V5_Testing Scripts

Joshua Berger, Jack Tillman, Gabe Sauvageau

Jul 10, 2019

CONTENTS:

CHAPTER

ONE

POST PROCESSING

1.1 Parser

1.1.1 Constants

optimized_parsing.**SIDES = {'black': 'BLK', 'red': 'RED'}**
Dictionary containing names of the sides, as well as their corresponding strings for table entry

Type dict

optimized_parsing.**MESSAGE_TAGS = ['STRESS_NG', 'IPMITOOL', 'STREAM_C', 'DD_TEST', 'HDPARM',**
List containing tags of the tests used during the current test run. Set manually.

Type list of str

optimized_parsing.**PREFIX_COLS = ['time_stamp', 'unit', 'side']**
List containing names of the prefix columns

Type list of str

1.1.2 Test

class optimized_parsing.**Test**(*json_file_name=None*)
An object which represents all of the information regarding a single test.

Parameters **json_file_name** (*str*) – The filename of the json file from which test information should be loaded.

tag
The tag associated with the test (also used as the name of the test, and the name of the table containing the test data).
Type str

re
The regular expression used for parsing log messages from the test.
Type re.Pattern

key_tag
The name of the variable by which test data is separated. Not used in all tests. An example is sensor for IPMI_TOOL, since every sensor has the same data fields, and thus are stored in separate rows.
Type str

data_info
A dictionary with keys corresponding to the names of the data columns in the test's table, and values corresponding to the type of each column.
Type dict

num_lines
The number of lines of log messages that contain data for a single test result.
Type int

num_extra_lines
The number of additional lines of log messages that *may* contain data for a single test result, often in a separate format.

Set to **0** if not present in the test's information file.
Type int

extra_re
The regular expression used for recognizing the data in extra lines. Only used by some tests.
Type `re.Pattern`

extra_data
The list containing the names of the data present in extra lines. Only used by some_tests.
Type list of str

1.1.3 Log

class optimized_parsing.**Log**(*filename*, *units*)
An object associated with a log messages file.

Contains details about what units and tests are expected to have results in the log.

Parameters

- **filename** (*str*) – The name of the log file.
- **units** (*list of str*) – The list containing the units with results in the log.

message_file_name
The name of the log messages file.
Type str

log_file
The log file itself.
Type file

units
The list of units associated with this log.
Type list of str

tests
A dictionary containing the name (tags) of tests associated with this log as keys, with the values as the corresponding *Test* objects.
Type dict

line_count
The number of lines in the log file.
Type float

cur_line_num
The current line number.
Type float

re_prefix
The regular expression associated with the prefixes of the messages in this log.
Type re.Match

add_test(*test*)
Associate a new test with this log.

Attempts to set up a new table in the database for storing data from this test. After it has done so, or if such a table already exists, this method adds the new test to the test dictionary *tests*, where the key is the string equivalent to this test's tag (*Test.tag*).
Parameters **test** (*Test*) – The *Test* to be added to this log.

Note: May raise errors when attempting to create a new table, as there is no available way to specify the length of an SQL string type.

---

extract_data()
Extracts test results from the log file.

For each line in the log file, checks if it has a relevant *prefix*. If it does, then it calls *parse_line()* on the line to attempt to convert data from the line into a row to be inserted into the database.

When it recieves the list of rows to be inserted, it then attempts to insert them into the corresponding test table in the database, using a query string generated by *insert_ignore_many_query()*, in order to ensure that duplicate entries are not inserted.

get_line_count()
Return the number of lines in the log file.
Returns The number of lines in the file with the name attr:*message_file_name*.
Return type float
Raises **IOError** – The file with the name attr:*message_file_name* could not be opened.

parse_line(*line*, *cur_test*, *recur_count=0*)
Converts data from a log message into a row to insert into an SQL table.

Given a log message line and the test the line is associated with, attempts to interpret the data in the line into a row dictionary.

