. There are few instances they should be necessary unless another way cannot be accomplished or the xcheck option is being used. Here are known instances where they could help:

- An input column must always be empty, so the test is set to '**nothing**'.
- In a combination of an xcheck row of related tests, a column must be empty in order to pass the test (and blank entries are not allowed for xcheck lists), so for that column and row the test is set to '**nothing**'.
- In a combination of an xcheck row of related tests, a column must not be empty in order to pass the test and for simplicity sake for that column and row the test is set to '**something**' (or an appropriate named test that requires at least one character).
- In a combination of an xcheck row of related tests, a column is allowed to be anything, empty or not, and a placeholder is needed in that spot in the list (blank entries are not allowed for xcheck lists). For that column and row the test is set to '**anything**' so as to avoid the “at least one” issues found in most of the tests.

Test Name	Description
nothing	True if empty, useful to enforce blank only
something	True if not empty, useful to enforce non-blank only (logically likely unnecessary but

	helps future proof the code, and idiot proof the coder!)
anything	This is the default used by the software when no test is specified (returns True without inspecting the data)

Python String and Byte Method Tests

The table below lists tests using available True/False Python built-in string and bytes type validations. Referring to the documentation below where these methods are defined is highly recommended (scrolling down a bit is required).

https://docs.python.org/3/library/stdtypes.html#string-methods

https://docs.python.org/3/library/stdtypes.html#bytes-and-bytearray-operations

These Python built-in string validations are approximately 2.5x faster than regular expressions – they are probably hard coded in C language – and make a good choice for many applications. Note that none of these tests limit the string length (e.g. isdigit is not just a single digit, it can be a string of digits).

Test Name (Unicode)	Test Name (ASCII)	Description
isalnum	b.isalnum	Unicode test includes all characters from isalpha, isdecimal, isdigit and isnumeric; ASCII version is a-z, A-Z and 0-9 only
isalpha	b.isalpha	Unicode ‘Letter’ characters; ASCII version is a-z and A-Z only
isascii	b.isascii	Includes entire ASCII range from x00-x7F (not available prior to Python 3.7)
isdecimal		Unicode base 10 number forms specified in the Unicode Gen Cat 'Nd'; essentially various forms of 0-9
isdigit	b.isdigit	Unicode decimals, special digits, superscripts; ASCII version is 0-9 only
islower	b.islower	only lowercase letters, allows spaces and other punctuation
isnumeric	–	a variety of Unicode fractions, digit, decimal, numeric symbols
isprintable	–	includes all Unicode printable characters; excludes ‘other’ and ‘separator’ types other than ASCII space.
istitle	b.istitle	uppercase characters precede lowercase characters; spaces or punctuation can precede uppercase characters; 'Hello, World' is valid, 'Hello, world' is not
isupper	b.isupper	only uppercase letters, allows spaces and other punctuation

Python int() and float() Tests

These hard coded tests leverage the wide range of numeric formats supported by Python. The code is written to return pass or fail based on successful execution of the conversion. For a full description of the formats supported, refer to the links included with their description:

Test Name	Description, Examples & Known Issues
float	a hard coded test using the Python float() conversion function; returns True if it succeeds or False if it fails. **0e0, 1e5, 1.08E+9, -2.34e5, +1.0005e5, 3.14_15_93, inf, nan, -Infinity** see https://docs.python.org/3/library/functions.html#float and https://docs.python.org/3/reference/lexical_analysis.html#floating also see **notation** test and **range()** option
int	a hard coded test using the Python int() conversion function; returns True if it succeeds or False if it fails. The int() function is set with base=0, meaning that all base variations are legal as input (e.g. binary, octal, decimal and hexadecimal forms). **0, 3, +5, -999, 0b0000_1111, 0b01010101, 0o127, 0xdeadbeef** see https://docs.python.org/3/library/functions.html#int and https://docs.python.org/3/reference/lexical_analysis.html#integers also see **integer** test and **range()** option

The *range(start:end)* Column Test

Integers

`range(int:int)`

To test integer columns against a range of values, specify the from/to (start:end) inclusive values. For instance, range(0:10) allows all integers from 0 to 10 to succeed. Numbers outside the range, or numbers with decimal points (e.g. 5.25) will fail, even if they are within the numeric range as would any non-integer encountered (any value that fails the Python int() conversion).