The exact interpretation process is dependant on the individual test, but in general it attemps to match the line to a regular expression corresponding to the current test's output format, and then uses the groups found by the regular expression, along with the variables specified in the test's *data_info* attribute to populate a row dictionary with the data.

For tests that do (or may) have multiple lines of data for a single test result (and thus a single row of data), will also check and attempt to parse the following lines as specified by the attributes of cur_test.

After successfully filling a dictionary for a full row of data for the current test, will call itself recursively on the next line in the log file, under the assumption that the next line also belongs to the same test. This continues until it either reaches the end of the lines associated with the current test, or it exceeds python's recursion limit. This allows it to return a list of rows to be inserted into a single table with one query, which is much faster than inserting each row individually.
Parameters
- **line** (*str*) – A string containing a single line from the log file, which presumable contains a log message generated by the test denoted in cur_test.

- **cur_test** (*Test*) – The *Test* that is expected to correspond to the test results contained in line.
- **recur_count** (*int*) – Stores current recursion depth, to avoid raising an error if the limit is reached. Set to `0` by default if no value is specified.

Returns A list of *row* dictionaries. Every row in the list is associated with the same test.

Return type list

set_re_prefix()
Set up the regular expression for the *prefix*.

The *prefix* is the part of the log message that contains the time stamp, the unit and side, and the test tag. The regular expression for matching the *prefix* is as such dependent on which units were tested and which tests were performed during the test run that the log file contains results from. The *prefix* makes it possible to determine if a log message came from a test or not, thus quickly ruling out some irrelevant log messages, and making it possible to determine which test the log messsage may have come from.

show_progress()
Print the current progress in parsing the log file.

1.1.4 Other Functions

optimized_parsing.**make_datetime**(*match_obj*)
Given a match object that contains groups with date/time info, return a corresponding python datetime object.

Parameters **match_obj** (`re.Match`) – A match object which contains time stamp information, and corresponding groups.

Returns A datetime object containing the time stamp information in `match_obj`.

Return type `datetime`

optimized_parsing.**return_or**(*val_list*)
Return the contents of *val_list* in as a regex *or* statement.

Parameters **val_list** (*list*) – A list of values `[v1,...,vn]`

Returns The string `"((v1)|...|(vn))"`, which is formatted appropriately for use in an *or* statement in a regular expression

Return type str

Example

```
>>> print(return_or(['tea','milk','coffee']))
"((tea)|(milk)|(coffee))"
```

optimized_parsing.**rowvals4SQLquery**(*row, col_info, pre_cols=['time_stamp', 'unit', 'side']*)
Generates a string representation of the values in a row.

Given a row dictionary, and a dictionary containing column info for the correspoding test, returns a string representation of the corresponding values in rows for every column of the table.

Parameters

- **row** (*dictionary*) – A dictionary containing the data for a single row of an SQL table. The keys are the names of columns, with the values as the corresponding values.

- **col_info** (*dicitionary*) – A dictionary containing the columns unique to a specific test as keys, with the corresponding values as the data types of the columns.
- **pre_cols** (*list*) – A list containing the names of columns which are not contained in `col_info`.

  This parameter is set to *PREFIX_COLS* by default, since the columns for *time_stamp*, *unit*, and *side* are those which are not unique to any single test, and thus are not contained within `col_info`.

Returns A string representing all the column data in `row` (with all data formatted as appropriate, and empty columns rendered as "NULL"), with data separated by commas and enclosed in parentheses. Suitable for use in an SQL *INSERT* query.

Return type str

optimized_parsing.**rowvals4SQLmany**(*row_list*, *col_info*)

Generates a string representation of the values in many rows.

Given a list of row dicitonaries, and dictionary containing column info for the corresponding test, returns a string representation of the row data.

Parameters

- **row_list** (*list*) – A list containing the rows (as dictionaries) to be inserted.

  Assumes every row dictionary contains corresponding values for every data column in the table.
- **col_info** (*dictionary*) – A dictionary containing the columns unique to a specific test as keys, with the corresponding values as the data types of the columns.

Returns A string representing the data in all of the rows in `row_list`, with each row's data string separated by commas. Suitable for use in an SQL *INSERT* query for many rows of data.

Return type str

optimized_parsing.**rowcols4SQLquery**(*cols, pre_cols=['time_stamp', 'unit', 'side']*)

Generates a string representation column names for a table.

The output is formatted for use in an SQL query (where the values of `cols` are the column names of the table).

Parameters

- **cols** (*list*) – A list of strings representing the names of the columns unique to a specific test table.
- **pre_cols** (*list*) – A list containing the names of columns which are not contained in the row dictionary.

  This parameter is set to *PREFIX_COLS* by default, since the columns for *time_stamp*, *unit*, and *side* are those which would not usually be contained in the row dicitionary, as they are not unique to a specific test table, and thus their values are extracted separately from those corresponding to the columns in `cols`.

Returns A string containing comma separated values from `pre_cols` and `cols`, enclosed in parentheses. Suitable for use when naming the columns in an SQL *INSERT* query.

Return type str

optimized_parsing.**insert_ignore_many_query**(*test*, *rows*)

Returns a string for an SQL *INSERT IGNORE* query for multiple rows of data.

Given a *Test* object and a list of rows, generate an SQL query string for inserting all of the rows in `rows` into the table corresponding to `test`.

Parameters

- **test** (*Test*) – The *Test* that corresponds to the data contained in `rows`.
- **rows** (*list*) – A list of row dictionaries to be inserted into the table corresponding to `test`.

Returns A string that, when executed as an SQL query, will perform an *INSERT IGNORE* of the rows in `rows` into the table corresponding to `test`

Return type str

Note: Because this code was created for internal use (and ideally automated use), it makes no attempt at data sanitization, meaning it could, in theory, allow a malicious actor with arbitrary access to this code to perform an SQL injection attack. However, if they only have access to the log file, it is unlikely that they would be able to embed arbitrary SQL queries that would still be recognized by the regular expressions that parse the line (assuming they cannot create arbitrary tests)

1.2 Results Viewer

1.2.1 Constants

`results_viewer.`**`MESSAGE_TAGS = {1: 'STRESS_NG', 2: 'IPMITOOL', 3: 'STREAM_C', 4: 'DD_TES`**
Dictionary containing shorthand numerical ids and the corresponding test names for each.

Type dict

`results_viewer.`**`SIDES = {'BLK': 'black', 'RED': 'red'}`**
Dictionary containing key-names for the sides in the database, as well as their corresponding full names/colors.

Type dict

`results_viewer.`**`TEMPERATURE_OPS = ['intake', 'outtake', 'ambient']`**
List of valid temperature keys (aka options).

Type list of str

`results_viewer.`**`DB = <Database(mysql://guest:*****@localhost/test_results)>`**
Database containing results from all tests.

Type `Database`

`results_viewer.`**`TEST_DATA_GRAPHS = {'DD_TEST': ['SDA'], 'FIBER_FPGA_TEMP': ['2.5', '3.5'],`**
Dictionary containing the names of tests to be auto-graphed, and the corresponding list of options to be graphed for each test.

Type dict

1.2.2 Selection Functions

`results_viewer.`**`test_run_select`**`()`
Allows the user to select test run that they wish to generate graphs for.

If the user enters 'o' or 'options', returns the list of test runs stored in the *TESTS* table of *DB*, along with their start and end times.

results_viewer.**test_select**(*start_time*, *end_time*)
Allows the user the select the test for which they wish to view data.

If the user enters 'o' or 'options', returns the list of available tests, along with a numerical shorthand they can be entered as.

Has additional options. One is to run `autogenerate_graphs()` by entering 's' or 'save', allowing the user to automatically generate a preset selection of graphs for their selected test run. The other allows the user to generate a graph of temperature vs. time, by entering 't'

Parameters

- **start_time** (datetime) – The time at which the current test run began.
- **end_time** (datetime) – The time at which the current test run ended.

results_viewer.**graph_select**(*test_key*, *start_time*, *end_time*)
Allows the user the select the test option for which they wish to generate a graph.