Floating Point Numbers

`range(float:float)`

To test floating point columns against a range of values, specify the from/to (start:end) inclusive values with decimal points, even if they appear unnecessary. For instance, range(0.0:10.0) directs the function to use floating point, and values from the column are converted using the Python float() conversion and tested to see if they fall in the range from 0 to 10. In this case, integers or numbers with decimal points (e.g. 5 or 5.25) will succeed when they fall within the range. Any non-integer, non-floating point value encountered (any value that fails the Python float() conversion) will fail.

Dates

`range(dyyyymmdd:dyyyymmdd)`

To test a date column falling within a date range, specify the from/to (start:end) inclusive values starting with the constant 'd' followed by a four digit year, two digit month and two digit day. Any other format used in the parameter will fail and the program will terminate with an error.

The dates in the CSV file can be patterned according to one of the following formats:

yyyymmdd – with leading zeros only (must be 8 digits)

yyyy/mm/dd or yyyy-mm-mm – month and day can be single digits

mm/dd/yyyy or mm-dd-yyyy – month and day can be single digits

The range() function will attempt to convert the incoming column value to a date object if it meets one of the above patterns. If the value does not match one of the patterns or fails to convert to a date object, the column value will fail. This range option uses the Python datetime module which is aware of proper dates and leap years, so any invalid date will also fail. This is a good way to validate dates even if a range is unimportant (e.g. use **range(d10000101:d21000101)** for 1/1/1000-1/1/2100)

The *lookup_(name)* Column Test

Examples

```
lookup_states, lookup_codes, lookup_my-list
```

The lookup test allows a list of values to be stored in a lookup table from values listed in the User Data area of the spreadsheet. The values are not inspected other than whitespace being removed from either end to avoid mismatches from fumble-fingering in the spreadsheet. During the test, if the column value matches any lookup table value (in whole), the test succeeds.

Standard Usage

A typical lookup test would be coded within the parameter file like so, which is, in practicality, just a long regular expression with OR bars:

csvp_options	...	Sales Region	...
Column Test		lookup_states	
		⋮	
User Data		AK	
		AL	
		AR	
		AZ	
		⋮	

Enhanced Usage Options

When a lookup Column Test is specified, for instance with the name **lookup_states**, a number of things can occur, in this order:

- The Named Tests area is searched to see if it has already been defined (e.g. state codes in columns 10 and 25). If so, that test is used. If not, this definition is registered in Named Tests.

- The providers table is searched to see if it is handled by a user function (see section *User Functions and Lookup Providers*). If so, that function is used; this would be less common.
- The User Data area is evaluated to see if it contains anything:
  - If it is empty, the Column Test name (i.e. lookup_states) is used to retrieve the list contents from the file system (see the section *Using External Lookup Table Files* below).
  - If it contains another *lookup_(name)* option, this name (e.g. lookup_sales_region_states) is used to retrieve the list contents from the file system (see the section *Using External Lookup Table Files* below).
  - If it contains values as in the *Standard Usage* above, those values are used to load the list.

With this logic, when the same lookup name is encountered in a later column while processing the parameters, the software will ignore the User Data and simply reuse the internal lookup table already defined. When the report is generated at the end of the file run, if the column was marked as using the **Profile** option, two profile reports are output: one for the valid lookup table values encountered, and one for the values not found in the lookup table (see the **Profile** option above).

Using External Lookup Table Files

An alternative to using inline vertical lists in the User Data area of the parameter file is to use lists from a file. As described above, to specify an external file instead of an inline list, use either a lookup in the Column Test with a blank User Data value, or use a lookup in the first User Data row and the software will go through the following steps to find the lookup file on the file system:

- If the name was specified in the configuration file (e.g. `lookup_states = d:\lutabs\states.txt`) it will use that file.
- Otherwise, the suffix '.txt' will be added to the name (e.g. lookup_states.txt) and it will be loaded from the directory from which the parameter file was loaded.

The external file needs to have values one per line (stacked vertically), with no punctuation, like the User Data area of the *Standard Usage* example above.

Here is an example of how the parameter column might look to load lookup_states.txt:

csvp_options	…	Sales Region	…
Column Test		lookup_states	
		⋮	
User Data			

or:

csvp_options	…	Sales Region	…
Column Test		lookup_region	
		⋮	
User Data		lookup_states	

This option allows a library of standard lists to be maintained and used within different parameter files according to the needs of those files. As with the standard use of User Data with the lookup option, the Column Test name is saved for later reuse so there is no need to re-specify the input file.