If the user enters 'o' or 'options', returns the list of valid options for the current test.

Parameters

- **test_key** (*Test*) – The *Test* which the user has currently selected.
- **start_time** (datetime) – The time at which the current test run began.
- **end_time** (datetime) – The time at which the current test run ended.

1.2.3 Data Retrieval and Graphing Functions

results_viewer.**get_data**(*cur_test*, *sel_opt*, *start_time*, *end_time*, *where_addendum=''*)
Retrieve test data from the database.

Get the data associated with the selected test run, from current test's database, limited to only the data specified by the selected option.

Parameters

- **cur_test** (*Test*) – The *Test* for which we wish to obtain data.
- **sel_opt** (*str*) – The user selected option denoting the subset of data we care about.
- **start_time** (datetime) – The time at which the currently selected test run began.
- **end_time** (datetime) – The time at which the currently selected test run ended.
- **where_addendum** (*str, optional*) – An optional string which is used to specify additional constraints for the query to the database. Empty by default, and only used by some tests.

Returns

- **x** (*dict*) – A dictionary containing a list of the desired x-axis data associated with each side
- **y** (*dict*) – A dictionary containing a list of the desired y-axis data associated with each side

results_viewer.**determine_units**(*cur_test*, *sel_opt*)
Returns to appropriate units for the selected test and option.

Parameters

- **cur_test** (*Test*) – The *Test* for which we wish to obtain data.
- **sel_opt** (*str*) – The user selected option denoting the subset of test data that we care about.

Returns **units** – The units of the data in the selected subset.

Return type str

results_viewer.**map_time2temp**(*times*, *data*)

Maps a list of time stamps and a list data to a list of temperatures and a list of data at the corresponding times.

Parameters

- **times** (list of datetime) – The list of time stamps to find corresponding temperature values for.
- **data** (*list*) – List of values, where the index of each value corresponds to an index in times. Included to ensure that the number of data points in the x and y axes are not misaligned.

Returns

- **temps_list** (*list of float*) – List of the temperature values corresponding to the time stamps in times.
- **temp_data** (*list*) – List of values, where the index of each value corresponds to an index in temps_list.

results_viewer.**generate_graph**(*cur_test*, *sel_opt*, *start_time*, *end_time*, *max_y=None*, *where_addendum=''*, *sub_title=''*, *save_to_file=False*, *graph_type='plot'*)

Performs the steps necessary to generate a graph for the selected subset of data.

Parameters

- **cur_test** (*Test*) – The *Test* for which we wish to obtain data.
- **sel_opt** (*str*) – The user selected option denoting the subset of test data that we care about.
- **start_time** (datetime) – The time at which the currently selected test run began.
- **end_time** (datetime) – The time at which the currently selected test run ended.
- **max_y** (*int, optional*) – The value to use for the max y value on the graphs.
- **where_addendum** (*str, optional*) – An optional string which is used to specify additional constraints for the query to the database. Empty by default, and only used by some tests.
- **sub_title** (*str, optional*) – The subtitle to append to the figure title. By default an empty string.
- **save_to_file** (*bool*) – A boolean telling the function if the figures should be saved to a file instead of displayed on screen. False by default.
- **graph_type** (*str*) – The type of graph to generate. Set to 'plot' by default. Valid options are 'plot' to generate a normal plot, or 'hist' to generate a histogram.

results_viewer.**gen_IPERF_graph**(*sel_opt*, *start_time*, *end_time*, *stf=False*)

Variant function for generating a graph of IPERF values, calls generate_graph() with additional parameters specific to IPERF

Parameters

- **sel_opt** (*str*) – The user selected option denoting the subset of test data that we care about.
- **start_time** (datetime) – The time at which the currently selected test run began.

- **end_time** (datetime) – The time at which the currently selected test run ended.
- **stf** (*bool*) – A boolean telling the function if the figures should be saved to a file instead of displayed on screen. False by default.

results_viewer.**gen_temp_graph**(*start_time*, *end_time*)
Generate a graph of the temperature data vs. time.

Parameters

- **start_time** (datetime) – The time at which the currently selected test run began.
- **end_time** (datetime) – The time at which the currently selected test run ended.

results_viewer.**autogenerate_graphs**(*start_time*, *end_time*)
Generate a set of predetermined graphs and save them as pngs.

The graphs are determined by the constant *TEST_DATA_GRAPHS*

Parameters

- **start_time** (datetime) – The time at which the currently selected test run began.
- **end_time** (datetime) – The time at which the currently selected test run ended.

1.2.4 Graph Helper Functions

results_viewer.**one_and_three_fig**(*upperx*, *uppery*, *upper_info*, *lowerx*, *lowery*, *lower_info*, *fig_info*)
Generate a figure with one large upper graph of data vs time, and three smaller lower graphs of data vs temperature.

Parameters

- **upperx** (*dict*) – Dictionary containing lists of x-axis data that correspond to each side in *SIDES*.
- **uppery** (*dict*) – Dictionary containing lists of y-axis data that correspond to each side in *SIDES*.
- **upper_info** (*dict*) – Dictionary contain information used when generating the upper graph, such as the graph format, the y limit, and various labels and titles.
- **lowerx** (*dict*) – Dictionary containing dictionaries that correspond to each side in *SIDES*, which in turn contain lists of x-axis data that correspond to each of three keys.
- **lowery** (*dict*) – Dictionary containing dictionaries that correspond to each side in *SIDES*, which in turn contain lists of y-axis data that correspond to each of three keys.
- **lower_info** (*dict*) – Dictionary contain information used when generating the lower graph, such as the graph format, the y limit, and various labels and titles.
- **fig_info** (*dict*) – Dictionary containing more general info that applies the the figure as a whole.

results_viewer.**tricolor_fig**(*x*, *y*, *info*, *fig_info*)
Generate a figure with one large graph, with data graphed in three different colors.

That is, three colors per side, for a total of six colors. Each color corresponds to one side and key.

Parameters

- **x** (*dict*) – Dictionary containing dictionaries that correspond to each side in *SIDES*, which in turn contain lists of x-axis data that correspond to each of three keys.

- **y** (*dict*) – Dictionary containing dictionaries that correspond to each side in *SIDES*, which in turn contain lists of y-axis data that correspond to each of three keys.
- **info** (*dict*) – Dictionary contain information used when generating the graph, such as the graph format, the y limit, and various labels and titles.
- **fig_info** (*dict*) – Dictionary containing more general info that applies the the figure as a whole.

results_viewer.**my_plotter**(*ax*, *xdata*, *ydata*, *fmt*, *param_dict*)

Plot xdata and ydata.

Parameters

- **ax** (axis) – The axis on which to plot the data.
- **xdata** (*list*) – The list of data for the x-axis.
- **ydata** (*list*) – The list of data for the y-axis.
- **fmt** (*str*) – The format string for the plot.
- **param_dict** (*dict*) – Additional parameters.

1.2.5 Test

class results_viewer.**Test**(*json_file_name*)

An object which represents all of the information regarding a single test.

Parameters **json_file_name** (*str*) – The filename of the json file from which test information should be loaded.

tag

The tag associated with the test (also used as the name of the test, and the name of the table containing the test data).

Type str

data_key

The name of the column which contains the names of the different subsets of data the user can choose from when generating a graph. Use is mutually exclusive with *key_list*.

Type str

data_info

A dictionary with keys corresponding to the names of the data columns in the test's table, and values corresponding to the type of each column.

Type dict

data_tag

The name of the column in which the desired data is stored, not used by all tests.

Type str

key_list

A list of the names of different subsets of data the user can choose from when generating a graph. Use is mutually exclusive with *data_key*.