The *regex_(name)* Column Test

Examples

```
regex_partnumbers, regex_AorB, regex_instead_of_a_lookup
```

This is a simple yet powerful option for a column test. As with the lookup option, the regex name is specified in the Column Test row and the regex itself is specified in the first row of the User Data area. Especially useful for very rigid column specifications which are often very easy to model. For instance, a column contains a coded field with the specific layout of AAnnnnnnnn - 2 uppercase alphas followed by 8 numeric digits. That is simply coded as '[A-Z]{2}[0-9]{8}' or '[A-Z]{2}\d{8}' .

Standard Usage

A typical regex test would be coded within the parameter file like so:

csvp_options	...	Channel	...
Column Test		regex_channel	
		⋮	
User Data		(online|offline)	

Enhanced Usage Options

When a regex Column Test is specified, for instance with the name **regex_channel**, a number of things can occur, in this order:

- The Named Tests table is searched to see if it has already been defined. If so, that test is used. If not, this definition is registered in Named Tests.
- The User Data area is evaluated to see if it contains anything:
  - If it is empty, the Column Test name (i.e. regex_channel) is used to retrieve the regex text from the file system (see the section *Using External Lookup Table Files* below).
  - If it contains another *regex_(name)* option, this name (e.g. regex_sales_channel) is used to retrieve the regex text from the file system (see the section *Using External Lookup Table Files* below).
  - If it contains a regex value as in the *Standard Usage* above, that regex is used.

With this logic, when the same regex name is encountered in a later column while processing the parameters, the software will ignore the User Data and simply reuse the regex already defined.

Using External Regular Expression Files

As with lookup tables, an alternative to using inline regular expressions in the first row of the User Data areas is to load the regular expression from a file. These files may have more than one record but in that case **must use the (?x) option** which allows line wrapping and ignores un-escaped whitespace outside of character classes. As described above, to specify an external file instead of an inline regex, use either a regex name in the Column Test with a blank User Data value, or use a regex name in the first User Data row and the software will go through the following steps to find the file on the file system:

- If the name was specified in the configuration file (e.g. `regex_channel = d:\re\channel.txt`), that file will be used.
- Otherwise, the suffix '.txt' will be added to the name (e.g. regex_channel.txt) and it will be loaded from the directory from which the parameter file was loaded.

Here is an example of how the parameter column might look to load regex_channel.txt :

csvp_options	...	Channel	...
Column Test		regex_channel	
		⋮	
User Data			

or:

csvp_options	...	Channel	...
Column Test		regex_sales_channel	
		⋮	
User Data		regex_channel	

This option allows a library of standard regular expressions to be maintained and used within different parameter files according to the needs of those files. As with the standard use of User Data with the lookup option, the Column Test name is saved for later reuse so there is no need to re-specify the regex or input file in the User Data in a later column using the same Column Test regex name.

Technical Notes

Keep in mind the following when writing regular expressions:

1. This software uses the standard Python “re” module.
2. Regular expressions can be tricky to get correct; always test extensively using small files.
3. The “fullmatch” method is used so there is no need to use start and end anchors (i.e. '^' and '$').
4. To avoid name collisions, use techniques such as (alpha), (?:alpha) and \(alpha\) to test for values without invoking named tests (or modify the Named Tests table to obfuscate the names).
5. (?a) turns on ASCII only matching for the pattern. It must be first in the string or can be used in a group. Most built-in regular expressions use this switch.

6. (?x) allows regular expressions to wrap to multiple lines. Break lines where spaces ARE NOT being tested and use carefully. Rather than attempting to use multiple lines in the parameter CSV file cell, loading expressions from an external file is recommended.
7. Use \x00 escape sequence for hexadecimal values and ranges (e.g. ASCII printable characters are in the range [\x20-\x7E]).
8. Use \unnnn escape sequence for Unicode values and ranges (e.g. UTF-8 characters à through å are in the range [\u00E0-\u00E5]).

For more information about how Python supports regular expressions, refer to these links:

https://docs.python.org/3/library/re.html

https://docs.python.org/3/howto/regex.html#regex-howto

The *xcheck_(name)* Column Test

Examples

```
xcheck_assembly, xcheck_brand_to_model, xcheck_make_to_manuf
```

The xcheck test allows performing multiple tests against multiple columns as a group of related tests. To establish this relationship, each column participating in the xcheck test group must have the same xcheck *name* in the Column Test row. This creates an association between all of those columns and those columns pass or fail together as a group for each input record. This option requires at least 2 participating columns and has no reasonable limit on how many can participate. Multiple xcheck groups can be used in the same parameter set, each group just needs to use unique names.

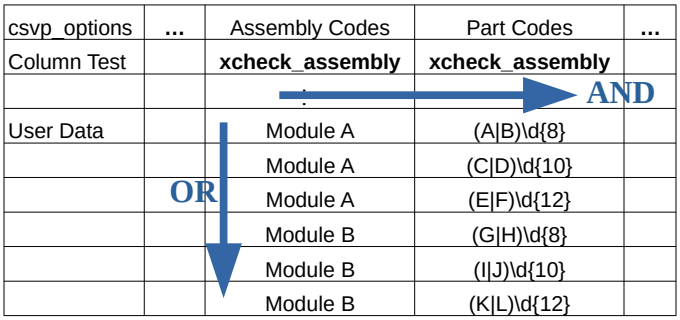
Standard Usage

The way the xcheck works is as follows:

1. Each row of the User Data area is read into internal tables for each column and compared to one another to ensure they are the same length. If they are not, the program will terminate with an error.
2. As each record of the CSV file is read, the values for each xcheck column are saved until the last participating member of the xcheck group is reached. At that point, the validation begins for that input record against the saved vales.
3. The xcheck test works left to right through the first row of tests. If they all succeed, the test succeeds. This is an AND condition. E.g. if column A True AND column B True AND column C True… then all of the fields are marked as “passed” and validation moves on to the next column unrelated to the xcheck group.
4. If any of the tests in the first row fails, xcheck moves on to the next row of tests. This is an OR condition. If all in row 1 True OR all in row 2 True OR all in row 3 True…

5. This proceeds until either a successful test row is reached or the end of the test rows are reached, in which case the test fails.

Take a very simple example. The records in a file have 2 coded columns, one for assemblies consisting of multiple parts, and part codes for the parts themselves. An "xcheck_assembly" is specified and both columns participate in the test:



In the above case, the examples could have been shortened to two rows by adding the regex strings together with OR bars (e.g. ((A|B)\d{8})|((C|D)\d{10})|((E|F)\d{12})). For demonstration purposes this case was presented because when more than 2 columns participate in an xcheck, it is very possible that there will be a need to repeat values in at least one of the columns (e.g. for automobiles, the 3 columns might be Make, Model and Trim, where Make would need to be repeated for every Model, but trim could probably be an OR'd regex).

The follow values are allowed as values in the User Data:

Text strings: Text string values such as "Module A" above are saved as regular expressions, as is, as simple pattern matches.

Named Tests: As with the Column Tests row, any built-in Named Test is available, including previously defined regular expressions (*regex_(name)*), and lookup tables (*lookup_(name)*). Keep in mind that "previously defined" refers to previously defined in a lower column number (column to the left).

range(): Integer, floating point and date ranges can all be used.

***lookup_(name)*:** The Named Tests table is searched to see if the lookup was previously defined, and if so, reused. If it is not found, the software will load the file from the file system as described above. That means that if this lookup was not previously defined, it will HAVE to be loaded from the file system in order to work. Keep in mind that this lookup test is only in effect for this position in the xcheck "grid" as a sort of nested OR condition, not unlike a long regular expression (e.g. in lieu of lookup_states, you used (AL|AK|AZ|AR|CA|CO…)). Also, since this is not a normal Column Test, profiling is not performed on matching and non-matching entries.

Regular expressions & *regex_(name)*: They can be single line regular expressions or a *regex_(name)* moniker. If a *regex_(name)* is used, the Named Tests table is searched for reuse. If it is not found, the software will load the file from the file system as described above.

Since xchecks can consume more processing time than typical Column Tests, try to place the most likely matches in the top-most rows in the parameter set or xcheck external file.

Be especially careful in xcheck test definitions with Named Test spellings as misspellings will just be assumed to be a simple string and converted to a regex. If there may be an issue, use the **verbose=True** config file switch and look for "regex-looking" names showing up in the list with values having the same simple pattern rather than an internally defined test.

For example, the Named Test 'decimal' looks like this in the Named Tests dictionary dump:

```
      Abbrev : <bound method RegexType.field_test of re.compile('(?a)[a-zA-Z]+\\.', re.ASCII)>
     decimal : <bound method RegexType.field_test of re.compile('(?a)[-+(]?(?:(?:\\.\\d+)|...>
        Name : <bound method RegexType.field_test of re.compile('(?a)[-a-zA-Z0-9\\\'\\" \\...>
```

But the string 'decimals' pattern would look like this:

```
range(20:100) : <bound method RangeType.field_test of range(20:100) -> 20-100>
     decimals : <bound method RegexType.field_test of re.compile('decimals')>
            A : <bound method RegexType.field_test of re.compile('A')>
```

Enhanced Usage Options

When an xcheck Column Test is specified, for instance with the name **xcheck_assembly**, a number of things can occur, in this order:

- The xcheck objects have their own dictionary, and this is checked to see if the name already exists. If it does, this column will participate in the xcheck group. Otherwise, this column starts a new xcheck group.
- The User Data area is evaluated to see what it contains:
  - If it contains another *xcheck_(name)*, this name (e.g. xcheck_assem_codes) is used to retrieve the test list contents from a text file on the file system (see the section *Using External xcheck Files* below). When the contents of the file are loaded, the test list is processed and lookup and regex tests are pulled from the Named Tests table or file system as needed.
  - If it contains an *xcheck_(name)[i]* with an index number (e.g. xcheck_assem_codes[0]) this name is used to retrieve the test list contents from a column of a CSV file on the file system (see the section *Using External xcheck Files* below). When the contents of the file are loaded, the test list is processed and lookup and regex tests are pulled from the Named Tests table or file system as needed.
  - If it contains a list of test values as in the *Standard Usage* above, those values are used to load the test list, pulling lookup and regex tests from the Named Tests table or file system as needed.

Using External xcheck Files

An alternative to using inline test lists in the User Data areas of each column is to load the test lists from a file. Unlike the lookup and regex tests, to use this option the User Data area must be used since the Column Test xcheck name will be the same for all participating columns and the ability to specify different sets of tests for each column is needed. The xcheck test list can be pulled from an external file in one of two ways:

xcheck text files: As with lookup and regex, an xcheck_name value can be placed in the User Data area which instructs the software to look for a text file and load the file from the system.

xcheck CSV files with index (e.g. xcheck_assembly[0]): Due to the complexity of xcheck parameters, it is easiest to work with them all side-by-side until satisfied they are correct. To support this, when an index value is specified as part of the xcheck name in the User Data area, it instructs the software to assume the input is a CSV file. The index, which is zero based (i.e. column 0 is the first column), determines which column of the CSV file to use to populate the test list for the column. The CSV file must be comma delimited and use double quotes for quoting fields. Using this option, the above example could look like this in the parameter file:

csvp_options	...	Assembly Codes	Part Codes	...
Column Test		xcheck_assembly	xcheck_assembly	
		⋮		
User Data		xcheck_assembly[0]	xcheck_assembly[1]	

The external file loading option does not alleviate the need to have a balanced number of tests for each participating column.

The software will go through the following steps to find the file on the file system:

- If the xcheck name was specified without an index (e.g. xcheck_assem_codes), the system will look for a text file:
  - If the name was specified in the configuration file (e.g. `xcheck_assem_codes = d:\xc\assem.txt`) that file will be used.
  - Otherwise, the suffix '.txt' will be added to the name (e.g. xcheck_assem_codes.txt) and it will be loaded from the directory from which the parameter file was loaded.
- If the xcheck name was specified with an index (e.g. xcheck_assembly[0]), the system will look for a CSV file:
  - If the name was specified in the configuration file (e.g. `xcheck_assembly = d:\xc\assem.csv`) that file will be used.
  - Otherwise, the suffix '.csv' will be added to the name (e.g. xcheck_assembly.csv) and it will be loaded from the directory from which the parameter file was loaded.
  - In either case, the data will be loaded from the column indicated by the index number.

This option allows a library of standard xcheck test list files to be maintained and used within different parameter files, or multiple times within the same parameter file, according to the needs of those files.

Report File

The report file is straightforward and self-explanatory. It reports on the filenames read from the configuration file, including user defined files and user function imports, provides status during the housekeeping portions of the script, provides internal dictionary dumps if verbose = True in the config file, and produces the following report sections:

Testing Grand Totals:

- Total Fields
- Total Test Errors
- Total Blank Errors
- Total Length Errors
- Total Max Length Errors

Statistics for each column, determined by parameters:

- Passed
- Failed
- Blank
- Blank Errors
- Length Errors
- Max Length Errors
- Column Profile - Unique Values and Occurrence Counters
- Lookup Failures - Unique Values and Occurrence Counters
- Unique Profile - Duplicate Values and Occurrence Locations (record sequence numbers)
- Statistical Profile - Statistical Analytics and Occurrence Locations of Dropped Values

Detailed Statistics for each xcheck row:

- Generated for every xcheck test defined, a group of rows with the group of columns and statistics, repeating, such as:
  - Row #1:
    - (Column Number) Column Name → Test
      - Passed
      - Failed
    - (Column Number) Column Name → Test

        - Passed
        - Failed
    - Row #2:
        - (Column Number) Column Name → Test
            - Passed
            - Failed
        - (Column Number) Column Name → Test
            - Passed
            - Failed
    - etc.

Record Processing Statistics:

- Total Records Read
- Total Bad Records
- Total Errors Written
- Total CSV Records Written
- Total Log Records Written
- Total Processing Time (in seconds)
- Processing Time / record (in milliseconds)

When the program is finished running, if any columns were reported in error, the program will terminate with a return code of 1. If there were no errors, the return code is 0.

- When csvprofiler.py is called from another program, it will now "return" the return code.
- When csvprofiler.py is run as the main script, it will now use sys.exit(*return code*).

Error CSV File

The error csv file is simply the input file echoed back to output with a new column 1 added. This new column includes a space separated list of the column numbers that were found to be in error for that record. Note that if a given column has exceeded its limit for reporting errors, if other column limit parameters allow printing the record, column 1 still contains all error column numbers for the record.

Bad Records

If bad CSV records are encountered during processing (too many or too few columns), the record will be output if the configuration file output limit hasn't been exceeded. Column 1 will contain a failure message with the input record number and the row will be added as columns 2-n.

Error Log File

The error log file is a tab separated text file, which allows it to be loaded into a spreadsheet for easier viewing or programmatically processed. The tabbed columns are the following:

1. **Key Field** – As specified in the configuration file. Either data from the specified column number or the input CSV record count (excluding header, if any).
2. **Column Name** – From the input header / parameter file for the column in error.
3. **Column Test** – The test name used for that column.
4. **Length and Maximum Length** – From the parameter file, in parenthesis **(0:0)**.
5. **Error Conditions** – The conditions that caused this column to be reported in error, in parenthesis, space separated, e.g. **(col max)**:

   col – represent Column Test error

   blk – represents blank test error

   len – represents length error

   max – represents maximum length error
6. **Column Data** – Data from the input CSV file column that failed the stated Column Test

Bad Records

If bad CSV records are encountered during processing (too many or too few columns), the record will be output if the configuration file output limit hasn't been exceeded. The failure message with the input record number, followed by the input CSV record will be output without any tabs.

Extending the Software Capabilities

Modifying profmod named_tests with User Test Functions

The named tests can be found in the profmod module, in the dictionary called "named_tests". They are placed in specific order to influence the csvpcg script when it automatically chooses a best fit test; the tie goes to the first test in the list with the greatest number of True matches. The tests can be rearranged, removed, renamed and new tests added using any name and regex value desired in any location in the dictionary; just follow the format of the existing regex examples and be careful with proper formatting (like trailing comma). A comment exists at the bottom of the dictionary that can be used as a template for adding a new regex test.

In addition to modifying the regular expressions, it is easy to add another type of test using a function. For example, look at the "anything_type_func" function. It just accepts a parameter called "field" and returns True or False (in this case always True). Simply create a function that performs the test desired

on “field” and return the appropriate boolean. When the function is coded, add a relevant name to the dictionary as the key and the name of the function (without parenthesis) as the value. The function is picked-up when used in the parameter file and stored with the “FieldTest” class object, executed later during the field test. This approach includes the ability to execute functions in other modules. Add the appropriate import statement and add the external function to the named_tests dictionary as described.

User Functions & Lookup Providers

The software is also designed to support user defined Python functions without modification of the CSV Profiler modules. These functions might add custom logic for complex field testing or used to provide database hooks for look-ups. There are two methods designed into the software to achieve this functionality, both using the *lookup_name* test option as the entry point to the provided functions:

Wrapper Module: Using this approach, a Python wrapper module is developed by the user that defines the function(s) and calls the csvprofiler main function to kickoff the validation run.

Imported Module: Using this approach, the software runs as usual but the user-provided Python modules and functions are defined in the configuration file.

Wrapper Module

This approach enables development of a Python wrapper script to provide new testing functions by taking over a “lookup” test with a “provider” function. This allows customized testing logic to be devised without changing the core modules. Its best use might be for interfacing with external data sources to assist with validation, such as an RDBMS or NoSQL database.

To use a wrapper module, simply develop the function code required and then add that function to profmod using the “add_provider” method, then call csvprofiler with the config file. See code sample below that works for both the Wrapper Module and the Imported Module approaches.

This feature can be used for single column tests or for multiple column tests similar in practice to the “xcheck” feature approach:

- lookup_1: save field, return True (or None)
- lookup_2: save field, return True (or None)
- lookup_3: save field and execute logic, return True or False

The only difference between this and xcheck is that xcheck marks all participating columns as False, but the result is essentially the same – the error gets reported.

Imported Module

This approach enables development of a Python script for import by profmod and used as an external test function provider. To enable this feature, for example to take over the lookup_states definition, specify the module and function using the following syntax in the configuration file:

```
lookup_states = import wrapdemo check_states
```

where wrapdemo is the module / script name and check_states is the function name. With this configuration file entry, when the software sees the lookup_states test defined in the parameters and pulls the definition from the configuration file, it will do the following:

- Use the Python importlib.import_module function to load the wrapdemo module. It will look locally so the module must be situated in the same folder as the profmod module.
- It will execute the module's init() method if it has one. Note that if the same module is used for more than one function, init() may be called multiple times so that needs to be accounted for when developing the init() code.

Wrapper & Imported Module Code Sample

Here is an example of how to use either a wrapper script or an imported module.

- When run as a wrapper, the "main" function is executed, calls init(), then uses add_provider to allow profmod to be aware of the function. Then it passes the configuration file name from the command line to run csvprofiler. When the lookup_states test is executed in profmod, it will call back to the check_states(field) function.
- When run as an imported module, the module is imported and the init() function is called by profmod when the lookup_states parameter is processed, and calls the check_states(field) function when the lookup_states test is executed.

During the init() function, this code creates an in-memory SQLite database and states table and loads it with state abbreviations. That table is searched for a matching item using the check_states function.

```
#!/usr/bin/env python3

import sys
import sqlite3
import csvprofiler as cp
import profmod as pm

states = ['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA',
          'HI', 'IA', 'ID', 'IL', 'IN', 'KS', 'KY', 'LA', 'MA', 'MD', 'ME',
          'MI', 'MN', 'MO', 'MS', 'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM',
          'NV', 'NY', 'OH', 'OK', 'OR', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX',
          'UT', 'VT', 'VA', 'WA', 'WV', 'WI', 'WY']

def init():

    global conn, c

    conn = sqlite3.connect(':memory:')
    c = conn.cursor()
    c.execute('''CREATE TABLE states (state text)''')
    for i in range(len(states)):
```

```
        c.execute(f"INSERT INTO states VALUES ('{states[i]}')")
    conn.commit()


def check_states(field):
    fld = (field, )
    c.execute('SELECT count(*) FROM states WHERE state=?', fld)
    if c.fetchone()[0] == 1:
        return True
    return False


def main():
    init()
    pm.add_provider('lookup_states', check_states)
    cp.main(sys.argv[1])
    conn.close()


if __name__ == '__main__':
    main()
```

Using the profmod Module in Another Application

The profmod module was built to manage the various test options allowing for different driver applications such as the csvprofiler for CSV files. While it keeps track of counters for failure and success along with methods for pulling and clearing them at a record level, it can be used to establish any number of test fields very easily and used independently in another Python application. For simple usage examples, see the main() function at the bottom of the module.

Coding Examples & Usage

The following code runs the 'Sentence' test against the text 'Hello World'. `f1` becomes the class instance holding the test definition with the 'field_test' method which runs the test, returning True, False or None (for blank field or some xcheck fields, see below for xcheck test results stipulations).

```
import profmod as pm
a = 'Hello World'
f1 = pm.FieldTest(1,'Desc','Sentence',length=0,maxlength=0,blankiserror=True,strip=False)
print(f1.field_test(a))
>>> True
```

FieldTest Class Parameters:

1. unique field number (int)
2. field name

3. test name – the same as Column Test described above
4. field length – 0 means no test is performed
5. max length – 0 means no test is performed
6. blankiserror – as described above, except use True/False Python boolean
7. strip – is strip surrounding spaces as described above, except use True/False Python boolean
8. aux – (not shown) is a list object:
    - a single value list for *regex_(name)* option (or single string object), or
    - a multi-value list for *lookup_*(*name)* and *xcheck_*(*name)* options, or
    - a name for loading from the file system (see documentation above for lookup, regex and xcheck).

Note: To support loading files from the file system, there is an empty dictionary in profmod called "config" that can be populated with key value pairs (i.e. key=lookup_states, value=d:\lu\states.txt). It is also possible to populate it with a value for 'param_file' to give it a file path in which to look. If so, it must contain a path and filename, even if the filename is bogus, so that the path can be extracted. If none is provided, the software will look in the current path '.\'.

Test Results Tuple

Detailed results about the tests performed are stored in 4 dictionaries containing True/False values for each field number: field error, blank error, length error, and max length error. To access the information and clear them back to False (for record level processing), use the following:

```
mytuple = pm.get_all_flags()
```

This will return a tuple of 4 lists containing field numbers for those fields that failed in the 4 categories, in order, and reset all flags back to False. For instance, if both fields 1 and 2 failed their respective field tests, and field 2 also failed the max length test, the tuple would contain ([1, 2], [], [], [2]). If there were no errors since the last time the instruction was run, the tuple will contain ([], [], [], []). To access the flags directly, access the dictionaries:

- error_flag_dict
- blank_flag_dict
- length_flag_dict
- maxlen_flag_dict

xcheck Test Results

The actual xcheck testing is performed on the last column (field) registered with the test, which then updates the field error flags for all of the participating columns (True or False, as a group). All columns leading up to the last simply save the field value and return None, since the test cannot be performed until all of the field values have been obtained. But there is an exception. If a max length error was encountered during any of these field tests, the result will come back False for that field test.

Additional Module Resources

stats counter dictionary

The stats counter dictionary keys and values are:

- stats[0]['Total Fields'] → FieldTest classes defined
- stats[0]['Total Test Errors'] → for all fields
- stats[0]['Total Blank Errors'] → for all fields
- stats[0]['Total Length Errors'] → for all fields
- stats[0]['Total Max Length Errors'] → for all fields
- stats[fieldnum]['Passed'] → count of fields that passed the field test for fieldnum (int)
- stats[fieldnum]['Failed'] → count of fields that failed the field test for fieldnum (int)
- stats[fieldnum]['Blank'] → count of fields that were blank for fieldnum (int)
- stats[fieldnum]['Length Error'] → count of fields that failed the length test for fieldnum (int)
- stats[fieldnum]['Max Length Error'] → count of fields that failed the maximum length test for fieldnum (int)

There are a few methods that can assist with access:

- **get_stats()** – provides the entire dictionary, just the high level counter dictionary with fieldnum=0, or an individual counter dictionary using the fieldnum= parameter for the desired field
- **print(show_formatted_dict(get_stats()))** – will display a human-readable representation of the stats dictionary
- **report_grand_totals(rm)** – where rm is record manager (which could easily be just the function **print**). The csvprofiler uses this function.
- for reporting individual column statistics, the csvprofiler uses the following code:

```
for k in sorted(pm.ftclass_dict.keys()):
    pm.ftclass_dict[k].report_totals(rm.write)
```

```
for k in sorted(pm.xcheck_objects_dict.keys()):
    pm.xcheck_objects_dict[k].report_totals(rm.write)
```

Viewing All Dictionaries

Use the following instructions to show **repr**esentation of internal dictionaries for debugging:

```
print(pm.show_all_dicts())
print(pm.show_formatted_dict(pm.show_globals()))
```

Helpful File Encoding References:

https://en.wikipedia.org/wiki/ASCII

https://en.wikipedia.org/wiki/ISO/IEC_8859-1

https://en.wikipedia.org/wiki/Windows-1252

http://unicode.org/charts/

https://www.charset.org/utf-8

http://www.fileformat.info/info/unicode/category/index.htm

GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. <https://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be

designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer

review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is

included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the

present version, but may differ in detail to address new problems or concerns. See https://www.gnu.org/licenses/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  YEAR  YOUR NAME.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3 or
any later version published by the Free Software Foundation; with no
Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A
copy of the license is included in the section entitled "GNU Free
Documentation License".
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with … Texts." line with this:

```
with the Invariant Sections being LIST THEIR TITLES, with the Front-
Cover Texts being LIST, and with the Back-Cover Texts being LIST.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.