Type list of str

units_key

The name of the column which contains the units of the data stored in the current row of the table. Not present for all tests.

Type str

units_dict
A dictionary matching the options from *key_list* with the appropriate units. Present when *units_key* is not.
Type dict

1.2.6 Other Functions

results_viewer.**return_options**(*options_list*, *sep_char*)
Return the options listed in options_list.

Parameters

- **options_list** (*list of str*) – The list of the options to return.
- **sep_char** (*str*) – The character to use when separating the options.

Returns A string listing the available options, with each option separated by the character defined by sep_char.

Return type str

results_viewer.**return_help**()
Return the standard help message.

Returns The standard help message, listing valid user inputs.

Return type str

1.3 Temperature Data Shifter

1.3.1 Constants

temp_data_shifter.**RESULTS_DB = <Database(mysql://guest:*****@localhost/test_results)>**
Database containing results from all tests.

Type Database

temp_data_shifter.**TEMPS_DB = <Database(sqlite:///../Results/DAQTEMPS.db)>**
Database containing temperature data collected during the test run.

Type Database

1.3.2 Important Functions

temp_data_shifter.**temp_shifter**()
Shifts temperature data from *TEMPS_DB* to *RESULTS_DB*.

Takes in temperature data from *TEMPS_DB*, rounds their time stamps to the nearest ten seconds, formats them as rows split by side, and inserts the rows into the *TEMPERATURES* table in *RESULTS_DB*

Assumes:

- Both databases exist as defined
- The row in *TESTS* with the time interval containing the data to be shifted exists
- The tables in both databases exist and are formatted as expected

temp_data_shifter.**convert_row**(*row*)
Converts a row from *TEMPS_DB* to two rows for *RESULTS_DB*.

Takes in a single row of temperature data from *TEMPS_DB* and converts it into two rows of temperature data to be inserted into *TEMPERATURES* in *RESULTS_DB*.

Parameters **row** (*dict*) – A dictionary representing the row retrieved from *TEMPS_DB*. Assumes the row was retrieved successfully.

Returns **out_rows** – A list containing the two newly generated rows.

Return type list of dict

1.3.3 Other Functions

temp_data_shifter.**round_time**(*tm_str*)
Generates a `datetime.datetime` object containing a rounded version of a time stamp string.

Parameters **tm_str** (*str*) – A string containing a representation of a `datetime.datetime` object in standard format.

Standard format is defined as `"YYYY-MM-DD hh:mm:ss[:msecs]"`

Returns A `datetime.datetime` object, containing the timestamp from `tm_str` rounded to the nearest ten second mark

Return type `datetime.datetime`

temp_data_shifter.**rowvals4SQLquery**(*row*)
Generates a string representation of the values in `row`, formatted for an SQL query.

Parameters **row** (*dict*) – The dictionary representing the row. Assumes `row` contains corresponding values for every data column in the table

Returns A string containing comma separated values from `row`, enclosed in parentheses

Return type str

temp_data_shifter.**rowvals4SQLmany**(*row_list*)
Generates a string representation of the values in many rows, formatted for an SQL query.

Parameters **row_list** (*list of dict*) – A list containing the row dictionaries to be inserted. Assumes every row dictionary contains corresponding values for every data column in the table.

Returns A string containing strings generated by `rowwvals4SQLquery()` for each row in `row_list`, separated by commas.

Return type str

temp_data_shifter.**insert_ignore_many_query**(*table*, *rows*)
Generate an appropriate string for inserting potentially duplicate rows into an SQL table.

Parameters

- **table** (*str*) – The name of an SQL table in the current database. Assumes a table with that name exists in *RESULTS_DB*.
- **rows** (*list of dict*) – A list of rows to insert into `table`. Assumes every row dictionary contains corresponding values for every data column in the table.

Returns A string representing an SQL statement that, if executed, performs an *INSERT IGNORE* of every row in `rows` into `table`.

Return type str

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX