managealooma Documentation

Release 0.1.0

Jay Rosenthal

Oct 12, 2019

Contents

CHAPTER 1

About Manage Alooma

`managealooma` is a wrapper for the Alooma ETL tool API that makes it easy to programmatically manage the tool and provides functions for transforming events.

- Docs: http://managealooma.readthedocs.org/
- Source: https://github.com/hoverinc/managealooma
- Download: http://pypi.python.org/pypi/managealooma

CHAPTER 2

About Alooma

Alooma is an ETL tool for building out datawarehouses. Alooma does the basic tasks of connecting to sources for you such as replicating a database, receiving a webhook or connecting to a cloud app API. Alooma also inserts your data to a datawarehouse such as Snowflake, Redshift, or Bigquery. In between those steps Alooma allows you to transform your data any way you want using real Python code. This package contains may functions to make programmatic management of Alooma easy!

CHAPTER 3

Contents

3.1 License

3.2 Help

To report bugs or to ask for help please email jay.rosenthal@hover.to

3.3 Getting Started

This document will take you through setting up a local repo, testing your transformations locally, using the transformation functions in the managealooma package, and using one of management classes in the managealooma package.

3.3.1 Installation

This package is available from the Python Package Index. If you have pip you should be able to do:

```
$ pip install managealooma
```

3.3.2 Local Configuration

1. Create a workspace for your Alooma code

```
$ mkdir -p alooma/code
$ touch alooma/event_sample.py alooma/event_test.py alooma/manage.py alooma/code/__
↪init__.py alooma/code/code_engine.py
```

2. Check that your files match this directory structure:

```
alooma/
    alooma_code/
        __init__.py
        code_engine.py
    event_sample.py
    event_test.py
    manage.py
```

3.3.3 Test an Event

1. Each Alooma event is a dictionary. Alooma takes your data and adds *_metadata* fields that help you manage the data. Add the following sample event to `event_sample.py`:

```
event = {"myField": "stuff",
         "createdAt": "2019-08-01 00:00:00",
         "xmin::text::bigint": 123456789,
         "aNumber": 6,
         "id": 1,
         "aDictionary": {"one": "Some Stuff",
                         "two": "More Stuff",
                         "three": {"Another dict": {"Inside Three": 3.3,
                                                    "More Inside Three": 3.4}
                                   },
                         },
         "_metadata": {
             "@uuid": "1a1a1a-2b2b-3c3c-4d4d-5e5e5e5e5e",
             "event_type": "my_schema.my_table",
             "input_label": "my_input",
             "@parent_uuid": ""
         }
         }
```

2. Each of these samples will run through the code you have in `code_engine.py` file. Let's add this code to `code_engine.py` to change the keys of the sample event from camelCase to snake_case.

```
from managealooma import convert_all_event_fields_to_snake_case

def transform(event):
```

(continues on next page)

(continued from previous page)

```
    # Converts the all keys of the event dictionary to snake_case
    event = convert_all_event_fields_to_snake_case(event)

    return event
```

3. Now we need code that will run the event in `event_sample.py` through the code in `code_engine.py`. Add the code below to `event_test.py`

```
# The TransformationTest class takes an event and run it through your transformation␣
↪code
from managealooma import TransformationTest

# Imports your local transformation code. It will also import any submodules in the␣
↪directory.
import code.code_engine as ce

# This imports the sample event for testing.
from event_sample import event

# Instantiate the TransformationTest class and test a single event
T = TransformationTest(api=None, code_package=ce, preview_full_events=True, preview_
↪difference_dicts=False, local_or_api='local')
T.test_single_event(sample=event)
```

4. Now run the file and the event will print to your console before and after the transformation

```
$ python event_test.py
```

5. You can change the params in TransformationTest to summarize the diffs in the event instead of viewing the entire before and after

```
T = TransformationTest(api=None, code_package=ce, preview_full_events=False, preview_
↪difference_dicts=True, local_or_api='local')
```

6. You can add as many transformations as you want to `code_engine.py`. Let's add more transformations from *managealooma.transformation_functions*. Change your `code_engine.py` to this:

```
from managealooma import convert_all_event_fields_to_snake_case, add_composite_key,␣
↪map_value_in_list_to_dictionary_key, flatten_json

def transform(event):
    # Converts the all keys of the event dictionary to snake_case
    event = convert_all_event_fields_to_snake_case(event)

    # More transformations. Read the docs to see all the options or write your own!
    event = flatten_json(event, field_list='a_dictionary', levels=1, keep_
↪original=False, dump_to_string=True)
    event = add_composite_key(event, field_list=['id', 'created_at'], key_name='id_
↪created_datetime')
    event = map_value_in_list_to_dictionary_key(event, mapping_dict_with_lists={'one_
↪to_three': [1, 2, 3], 'four_to_six': [4, 5, 6]}, existing_column='a_number', new_
↪column='number_category', allow_nulls=True, passthrough=True)

    return event
```

7. Run that file 1 more time to see the transformations

```
$ python event_test.py
```

3.3.4 Use the API

Alooma has a robust API that will let you programmatically manage the tool. The *managealooma* package contains main functions to help utilize these features. Typical usage will often mean running the code with *apply_changes=False* to visually inspect your changes first. Then you can set *apply_changes=True* to execute your adjustments. This example will walk you through changing the mapping mode for an event.

In this example we'll only edit the `manage.py` file. 1. Add imports for *alooma*, *os*, and the mappings class from *managealooma*

```
import alooma
from managealooma import Mappings
from os import environ
```

2. Add your credentials for Alooma and instantiate the *api*.

```
alooma_credentials = {'account_name': environ.get('ALOOMA_ACCOUNT_NAME'),
                      'api_key': environ.get('ALOOMA_API_KEY')}

api = alooma.Client(api_key=alooma_credentials["api_key"], account_name=alooma_
↪credentials["account_name"])
```

3. Add code to instantiate the *Mapping* class and use the *change_mapping_mode* function. Change *MY_EVENT.NAME* to the name of one of your events.

```
M = Mappings(api=api, event_name='MY_EVENT.NAME', preview_full=True, preview_
↪changes=False, apply_changes=False, pprint_indent=2, pprint_width=250, pprint_
↪depth=5)
M.change_mapping_mode(new_mapping_mode='STRICT')
```

4. Now your file should be complete like this:

```
import alooma
from managealooma import Mappings
from os import environ

alooma_credentials = {'account_name': environ.get('ALOOMA_ACCOUNT_NAME'),
                      'api_key': environ.get('ALOOMA_API_KEY')}

api = alooma.Client(api_key=alooma_credentials["api_key"], account_name=alooma_
↪credentials["account_name"])

M = Mappings(api=api, event_name='MY_EVENT.NAME', preview_full=True, preview_
↪changes=False, apply_changes=False, pprint_indent=2, pprint_width=250, pprint_
↪depth=5)
M.change_mapping_mode(new_mapping_mode='STRICT')
```

5. Run the file and you'll see the full mapping printed with before and after changes.

```
$ python manage.py
```

6. It's a little hard to see the mapping moode in the mapping dictionaries. Let's change the print parameters to only see the changes.

```
M = Mappings(api=api, event_name='MY_EVENT.NAME', preview_full=False, preview_
↪changes=True, apply_changes=False, pprint_indent=2, pprint_width=250, pprint_
↪depth=5)
```

7. Now make sure to change the mapping mode to a new value

```
# The mapping code can be AUTO_MAP, STRICT, or FLEXIBLE. Try changing the value to a␣
↪different value than the current setting
M.change_mapping_mode(new_mapping_mode='STRICT')
```

8. Finally, if you want to execute these changes then sett *apply_changes=True*

```
M = Mappings(api=api, event_name='MY_EVENT.NAME', preview_full=False, preview_
↪changes=True, apply_changes=True, pprint_indent=2, pprint_width=250, pprint_depth=5)
```

3.4 Column DDL

Functions to generate DDL queries to change columns directly in the datawarehouse.

`ColumnDDL.`**`__init__`**(*tuple_or_tuple_list*, *change_type='data_type'*, *has_log=False*, *case='UPPER'*)

Prints DDL for direct DB changes of data_type, drop, or add

Parameters

- **`tuple_or_tuple_list`** – A single tuple or list of tuples with information about the changes to make
- **`change_type`** – The type of DDL to create out of data_type, add, rename, combine or drop
- **`log_table`** – True if the event has a log table
- **`case`** – UPPER or LOWER case for the DDL statements

For a change_type of data_type or adding a column the tuple_or_tuple_list should contain the schema.table, the column name, and the column data type.

```
('schema_name.table_name', 'column_one', 'INT')
or
[('schema_name.table_name', 'column_one', 'INT'),
('schema_name.table_name', 'column_two', 'VARCHAR(16777216)')]
```

For a change_type of dropping a column a column the tuple_or_tuple_list should contain the schema.table and the column name

```
('schema_name.table_name', 'column_one')
or
[('schema_name.table_name', 'column_one'),
('schema_name.table_name', 'column_two')]
```

For a change_type of renaming a column a column the tuple_or_tuple_list should contain the schema.table, the current column name, and the new column name

```
('schema_name.table_name', 'my_bad_column', 'my_good_column', 'INT')
or
[('schema_name.table_name', 'my_bad_column_one', 'my_good_column_one', 'INT'),
('schema_name.table_name', 'my_bad_column_two', 'my_good_column_two',
↪'VARCHAR(16777216)')]
```

(continues on next page)

(continued from previous page)

```
```

For a change_type of combining columns the tuple_or_tuple_list should contain the schema.table, the current column name, the new column name, and the column data type.

```
('schema_name.table_name', 'my_bad_column', 'my_good_column', 'INT')
or
[('schema_name.table_name', 'my_bad_column_name_one', 'my_good_column_name_one',
↪'INTEGER'),
('schema_name.table_name', 'my_bad_column_name_two', 'my_good_column_name_two',
↪'VARCHAR(16777216)')]
```

3.4.1 Convert Tuple to List

`ColumnDDL.`**`convert_tuple_to_list`**`()`

Converts single tuples into a list of tuples so list iteration on single event changes works

Parameters **`tuple_or_tuple_list`** – A tuple or a tuple list

Returns A list of tuple

3.4.2 Create DDL Statements

`ColumnDDL.`**`create_ddl_statements`**`()`

Creates DDL statements, prints the statements, and returns the statements

Parameters

- **`event_column_type_tuple`** – The tuple from the function specified
- **`change_type`** – The type of column change statements to print. Options are column, drop, combine_drop
- **`log_table`** – True if the columns to alter have log tables

Returns A string with all the queries generated

3.4.3 Add Column

`ColumnDDL.`**`add_column`**`(`*schema_name*, *table_name*, *column_to_add*, *new_column_type*`)`

Prints add column statements

Parameters

- **`schema_name`** – The name of the schema
- **`table_name`** – The name of the table
- **`column_to_add`** – The name of the column to add

Returns The query

For a change_type of data_type or adding a column the tuple_or_tuple_list should contain the schema.table, the column name, and the column data type.

```
('schema_name.table_name', 'column_one', 'INT')
or
[('schema_name.table_name', 'column_one', 'INT'),
('schema_name.table_name', 'column_two', 'VARCHAR(16777216)')]
```

3.4.4 Change Column Data Type

`ColumnDDL.`**`change_column_data_type`**(*table_name*, *schema_name*, *column_name*, *new_column_type*, *new_column_type_no_count*)

Prints the DB DDL statement for a single column

Parameters

- **`schema_name`** – The name of the schema
- **`table_name`** – The name of the table
- **`log_table`** – True if the table has a log table

Returns The query

For a change_type of data_type or adding a column the tuple_or_tuple_list should contain the schema.table, the column name, and the column data type.

```
('schema_name.table_name', 'column_one', 'INT')
or
[('schema_name.table_name', 'column_one', 'INT'),
('schema_name.table_name', 'column_two', 'VARCHAR(16777216)')]
```

3.4.5 Combine Columns

`ColumnDDL.`**`combine_columns`**(*schema_name*, *table_name*, *column_to_drop*, *column_to_keep*, *column_to_keep_type*)

Prints statement to combine data from 2 columns into a target column and drop the non-target column

Parameters

- **`fully_qualified_table_name_db`** – The schema.table_name of the table to change
- **`column`** – The tuple for an individual column
- **`log_table`** – True if the table has a log table

Returns The query

For a change_type of combining columns the tuple_or_tuple_list should contain the schema.table, the current column name, the new column name, and the column data type.

```
('schema_name.table_name', 'my_bad_column', 'my_good_column', 'INT')
or
[('schema_name.table_name', 'my_bad_column_name_one', 'my_good_column_name_one',
↪'INTEGER'),
('schema_name.table_name', 'my_bad_column_name_two', 'my_good_column_name_two',
↪'VARCHAR(16777216)')]
```

3.4.6 Drop Column

`ColumnDDL.`**`drop_column`**`(`*schema_name*, *table_name*, *column_to_drop*`)`

Prints drop column statements

Parameters

- **`schema_name`** – The name of the schema
- **`table_name`** – The name of the table
- **`column_to_drop`** – The name of the column to drop

Returns the query

For a change_type of dropping a column a column the tuple_or_tuple_list should contain the schema.table and the column name

```
('schema_name.table_name', 'column_one')
or
[('schema_name.table_name', 'column_one'),
('schema_name.table_name', 'column_two')]
```

3.4.7 Rename Column

`ColumnDDL.`**`rename_column`**`(`*table_name*, *schema_name*, *existing_column_name*, *new_column_name*`)`

Prints the DB DDL statement to rename a single column

Parameters

- **`schema_name`** – The name of the schema
- **`table_name`** – The name of the table
- **`existing_column_name`** – The existing column name of the colum
- **`new_column_name`** – The new column name for the column

Returns The query

For a change_type of renaming a column a column the tuple_or_tuple_list should contain the schema.table, the current column name, and the new column name

```
('schema_name.table_name', 'my_bad_column', 'my_good_column', 'INT')
or
[('schema_name.table_name', 'my_bad_column_one', 'my_good_column_one', 'INT'),
('schema_name.table_name', 'my_bad_column_two', 'my_good_column_two',
↪'VARCHAR(16777216)')]
```

3.5 Consolidations

Consolidations take the new data that's been inserted into the temporary _log table and combine it into the main table. The terms consolidation query and scheduled queries can be used interchangeably.

`Consolidations.`**`__init__`**`(`*api*, *preview_changes=True*, *apply_changes=False*, *pprint_indent=2*, *pprint_width=250*, *pprint_depth=5*`)`

Consolidations are Alooma's way of making sure only the most recent row for each table appears in the main table.
Consolidation management the v2 API. The class is initiated with the following variables:

Parameters

- **`api`** – The Alooma API client authentication
- **`preview_changes`** – Prints the consolidation or consolidations changes if True
- **`apply_changes`** – Executes the consolidation changes if True
- **`pprint_indent`** – The indent value to pprint dictionaries
- **`pprint_width`** – The width value to pprint dictionaries
- **`pprint_depth`** – The depth value to pprint dictionaries

3.5.1 Get Schedule Queries

`Consolidations.`**`get_scheduled_queries`**`()`

Gets the list of scheduled queries from Alooma

Returns The JSON response with all of the scheduled queries

3.5.2 Scheduled Query Table

`Consolidations.`**`scheduled_query_table`**`()`

Retrieves a list of the consolidation queries running in Alooma. Prints the information in a dataframe if the class preview_changes flag is True.

Returns A dataframe with the information on the scheduled queries

3.5.3 Create Consolidation

`Consolidations.`**`create_consolidation`**`(`*event_name*, *cron_schedule='*/15 * * * *'*, *custom_variables=False*`)`

Create Consolidation Given Configuration

Parameters

- **`event_name`** – The event type for which to create the consolidation
- **`cron_schedule`** – The cron schedule on which to run the schedule query. Default is 15 minutes.
- **`custom_variables`** – custom variables to add to consolidation. For consolidations from database tables set custom variables to None. For consolidations for all other tables the recommended customer variable settings are custom_variables = {"all_ri_fields": ["_metadata.@timestamp"], "ri_field": "_metadata.@timestamp", "ri_column": "_metadata.@timestamp"}

3.5.4 Get Scheduled Query For Event

`Consolidations.`**`get_scheduled_query_for_event`**`(`*event_name*`)`

Gets the scheduled query for a specific event

Parameters **`event_name`** – The name of the event for which to retrieve the schedule query

Returns Returns the schedule query

3.5.5 View Schedule Query For Event

`Consolidations.`**`view_schedule_query_for_event`**(*event_name*, *show_configuration_only=True*)

Prints the scheduled query for an event when

Parameters

- **`event_name`** – The event to view a scheduled query for
- **`show_configuration_only`** – Print the configuration only without the query

Returns The scheduled query

3.5.6 Remove Schedule Query

`Consolidations.`**`remove_scheduled_query`**(*schquery*)

Removes the scheduled query for a specific ID

Parameters **`schquery`** – the ID of a schedule query to remove

Returns None

3.5.7 Remove Scheduled Query For Event

`Consolidations.`**`remove_scheduled_query_for_event`**(*event_name*, *show_configuration_only=True*)

Removes the scheuled query for an event

Parameters

- **`event_name`** – The name of the event for which to remove the consolidation
- **`print_consolidations`** – print the current consolidation
- **`show_configuration_only`** – Print the configuration only without the query
- **`apply_change`** – apply the remove consolidation

Returns

3.5.8 Change Scheduled Query Frequency For Event

`Consolidations.`**`change_scheduled_query_frequency_for_event`**(*event_name*, *cron_schedule='*/15 * * * *'*)

Change the frequency of the scheduled query

Parameters

- **`event_name`** – he name of the event for which to remove the consolidation
- **`cron_schedule`** – The new cron schedule to set for the event

Returns None

3.6 Events

Each event equates to a mapping between a dictionary and a single target table in the data warehouse. This class retrieves lists of events and information about events.

`Events.`**`__init__`**`(`*api*`)`

Gets information from Alooma on events and prints detailed information about events.

Parameters **`api`** – api authentication using the Alooma package

3.6.1 Print Sorted List

`static` `Events.`**`print_sorted_list`**`(`*lst*`)`

Parameters **`lst`** – list Takes a list and prints a sorted list

3.6.2 Get All Events

`Events.`**`get_all_events`**`()`

Gets the information for all events from Alooma

Returns A list of dictionaries with information for each event

3.6.3 View Events

`Events.`**`view_events`**`(`*single_input=None*, *single_event=None*, *print_format='table'*, *table_limit=None*, *pprint_indent=2*, *pprint_width=250*, *pprint_depth=5*`)`

Prints a data from with the event info

Parameters

- **`single_input`** – The name of a specific input to filter the results
- **`single_event`** – The name of a specific event to filter the results
- **`print_format`** – Specify 'table' to print event info as tables or 'json' to print as dictionaries
- **`table_limit`** – Limit the number of events printed in the dataframe
- **`pprint_indent`** – The indent value to pprint dictionaries
- **`pprint_width`** – The width value to pprint dictionaries
- **`pprint_depth`** – The depth value to pprint dictionaries

Returns The event data in the print_format of a dataframe or a list of dictionaries

3.6.4 List Events

`Events.`**`list_events`**`(`*input_labels='all'*, *print_lst=False*, *add_quotes_commas=False*`)`

Prints and/or returns a list of event names

Parameters

- **`input`** – string 'all' for all events or specify an input label for a specific events
- **`print_lst`** – boolean True to print the list to the console

- **add_quotes_commas** – boolean True to print the list to the console with single quotes around strings and commas after each item

Returns list of events by name

3.6.5 Delete Event

`Events.`**`delete_event`**`(`*event_name*`)`

Deletes an event from the mapper and prints the event_name and API response

Parameters **event_name** – The name of the event to delete

Returns None

3.7 Inputs

Inputs are the sources which Alooma pulls data from. Examples of inputs are webhooks, databases, and cloud applications.

`Inputs.`**`__init__`**`(`*api, preview_full=True, preview_changes=True, apply_changes=False, pprint_indent=2, pprint_width=250, pprint_depth=5*`)`

View, created and edit Alooma inputs. Inputs are the source of data to bring into Alooma. The class is initiated with the following variables:

Parameters

- **api** – The Alooma API client authentication
- **preview_full** – Prints the input and changed input if True
- **preview_changes** – Prints the changes in the input by category if True
- **apply_changes** – Executes the mapping changes if True
- **pprint_indent** – The indent value to pprint dictionaries
- **pprint_width** – The width value to pprint dictionaries
- **pprint_depth** – The depth value to pprint dictionaries

3.7.1 Print Sorted List

`static` `Inputs.`**`print_sorted_list`**`(`*lst*`)`

Parameters **lst** – list Takes a list and prints a sorted list

3.7.2 Get All Inputs

`Inputs.`**`get_all_inputs`**`()`

Gets the information for all inputs from Alooma

Returns A list of dictionaries with information for each input

3.7.3 Get Input

`Inputs.`**`get_input`**`(`*input_name*`)`

Gets the input information for a specific input

Parameters **`input_name`** – The name of the input to retrieve

Returns A dictionary with the input

3.7.4 View Inputs

`Inputs.`**`view_inputs`**`(`*single_input=None*, *print_format='table'*`)`

Prints a data from with the event info

Parameters

- **`single_input`** – string The name of a specific input to filter the results
- **`print_format`** – string 'table' to print event info as tables or 'json' to print as dictionaries

Returns The list of input dictionaries

3.7.5 Delete Input

`Inputs.`**`delete_input`**`(`*input_name*`)`

Delete an input. This can not be undone!

Parameters **`input_name`** – The name of the input to delete

Returns None

3.7.6 Pause Input

`Inputs.`**`pause_input`**`(`*input_name*`)`

Pause the input from pulling data

Parameters **`input_name`** – The name of the input to pause

Returns None

3.7.7 Resume Input

`Inputs.`**`resume_input`**`(`*input_name*`)`

Resume the input so it will pull date

Parameters **`input_name`** – The name of the input to resume pulling data

Returns None

3.7.8 List Inputs

`Inputs.`**`list_inputs`**`(`*add_quotes_commas=False*`)`

Prints and/or returns a list of input names

Parameters

- **`print_lst`** – boolean True to print the list to the console

- **add_quotes_commas** – boolean True to print the list to the console with single quotes around strings and commas between each name

Returns list of the names of inputs

3.7.9 List Tables

Inputs.**list_tables**(*input_name*)

Prints and/or returns the list of tables in an input

Parameters **single_input** – string The input for which to print or return the tables

Returns The list if tables in an input

3.7.10 Create Input Database

Inputs.**create_input_database**(*source_credentials*, *new_input_name*, *existing_input*, *tables_dict=None*, *auto_map=True*, *input_default_schema=None*, *replication_type='incremental_load'*, *batch_size=100000*)

Create a new input with a database as the source

Parameters

- **source_credentials** – The database credentials and configuration for the source
- **new_input_name** – The name of the new input
- **existing_input** – The name of the existing input from which to copy the configuration
- **tables_dict** – A dictionary containing the tables to replicate and the update indicator such as {"table_name": "xmin::text::bigint"}
- **auto_map** – True if the input should be auto-mapped
- **input_default_schema** – The default schema for the data in the target database
- **replication_type** – The type of replication to apply
- **batch_size** – The number of rows to pull in each batch

Returns None

Example database source credentials:

```
{'server': 'server_name',
 'schema': 'schema_name',
 'port': 'port',
 'database': 'database',
 'db_type': 'psql',
 'user': 'username',
 'password': 'password'}
```

3.7.11 Edit Input Configuration

Inputs.**edit_input_configuration**(*input_name=None*, *field_to_edit=None*, *new_field_value=None*)

Edit a single field in an input configuration at a time

Parameters

- **input_name** – The name of the input to edit
- **field_to_edit** – The name of the field to edit
- **new_field_value** – The new value for the field

Returns None

3.7.12 Add Table to Input

`Inputs.`**`add_table_to_input`**(*input_name*, *new_table_dict={}*)

Add tables to an existing input

Parameters

- **input_name** – The name of the input to add tables to
- **new_table_dict** – A dictionary with the tables to add and their update field such as {“table_name”: “xmin::text::bigint”}

Returns None

3.7.13 Change Auto Mapping Mode

`Inputs.`**`change_auto_mapping_mode`**(*input_name=None*, *new_mapping_mode=False*)

Edit a single field in an input configuration at a time

Parameters

- **input_name** – The name of the input to alter
- **new_mapping_mode** – The new automapping mapping mode to set of True or False

Returns None

3.7.14 Add Template to Parameter Configuration

`Inputs.`**`add_template_to_parameter_configuration`**(*input_name*, *add_to_parameter*, *template*)

Add a template to single parameter or a list of parameters in an input configuration

Parameters

- **input_name** – The input to add template to
- **add_to_parameter** – a single parameter or a list of parameters that you want to add template to
- **template** – e.g. ‘%Y-%m-%d’

Returns None

3.7.15 Edit Parameter Configuration

`Inputs.`**`edit_parameter_configuration`**(*input_name*, *parameter_to_edit*, *value_to_set*, *new_value*)

edit API parameter value, e.g. changing days past value from 100 to 10

Parameters

- **input_name** – The input to make the change for
- **parameter_to_edit** – a single parameter that you want to edit
- **value_to_set** – name of the field you want to edit
- **new_value** – new value you want to set

Returns None

3.7.16 Preview Input Changes

Inputs.**preview_input_changes**(*input*, *new_input*, *show_matching*, *show_changed*, *show_removed*, *show_added*)

Takes an original mapping and altered mapping and prints various views on the changes

Parameters

- **input** – A dictionary representing the current state of the mapping
- **new_input** – An altered dictionary representing the changed state of the mapping
- **show_matching** – Show the key value pairs that match between the two mappings
- **show_changed** – Show the key value pairs that have been changed between the two mappings
- **show_removed** – Show the key value pairs that were removed from the current state of the mapping
- **show_added** – Show the key value paids that were added to the changed state of the mapping

Returns None

3.7.17 Apply Input Changes

Inputs.**apply_input_changes**(*input*, *print_message*)

Apply the input changes in the Alooma API

Parameters

- **input** – The new input to set
- **print_message** – A message to print after a successful change

Returns None

3.8 Mappings

Mappings are the conversion of a dictionary to a flattened table structure. The mapping class allows you to perform operations to view and alter the settings for a single event type.

Mappings.**__init__**(*api*, *event_name*, *preview_full=True*, *preview_changes=True*, *apply_changes=False*, *pprint_indent=2*, *pprint_width=250*, *pprint_depth=5*)

View and change Alooma mappings. Mappings are the conversion of a dictionary to a flattened table structure. The class is initiated with the following variables:

Parameters

- **`api`** – The Alooma API client authentication
- **`event_name`** – The name of the event to view or change settings for
- **`preview_full`** – Prints the mapping or mapping changes if True. The default is True.
- **`preview_changes`** – Prints the changes in the mapping by category if True
- **`apply_changes`** – Executes the mapping changes if True
- **`pprint_indent`** – The indent value to pprint dictionaries
- **`pprint_width`** – The width value to pprint dictionaries
- **`pprint_depth`** – The depth value to pprint dictionaries

3.8.1 Get Mapping For Event

`managealooma.mappings.Mappings.`**`get_mapping_for_event`**`()`

Gets the mapping for the event name that the class was initialized with

Returns a dictionary with the mapping

3.8.2 Get Mapping For Event

`Mappings.`**`view_mapping`**`(`*view_field_mappings=False*`)`

Gets the mapping for an event and allows printing with or with the field details

Parameters **`view_field_mappings`** – Hides the field details when set to false. Useful for quick view of consolidation and mapping details without all the fields

Returns Returns the mapping dictionary the user viewed

Sample Mapping printed with view_field_mappings=False. The mapping['fields'] key with the list of fields IS NOT printed.

```
{'autoMappingError': None,
 'consolidation': {'consolidatedSchema': 'MY_SCHEMA',
                   'consolidatedTableName': 'MY_TABLE',
                   'consolidationKeys': ['ID'],
                   'viewSchema': None},
 'inputObjects': {'12345-asdfg': ['98765-zxcvb']},
 'mapping': {'isDiscarded': False,
             'outputHint': '{"table":"my_table","schema":"MY_SCHEMA"}',
             'outputId': 'a1s2d3-f4g5h6',
             'readOnly': False,
             'schema': 'MY_SCHEMA',
             'tableName': 'MY_TABLE_LOG'},
  'mappingMode': 'AUTO_MAP',
  'name': 'MY_SCHEMA.MY_TABLE',
  'origInputLabel': 'production_database',
  'schemaUrls': ['schema?id=12345-asdfg&schema_object=my_table',
                 'schema?id=d=12345-asdfg&sschema_object=deleted_rows'],
  'state': 'MAPPED',
  'usingDefaultMappingMode': False}
```

Sample Mapping printed with view_field_mappings=True. The mapping['fields'] key with the list of fields IS printed.

```
{'autoMappingError': None,
 'consolidation': {'consolidatedSchema': 'MY_SCHEMA',
                   'consolidatedTableName': 'MY_TABLE',
                   'consolidationKeys': ['ID'],
                   'viewSchema': None},
  'fields': [ {'fieldName': 'id',
              'fields': [],
              'mapping': {'columnName': 'ID',
              'columnType': {'nonNull': True,
                              'precision': 38,
                              'scale': 0,
                              'type': 'NUMERIC'},
                              'isDiscarded': False,
                              'machineGenerated': False,
                              'subFields': None}},
              {'fieldName': 'name',
              'fields': [],
              'mapping': {'columnName': 'NAME',
              'columnType': {'length': 16777216,
                              'nonNull': False,
                              'truncate': False,
                              'type': 'VARCHAR'},
              'isDiscarded': False,
              'machineGenerated': False,
              'subFields': None}}
              ],
 'inputObjects': {'12345-asdfg': ['98765-zxcvb']},
 'mapping': {'isDiscarded': False,
             'outputHint': '{"table":"my_table","schema":"MY_SCHEMA"}',
             'outputId': 'a1s2d3-f4g5h6',
             'readOnly': False,
             'schema': 'MY_SCHEMA',
             'tableName': 'MY_TABLE_LOG'},
  'mappingMode': 'AUTO_MAP',
  'name': 'MY_SCHEMA.MY_TABLE',
  'origInputLabel': 'production_database',
  'schemaUrls': ['schema?id=12345-asdfg&schema_object=my_table',
              'schema?id=d=12345-asdfg&sschema_object=deleted_rows'],
  'state': 'MAPPED',
  'usingDefaultMappingMode': False}
```

3.8.3 Change Mapping Mode

`managealooma.mappings.Mappings.`**`change_mapping_mode`**(*self*, *new_mapping_mode='STRICT'*)

Change the mapping mode. Alooma has 3 modes of AUTO_MAP, STRICT, and FLEXIBLE. We only use AUTO_MAP or STRICT

Parameters **`new_mapping_mode`** – The new mapping mode to set: AUTO_MAP, STRICT, and FLEXIBLE

Returns The altered mapping

The mapping settings keys and sample values

```
{'autoMappingError': None,
'mappingMode': None,
'usingDefaultMappingMode': True}
```

The mapping mode with the specified new mappingMode to alter

```
{'autoMappingError': None,
'mappingMode': 'AUTO_MAP',
'usingDefaultMappingMode': True}
```

3.8.4 Change Mapping Consolidation Settings

`managealooma.mappings.Mappings.`**`change_mapping_consolidation_settings`**`(`*self, consolidation_schema, consolidation_table_name, consolidation_keys*`)`

Updates the consolidation information for an event

Parameters

- **`consolidation_schema`** – The schema of consolidated table
- **`consolidation_table_name`** – The name of the consolidated table
- **`consolidation_keys`** – The consolidation keys (primary key) for the table. Takes a single field or a list

Returns The altered mapping

The old consolidation settings

```
{'consolidatedSchema': 'MY_SCHEMA_TEMP',
'consolidatedTableName': 'NY_TABLE_ITEMS',
'consolidationKeys': ['IDENTIFIER'],
'viewSchema': None}
```

The new consolidation settings to apply

```
{'consolidatedSchema': 'MY_SCHEMA',
'consolidatedTableName': 'MY_TABLE',
'consolidationKeys': ['ID'],
'viewSchema': None}
```

3.8.5 Change Mapping Consolidation Key

`Mappings.`**`change_mapping_consolidation_key`**`(`*new_consolidation_key*`)`

Change the conolidation key only

Parameters **`new_consolidation_key`** – The new consolidation key. This is the primary key for the table.

Returns The altered mapping

The consolidation key is set to the field name ID when tables are auto-mapped

```
{'consolidatedSchema': 'MY_SCHEMA',
'consolidatedTableName': 'MY_TABLE',
'consolidationKeys': ['ID'],
'viewSchema': None}
```

This example will change the key to a field named IDENTIFIER

```
{'consolidatedSchema': 'MY_SCHEMA',
'consolidatedTableName': 'MY_TABLE',
'consolidationKeys': ['IDENTIFIER'],
'viewSchema': None}
```

3.8.6 Change Mapping to Use Log

`Mappings.`**`change_mapping_to_use_log`**`()`

Changes the mapping from using the consolidated table to the log table. Used when adjusting manual mappings

Parameters **`event_name`** – The event name of the mapping to alter

Returns The altered mapping

Events without a _log do not have consolidated table information

```
'mapping': {'isDiscarded': False,
 'outputHint': '{"table":"my_table","schema":"MY_SCHEMA"}',
 'outputId': 'a1s2d3-f4g5h6',
 'readOnly': False,
 'schema': 'MY_SCHEMA',
 'tableName': 'MY_TABLE'}
```

This example will change the key to a field named IDENTIFIER

```
'mapping': {'isDiscarded': False,
  'outputHint': '{"table":"my_table","schema":"MY_SCHEMA"}',
  'outputId': 'a1s2d3-f4g5h6',
  'readOnly': False,
  'schema': 'MY_SCHEMA',
  'tableName': 'MY_TABLE_LOG'}
```

3.8.7 Change Mapping for Manual Consolidation Creation

`Mappings.`**`change_mapping_for_manual_consolidation_creation`**`(`*consolidation_schema, consolidation_table_name, consolidation_keys, case='UPPER'*`)`

Updates the mapping after creating the log table manually

Parameters

- **`consolidation_schema`** – The schema of consolidated table
- **`consolidation_table_name`** – The name of the consolidated table

- **consolidation_keys** – The consolidation keys (primary key) for the table. Takes a single field or a list
- **case** – UPPER if your events are MY_SCHEMA.MY_EVENT and LOWER if the events are my_schema_my_event

Returns The altered mapping

First adjust the tables in the data warehouse.

```
ALTER TABLE MY_TABLE RENAME TO MY_TABLE_LOG;
CREATE TABLE MY_TABLE LIKE MY_TABLE_LOG;
```

Then run change_mapping_for_manual_consolidation_creation to change the mapping to insert data into the new log table. This is the old mapping:

```
 'consolidation': {'consolidatedSchema': None,
                   'consolidatedTableName': None,
                   'consolidationKeys': ,
                   'viewSchema': None},
'mapping': {'isDiscarded': False,
            'outputHint': '{"table":"my_table","schema":"MY_SCHEMA"}',
            'outputId': 'a1s2d3-f4g5h6',
            'readOnly': False,
            'schema': 'MY_SCHEMA',
            'tableName': 'MY_TABLE'},
```

These are the changes that will be applied with the new mapping

```
 'consolidation': {'consolidatedSchema': 'MY_SCHEMA',
                   'consolidatedTableName': 'MY_TABLE',
                   'consolidationKeys': ['ID_FIELD'],
                   'viewSchema': None},
'mapping': {'isDiscarded': False,
            'outputHint': '{"table":"my_table","schema":"MY_SCHEMA"}',
            'outputId': 'a1s2d3-f4g5h6',
            'readOnly': False,
            'schema': 'MY_SCHEMA',
            'tableName': 'MY_TABLE_LOG'},
```

Lastly add consolidation queries to combine the new data with the existing data using `create_consolidation()`

3.8.8 Delete Field From Mapping

Mappings.**delete_field_from_mapping**(*field_name*)

Gets a mapping, deletes a field from the mapping, and resets the mapping table statistics.

Parameters

- **event_name** – the name of event for which to make the change
- **field_name** – the field name that should be deleted

Returns The altered mapping

If print_changes is specified the details for the field to remove is printed

```
{'fieldName': 'name',
             'fields': [],
             'mapping': {'columnName': 'NAME',
             'columnType': {'length': 16777216,
                             'nonNull': False,
                             'truncate': False,
                             'type': 'VARCHAR'},
             'isDiscarded': False,
             'machineGenerated': False,
             'subFields': None}}
```

Then the entire new mapping is printed. The 'name' field is not longer in the mapping

```
{'autoMappingError': None,
 'consolidation': {'consolidatedSchema': 'MY_SCHEMA',
                  'consolidatedTableName': 'MY_TABLE',
                  'consolidationKeys': ['ID'],
                  'viewSchema': None},
  'fields': [ {'fieldName': 'id',
              'fields': [],
              'mapping': {'columnName': 'ID',
              'columnType': {'nonNull': True,
                              'precision': 38,
                              'scale': 0,
                              'type': 'NUMERIC'},
                              'isDiscarded': False,
                              'machineGenerated': False,
                              'subFields': None}}
              ]
 'inputObjects': {'12345-asdfg': ['98765-zxcvb']},
 'mapping': {'isDiscarded': False,
             'outputHint': '{"table":"my_table","schema":"MY_SCHEMA"}',
             'outputId': 'a1s2d3-f4g5h6',
             'readOnly': False,
             'schema': 'MY_SCHEMA',
             'tableName': 'MY_TABLE_LOG'},
  'mappingMode': 'AUTO_MAP',
  'name': 'MY_SCHEMA.MY_TABLE',
  'origInputLabel': 'production_database',
  'schemaUrls': ['schema?id=12345-asdfg&schema_object=my_table',
              'schema?id=d=12345-asdfg&sschema_object=deleted_rows'],
  'state': 'MAPPED',
  'usingDefaultMappingMode': False}
```

3.8.9 Change Field Mapping Settings

`Mappings.`**`change_field_mapping_settings`**(*field_name*, *new_data_type*, *truncate=False*, *non_null=False*)

Updates a single filed in a mapping

Parameters

- **`field_name`** – The field name to alter
- **`new_data_type`** – The new datatype for the mapping such as VARCHAR(1024) or INT
- **`truncate`** – Set to True if the event should be truncated when it's longer than the specific mapping length. Redshift's max VARCHAR is 65535 and Snowflake's max VARCHAR is

16777216.

- **`non_null`** – Set to true if the field is needs to be not null

Returns The altered mapping

The current mapping field settings

```
{'fieldName': 'name',
             'fields': [],
             'mapping': {'columnName': 'NAME',
             'columnType': {'INT':
                             'nonNull': False,
                             'truncate': False,
                             'type': 'VARCHAR'},
             'isDiscarded': False,
             'machineGenerated': False,
             'subFields': None}}
```

The new mapping field settings to apply

```
{'fieldName': 'name',
             'fields': [],
             'mapping': {'columnName': 'NAME',
             'columnType': {'length': 1024,
                             'nonNull': False,
                             'truncate': True,
                             'type': 'VARCHAR'},
             'isDiscarded': False,
             'machineGenerated': False,
             'subFields': None}}
```

3.8.10 Change Field Varchar Length

`Mappings.`**`change_field_varchar_length`**(*field_name*, *new_length*)

Updates only the length of a varchar for a field.

Parameters

- **`field_name`** – The field name to alter
- **`new_length`** – The new length for the varchar

Returns None

```
{'fieldName': 'name',
             'fields': [],
             'mapping': {'columnName': 'NAME',
             'columnType': {'length': 1024,
                             'nonNull': False,
                             'truncate': False,
                             'type': 'VARCHAR'},
             'isDiscarded': False,
             'machineGenerated': False,
             'subFields': None}}
```

```
{'fieldName': 'name',
             'fields': [],
```

(continues on next page)

(continued from previous page)

```
              'mapping': {'columnName': 'NAME',
              'columnType': {'length': 16777216,
                              'nonNull': False,
                              'truncate': True,
                              'type': 'VARCHAR'},
              'isDiscarded': False,
              'machineGenerated': False,
              'subFields': None}}
```

3.8.11 Change Field Null Constraint

Mappings.**change_field_null_constraint**(*field_name*, *nonnull=False*)

Removes the NULL constraint from a column

Parameters

- **field_name** – The column for which to remove the constraint
- **nonnull** – The new null setting for the field

Returns The altered mapping

The field mapping details with the current nonnull setting

```
{'fieldName': 'id',
'fields': [],
'mapping': {'columnName': 'ID',
           'columnType': {'nonNull': True,
                          'precision': 38,
                          'scale': 0,
                          'type':'NUMERIC'},
           'isDiscarded': False,
           'machineGenerated': False,
           'subFields': None}}
```

The field mapping details with the new nonnull setting

```
{'fieldName': 'id',
'fields': [],
'mapping': {'columnName': 'ID',
           'columnType': {'nonNull': False,
                          'precision': 38,
                          'scale': 0,
                          'type':'NUMERIC'},
           'isDiscarded': False,
           'machineGenerated': False,
           'subFields': None}}
```

3.8.12 Check Consolidation Uses Log

Mappings.**check_if_consolidation_uses_log**()

Checks if the event uses a log table

Parameters **event_name** – The name of the event for which to check

Returns None

Prints the event name, consolidation key list, the table data is inserted to, and True/False if the insert contains _LOG in the name event_name [‘consolidation_key_list’] mapping_table_name True/False

3.8.13 Copy Mapping

`Mappings.`**`copy_mapping`**(*new_event*)

Copies mapping from the event the class is instantiated with to a new event. Only the name changes

Parameters

- **`new_event`** – The name of the event to copy to
- **`print_mapping`** – Prints the changes if specified
- **`apply_changes`** – Applies the changes if specified

Returns The altered mapping

3.8.14 Set Mapping from Existing Mapping

`Mappings.`**`set_mapping_from_existing_mapping`**(*new_event_name*, *new_schema*, *new_table*, *new_input_label*)

Takes the mapping from the event the class is instantiated with event and uses it to set a new mapping

Parameters

- **`new_event_name`** – The event for which to set the new mapping
- **`new_schema`** – The schema for the new mapping
- **`new_table`** – The table for the new mapping
- **`new_input_label`** – The input label of the new mapping

Returns The altered mapping

3.8.15 Add Field To Mapping

`Mappings.`**`add_field_to_mapping`**(*field_name*, *column_name*, *data_type*, *precision=38*, *scale=0*, *length=16777216*, *truncate=False*, *non_null=False*)

Adds a single field without sub-fields to the mapping. Works for Snowflake data types only.

Parameters

- **`field_name`** – The field name to add
- **`column_name`** – The name of the column in the target DB
- **`data_type`** – The data type for the field out of NUMBER, VARIANT, BOOLEAN, VARCHAR, FLOAT, TIMESTAMP
- **`precision`** – The precision for numeric data
- **`scale`** – The scale for numeric data
- **`length`** – The length for a varchar
- **`truncate`** – Set to True if the event should be truncated when it’s longer than the specific mapping length. Redshift’s max VARCHAR is 65535
- **`non_null`** – Set to true if the field is needs to be not null

Returns The altered mapping

Prints the entire existing mapping before the field is added

```
{'autoMappingError': None,
 'consolidation': {'consolidatedSchema': 'MY_SCHEMA',
                   'consolidatedTableName': 'MY_TABLE',
                   'consolidationKeys': ['ID'],
                   'viewSchema': None},
  'fields': [ {'fieldName': 'id',
              'fields': [],
              'mapping': {'columnName': 'ID',
              'columnType': {'nonNull': True,
                              'precision': 38,
                              'scale': 0,
                              'type': 'NUMERIC'},
                              'isDiscarded': False,
                              'machineGenerated': False,
                              'subFields': None}}
              ]
 'inputObjects': {'12345-asdfg': ['98765-zxcvb']},
 'mapping': {'isDiscarded': False,
             'outputHint': '{"table":"my_table","schema":"MY_SCHEMA"}',
             'outputId': 'a1s2d3-f4g5h6',
             'readOnly': False,
             'schema': 'MY_SCHEMA',
             'tableName': 'MY_TABLE_LOG'},
  'mappingMode': 'AUTO_MAP',
  'name': 'MY_SCHEMA.MY_TABLE',
  'origInputLabel': 'production_database',
  'schemaUrls': ['schema?id=12345-asdfg&schema_object=my_table',
              'schema?id=d=12345-asdfg&schema_object=deleted_rows'],
  'state': 'MAPPED',
  'usingDefaultMappingMode': False}
```

Then prints the entire new mapping to set after the field is added. This example adds the NAME field as a VARCHAR

```
 {'autoMappingError': None,
'consolidation': {'consolidatedSchema': 'MY_SCHEMA',
                  'consolidatedTableName': 'MY_TABLE',
                  'consolidationKeys': ['ID'],
                  'viewSchema': None},
 'fields': [ {'fieldName': 'id',
             'fields': [],
             'mapping': {'columnName': 'ID',
             'columnType': {'nonNull': True,
                             'precision': 38,
                             'scale': 0,
                             'type': 'NUMERIC'},
                             'isDiscarded': False,
                             'machineGenerated': False,
                             'subFields': None}},
             {'fieldName': 'name',
             'fields': [],
             'mapping': {'columnName': 'NAME',
             'columnType': {'length': 16777216,
                             'nonNull': False,
```

(continues on next page)

(continued from previous page)

```
                              'truncate': False,
                              'type': 'VARCHAR'},
             'isDiscarded': False,
             'machineGenerated': False,
             'subFields': None}}
             ]
'inputObjects': {'12345-asdfg': ['98765-zxcvb']},
'mapping': {'isDiscarded': False,
            'outputHint': '{"table":"my_table","schema":"MY_SCHEMA"}',
            'outputId': 'a1s2d3-f4g5h6',
            'readOnly': False,
            'schema': 'MY_SCHEMA',
            'tableName': 'MY_TABLE_LOG'},
 'mappingMode': 'AUTO_MAP',
 'name': 'MY_SCHEMA.MY_TABLE',
 'origInputLabel': 'production_database',
 'schemaUrls': ['schema?id=12345-asdfg&schema_object=my_table',
             'schema?id=d=12345-asdfg&sschema_object=deleted_rows'],
 'state': 'MAPPED',
 'usingDefaultMappingMode': False}
```

3.8.16 Remove Unmapped Fields and Clear Table Stats

`Mappings.`**`remove_unmapped_fields_and_clear_table_stats`**`()`

Remove unmapped fields from the a mapping and clear the UI stats. The mapper can become very slow over time and clearing the table stats periodically will help speed it back up.

Returns none

3.9 Samples

Samples are dictionaries with a small subset of data for each event type.

`Samples.`**`__init__`**`(`*api*, *sample_event_directory*`)`

For each event Alooma stores sample dictionaries. This class will help you interact and copy the samples for testing. We run integration testing on the entirety of the sample set before deploying.

Parameters

- **`api`** – The Alooma API client authentication
- **`sample_event_directory`** – The full path and name of the directory in which to store sample events such as /Users/myname/code/hover/alooma-etl/sample_events

3.9.1 Build Sample File Path and File Name

`Samples.`**`build_sample_file_path_and_file_name`**`(`*sample_file_type='event'*, *name=None*`)`

Constructs the file path and name for writing the samples

Parameters

- **`sample_file_type`** – The type of file either 'event' or 'input'
- **`name`** – The name of the event or input

Returns a file path with the file name

3.9.2 Print Samples

static `Samples.`**`print_samples`**(*sample_list*, *view_all_or_index='all'*, *pprint_indent=2*, *pprint_width=200*, *pprint_depth=5*)

Prints all the samples in a list or pretty prints a single sample

Parameters

- **`sample_list`** – A list of samples
- **`view_all_or_index`** – 'all' to print all samples or a number to print sample by index
- **`pprint_indent`** – The indent value to pprint dictionaries
- **`pprint_width`** – The width value to pprint dictionaries
- **`pprint_depth`** – The depth value to pprint dictionaries

Returns None

3.9.3 Get Samples for Event from Alooma API

`Samples.`**`get_samples_for_event_from_alooma_api`**(*event_name*)

Gets samples for a single event from the API

Parameters **`event_name`** – The event for which to get samples

Returns A list of samples

3.9.4 View Samples for Event from Alooma API

`Samples.`**`view_samples_for_event_from_alooma_api`**(*event_name*, *view_all_or_index='all'*)

View samples for an event from the API. Print all or a single event by index.

Parameters

- **`event_name`** – The name of the event for which to view samples
- **`view_all_or_index`** – Specify to view 'all' the samples for an event or a specific event by index number

3.9.5 Get Samples from File

static `Samples.`**`get_samples_from_file`**(*file_path*)

Returns

3.9.6 Get Samples from Saved Sample Files

`Samples.`**`get_samples_from_saved_sample_files`**(*sample_file_type='event'*, *name=None*)

Gets the samples for an event or input from a saved file

Parameters

- **sample_file_type** – The name of the file to retrieve samples for. Exclude the .json extension
- **name** – The name of the event or input to get the samples for

Returns A list with the samples from the file

3.9.7 View Samples from File

`Samples.`**`view_samples_from_file`**(*sample_file_type='event', name=None, view_all_or_index='all', pprint_indent=2, pprint_width=200, pprint_depth=5*)

Prints the samples for an event or input from the API

Parameters

- **sample_file_type** – Whether the samples are for an 'event' or an 'input'
- **name** – The name of the event or input
- **view_all_or_index** – Specify to view 'all' the samples for an event or a specific event by index
- **pprint_indent** – The indent value to pprint dictionaries
- **pprint_width** – The width value to pprint dictionaries
- **pprint_depth** – The depth value to pprint dictionaries

Returns None

3.9.8 Save Alooma Samples to Files

`Samples.`**`save_alooma_samples_to_files`**(*event_name=None, input_name=None*)

Saves samples to files in the specified directory. If no event_name or input_name are specified samples for ALL events will be written to a file

Parameters

- **event_name** – The event name for samples to save. If specified only samples for this event are retrieved
- **input_name** – The input name for samples to save. If specified all samples for this input are retrieved

Returns None

3.9.9 Write Alooma Samples to Files

`Samples.`**`write_alooma_samples_to_files`**(*event_list, input_name=None*)

Writes samples to files for each event. Writes input file with samples for all events in the input when input_name is specified.

Parameters

- **event_list** – A list of events for which to write samples for
- **input_name** – The name of the input to write the samples for to limit the samples to a specific input

Returns None

3.9.10 Get Sample from Any API

static Samples.**get_sample_from_any_api**(*url*, *payload*, *api_key*, *input_label*, *print_api_response=True*, *test_index_number=0*, *input_type='rest_endpoint'*)

Gets a sample to test from a 3rd party API using the requests package and adds the required Alooma _metadata to the event

Parameters

- **url** – The URL for which to make the get request e.g. https://harvest.greenhouse.io/v1/scheduled_interviews
- **payload** – A JSON dictionary with the parameters to add to the requests e.g. {'updated_after': '2018-08-25T00:00:00Z'}
- **api_key** – The API key with which to make the request
- **preview_events** – If True all the events retrieved will be printed to the console
- **input_label** – The input_label to give these events when deployed in Alooma
- **test_index_number** – The index number of the event to test
- **input_type** – The type of input the data will use when deployed to Alooma

Returns An event with metadata added that can be tested through the transformation code

3.9.11 Get Sample from PostgreSQL Database

static Samples.**get_sample_from_postgresql_database**(*db_host*, *db_port*, *db_user*, *db_password*, *db_name*, *input_label*, *table_name*, *id_field*, *row_id*, *print_event=True*)

3.10 Secrets

Secrets are Alooma's method for storing environment variables.

Secrets.**__init__**(*api*)

Manages secrets in Alooma. Secrets are Alooma's way of using Environment variables.

Parameters **api** – The Alooma API client authentication

3.10.1 Set Secrets

Secrets.**set_secrets**(*secrets_dict*)

Sets the secrets in Alooma.

Parameters **secrets_dict** – A dictionary with the secrets as k, v pairs. ex: {"my_user": "example", "my_password": "12345678"}

Returns None

3.10.2 Get Secrets

`Secrets.`**`get_secrets`**`()`

Gets the list of secrets and prints the list.

Returns None

3.10.3 Delete Secrets

`Secrets.`**`delete_secrets`**`(`*secret*`)`

Deletes a secret from Alooma.

Parameters **`secret`** – The name of the secret to delete. ex: {"my_user": "example", "my_password": "12345678"} Set to my_user to delete that secret.

Returns None

3.11 System

Retrieve and interact with system notifications programatically.

`System.`**`__init__`**`(`*api*`)`

Gets information on system information like notifications, status codes, and metrics :param api: api authentication using the Alooma package

3.11.1 Get Status Codes

`System.`**`get_status_codes`**`()`

Gets the possible status codes from Alooma

Returns A dictionary with status codes and their descriptions

3.11.2 Status Code Info

`System.`**`status_code_info`**`(`*print_format='table', table_limit=None, pprint_indent=2, pprint_width=20, pprint_depth=5*`)`

Prints the status codes that Alooma may return from with the event info

Parameters

- **`print_format`** – string 'table' to print event info as tables or 'json' to print as dictionaries
- **`table_limit`** – A limit to the columns to print
- **`pprint_indent`** – The indent value to pprint dictionaries
- **`pprint_width`** – The width value to pprint dictionaries
- **`pprint_depth`** – The depth value to pprint dictionaries

Returns A dictionary with the status codes and their descriptions

3.11.3 System Metric Info

`System.`**`system_metric_info`**(*metric_type_names=None*, *last_n_minutes=60*)

Gets the system metrics and prints a dataframe with the results

Parameters

- **`metric_type_names`** – string or list A list of systems metrics or a single metric name.
- **`last_n_minutes`** – The length of time in minutes counting back from the current time to retrieve notifications for

Returns A dataframe with the inforamtion

3.12 Transformation Functions

Generic reusable functions to apply transformations to event dictionaries.

3.12.1 Add Column Based On Null

`managealooma.transformation_functions.`**`add_column_based_on_null`**(*event*, *field*, *new_field*, *new_value_if_null*, *new_value_if_not_null*)

Checks and adds a value to a new field based on NULL

Parameters

- **`event`** – A dictionary
- **`field`** – The name of the field to check
- **`new_field`** – The name of the new field
- **`new_value_if_null`** – The value for the new_field if field IS NULL
- **`new_value_if_not_null`** – The value for the new_field if field IS NOT NULL

Returns An altered dictionary

Examples:

```
# Example #1
event = {'do i have digits?': '1234'}
event = add_column_based_on_null(event, field='do i have digits?', new_field=
↪'digits', new_value_if_null='N', new_value_if_not_null='Y')
event = {'digits': 'Y',
         'do i have digits?': '1234'}

# Example #2
event = {'do i have digits?': None}
event = add_column_based_on_null(event, field='do i have digits?', new_field=
↪'digits', new_value_if_null='N', new_value_if_not_null='Y')
event = {'digits': 'N',
         'do i have digits?': None}
```

3.12.2 Add Columns with Default

managealooma.transformation_functions.**add_columns_with_default**(*event*, *field_and_default_dict*)

Adds a column with the default value to every event where it's not already present

Parameters

- **event** – A dictionary
- **field_and_default_dict** – A dictionary with keys and default values {field_1: default_for_field_1, field_2: default_for_field_2}

Returns An altered dictionary

Examples:

```
# Example #1
event = {'an existing column': 'some value'}
event = add_columns_with_default(event, field_and_default_dict={"new column 1":
↪'my default of 1', "new column two": 2})
event = {'an existing column': 'some value',
         'new column 1': 'my default of 1',
         'new column two': 2}

# Example #2
event = {'an existing column': 'some value'}
event = add_columns_with_default(event, field_and_default_dict={"mark_for_delete
↪": False})
event = {'an existing column': 'some value',
         'mark_for_delete': False}
```

3.12.3 Add Composite Key

managealooma.transformation_functions.**add_composite_key**(*event*, *field_list*, *key_name*)

Creates a composite key to be used for a unique ID.

Parameters

- **event** – A dictionary
- **field_list** – A list of fields to combine to make the key
- **key_name** – The name of the new key field

Returns An altered dictionary

Examples:

```
# Example #1
event = {'a_field': 1234,
         'a_second_field': '2019-01-01',
         'a_third_field': 'abc'}
event = add_composite_key(event, field_list=['a_field', 'a_second_field', 'a_
↪third_field'], key_name='my_derived_key')
event = {'a_field': 1234,
         'a_second_field': '2019-01-01',
         'a_third_field': 'abc',
```

(continues on next page)

(continued from previous page)

```
         'my_derived_key':
         '1234-2019-01-01-abc'}
```

3.12.4 Add Duplicate Fields

`managealooma.transformation_functions.`**`add_duplicate_fields`**(*event*, *field_name*, *suffix_or_suffix_list*, *keep_original=False*)

Add duplicate values of a field with a suffix

Parameters

- **`event`** – A dictionary
- **`field_name`** – The name of the field to duplicate
- **`suffix_or_suffix_list`** – A single or list of suffixes to add to the field_name in the duplicates
- **`keep_original`** – True to keep the original field also

Returns An altered dictionary

Examples:

```
# Example #1
event = {'a_field': 'a_value'}
event = add_duplicate_fields(event, field_name='a_field', suffix_or_suffix_list=
↪'my_suffix', keep_original=False)
event = {'a_field_my_suffix': 'a_value'}

# Example #2
event = {'a_field': 'a_value'}
event = add_duplicate_fields(event, field_name='a_field', suffix_or_suffix_list=
↪'my_suffix', keep_original=True)
event = {'a_field': 'a_value',
         'a_field_my_suffix': 'a_value'}

# Example #3
event = {'a_field': 'a_value'}
event = add_duplicate_fields(event, field_name='a_field', suffix_or_suffix_list=[
↪'my_suffix','my_second_suffix'], keep_original=False)
event = {'a_field_my_second_suffix': 'a_value',
         'a_field_my_suffix': 'a_value'}
```

3.12.5 Add Suffix

`managealooma.transformation_functions.`**`add_suffix`**(*event*, *fields*, *suffix*, *separator='_'*)

Adds a suffix to a field or list of fields

Parameters

- **`event`** – A dict with the entire event
- **`field_or_field_list`** – A single field or list of fields for which to add a suffix
- **`suffix`** – The suffix to add to the fields

- **`separator`** – The character to place between the name and the suffix

Returns An altered event with the altered field names

Examples:

```
# Example #1
event = {'a_field': 'a_value'}
event = add_suffix(event, fields='a_field', suffix='an_ending')
event = {'a_field_an_ending': 'a_value'}

# Example #2
event = {'a_field': 'a_value'}
event = add_suffix(event, fields='a_field', suffix='an_ending', separator='---')
event = {'a_field---an_ending': 'a_value'}

# Example #3
event = {'a_field': 'a_value',
         'another_field': 'another_value'}
event = add_suffix(event, fields=['a_field', 'another_field'], suffix='an_ending')
event = {'a_field_an_ending': 'a_value',
         'another_field_an_ending': 'another_value'}
```

3.12.6 Convert All Event Fields to Snake Case

`managealooma.transformation_functions.`**`convert_all_event_fields_to_snake_case`**(*event*)

Converts all keys in an event to snake case. If a key is Partially_SnakeCase we'll get 2 underscores where there is currently one like partially__snake_case

Parameters **`event`** – An Alooma event

Returns A transformed event with all the keys in snake_case

Examples:

```
# Example #1
event = {'_metadata': {},
         'TitleCase': 'to_snake_case',
         'camelCase': 'to_snake_case',
         'snake_case': 'to_snake_case'}
event = convert_all_event_fields_to_snake_case(event)
event = {'_metadata': {},
         'camel_case': 'to_snake_case',
         'snake_case': 'to_snake_case',
         'title_case': 'to_snake_case'}
```

3.12.7 Convert Dictionary Fields to String

`managealooma.transformation_functions.`**`convert_dictionary_fields_to_string`**(*event*, *field_or_field_list*)

Dumps a list of fields to a string to keep Alooma from auto-parsing

Parameters

- **`event`** – A dict with the entire event
- **`field_or_field_list`** – A single field or list of fields to json.dumps to keep Alooma from doing infinte de-nesting

Returns A new event

Examples:

```
# Example #1
event = {'a_dict': {'field_one': 1,
        'field_two': 2}}
event = convert_dictionary_fields_to_string(event, field_or_field_list='a_dict')
event = {'a_dict': '{"field_one": 1, "field_two": 2}'}

# Example #2
event = {'a_dict': {'field_one': 1,
                    'field_two': 2},
        'a_second_dict': {'field_one': 1,
                          'field_two': 2}}
event = convert_dictionary_fields_to_string(event, field_or_field_list=['a_dict',
↪'a_second_dict'])
event = {'a_dict': '{"field_one": 1, "field_two": 2}',
        'a_second_dict': '{"field_one": 1, "field_two": 2}'}
```

3.12.8 Convert Null to Zero

`managealooma.transformation_functions.`**`convert_null_to_zero`**`(`*event*, *field_or_field_list*`)`

Converts the value in a field or field list from None to 0

Parameters

- **`event`** – a dict with the event
- **`field_or_field_list`** – A single field or list of fields to convert to 0 if null

Returns the updated event

Examples:

```
# Example #1
event = {'a_field': None}
event = convert_null_to_zero(event, field_or_field_list='a_field')
event = {'a_field': 0}

# Example #2
event = {'a_field': None,
        'another_field': None}
event = convert_null_to_zero(event, field_list=['a_field', 'another_field'])
event = {'a_field': 0,
        'another_field': 0}
```

3.12.9 Convert Spaces and Special Characters to Underscore

`managealooma.transformation_functions.`**`convert_spaces_and_special_characters_to_underscore`**`(`

Converts spaces and special characters to underscore so 'Thi$ i# jun&' becomes 'thi__i__jun_'

Parameters **`name`** – A string

Returns An altered string

Example use case: - A string might have special characters at the end when they are really the same field such as My_Field$ and My_Field# - We use this to covert the names to "my_field" to combine the values so the events will be easily grouped together

Examples:

```
# Example #1
input_string = '$Scr "get-rid^-of-the@" special #characters%&space'
output_string = convert_spaces_and_special_characters_to_underscore(input_string)
output_string = '_scr__get_rid__of_the___special__characters__space'
```

3.12.10 Convert String to Snake Case

managealooma.transformation_functions.**convert_string_to_snake_case**(*name*)

Converts a string to Snake Case

Parameters **name** – A string

Returns A string in snake case

Example use case: - Events from might have custom properties in camelCase like userId, and userEmail - Use this to rename keys to user_id and user_email for better ease of reading in the database

Examples:

```
# Example #1
input_string = 'MakeThisSnakeCase'
output_string = convert_string_to_snake_case(input_string)
output_string = 'make_this_snake_case'

# Example #2
input_string = 'Make This Snake Case'
output_string = convert_string_to_snake_case(input_string)
output_string = 'make_this_snake_case'

# Example #3
input_string = 'keep_this_snake_case'
output_string = convert_string_to_snake_case(input_string)
output_string = 'keep_this_snake_case'
```

3.12.11 Convert Values to None

managealooma.transformation_functions.**convert_values_to_none**(*event*, *field_or_field_list*, *field_values=None*)

Changes a field to None. If a field value is specified then only that value will be changed to None

Parameters

- **event** – A dictionary
- **field_or_field_list** – A single field or list of fields to convert to None
- **field_values** – The value to convert to None. If specified only these values are converted to None

Returns An altered dictionary

Examples:

```
# Example #1
event = {'a_field': 'a_value'}
event = convert_values_to_none(event, field_or_field_list='a_field')
event = {'a_field': None}

# Example #2
event = {'a_field': 'a_value',
         'another_field': 'another_value'
event = convert_values_to_none(event, field_or_field_list=['a_field', 'another_
↪field'])
event = {'a_field': None,
         'another_field': None}

# Example #3
event = {'a_field': 'a_value',
         'another_field': 'another_value'
event = convert_values_to_none(event, fields=['a_field', 'another_field'], field_
↪values='a_value')
event = {'a_field': None,
         'another_field': 'another_value'}

# Example #4
event = {'a_field': 'a_value',
         'another_field': 'another_value'
event = convert_values_to_none(event, fields=['a_field', 'another_field'], field_
↪values=['a_value', 'another_value'])
event = {'a_field': None,
         'another_field': None}
```

3.12.12 Convert Empty Value to None

managealooma.transformation_functions.**convert_empty_value_to_none**(*event*, *key_name*)

Changes an empty string of "" or " ", and empty list of [] or an empty dictionary of {} to None so it will be NULL in the database

Parameters

- **event** – A dictionary
- **key_name** – The key for which to check for empty strings

Returns An altered dictionary

Examples:

```
# Example #1
event = {'a_field': ' '}
event = convert_empty_value_to_none(event, key_name='a_field')
event = {'a_field': None}

# Example #2
event = {'a_field': '{}'}
event = convert_empty_value_to_none(event, key_name='a_field')
event = {'a_field': None}

# Example #3
```

(continues on next page)

(continued from previous page)

```
event = {'a_field': {}}
event = convert_empty_value_to_none(event, key_name='a_field')
event = {'a_field': None}
```

3.12.13 Convert Event Type Case

`managealooma.transformation_functions.`**`convert_event_type_case`**(*event*, *case_force_upper=False*)

Forces upper of lower case for event types at the end of the code engine. For Snowfalke force UPPER and for Redshift force lower.

Parameters

- **`event`** – A dict with the entire event
- **`case_force_upper`** – True to for upper case.

Returns An event with the case altered in event_type

Examples:

```
# Example #1
event = {'_metadata': {'event_type': 'My_SCHEMA.my_table'}}
event = convert_event_type_case(event)
event = {'_metadata': {'event_type': 'my_schema.my_table'}}

# Example #2
event = {'_metadata': {'event_type': 'My_SCHEMA.my_table'}}
event = convert_event_type_case(event, case_force_upper=True)
event = {'_metadata': {'event_type': 'MY_SCHEMA.MY_TABLE'}}
```

3.12.14 Flatten JSON

`managealooma.transformation_functions.`**`flatten_json`**(*event*, *field_or_field_list*, *levels*, *keep_original=False*, *dump_to_string=False*)

Flattens a list of fields from a dictionary n levels

Parameters

- **`event`** – the event that you want to pass through the function (formatted as a dictionary)
- **`field_or_field_list`** – A field or list of the fields that you want to flatten N levels deep
- **`levels`** – The number of levels that you want to parse the fields
- **`keep_original`** – True if you want to keep the original field in the event, false if you want to delete it

Returns The transformed event

Examples:

```
# Example #1
event = {'a_dict': {'field_one': 1,
                    'field_two': 2}}
```

(continues on next page)

(continued from previous page)

```
event = flatten_json(event, field_or_field_list='a_dict', levels=1)
event = {'a_dict_field_one': 1,
        'a_dict_field_two': 2}

# Example #2
event = {'a_dict': {'field_one': 1,
                    'field_two': 2},
        'a_second_dict': {'field_one': {'one more': 1.1,
                                        'one more again': 1.2},
                          'field_two': 2}}
event = flatten_json(event, field_or_field_list=['a_dict','a_second_dict'],␣
↪levels=1)
event = {'a_dict_field_one': 1,
        'a_dict_field_two': 2,
        'a_second_dict_field_one': {'one more': 1.1, 'one more again': 1.2},
        'a_second_dict_field_two': 2}

# Example #3
event = {'a_dict': {'field_one': 1,
                    'field_two': 2},
        'a_second_dict': {'field_one': {'one more': 1.1,
                                        'one more again': 1.2},
                          'field_two': 2}}
event = flatten_json(event, field_or_field_list=['a_dict','a_second_dict'],␣
↪levels=2)
event = {'a_dict_field_one': 1,
        'a_dict_field_two': 2,
        'a_second_dict_field_one_one more': 1.1,
        'a_second_dict_field_one_one more again': 1.2,
        'a_second_dict_field_two': 2}
```

3.12.15 Flatten JSON 1 Level

`managealooma.transformation_functions.`**`flatten_json_1_level`**(*event*, *field_name*, *field_name_underscore*, *dump_to_string*)

Flattens a JSON field 1 level. This function is used in flatten JSON

Parameters

- **`event`** – A dictionary
- **`field_name`** – The field name to flatten
- **`field_name_underscore`** – The field name with an underscore appended
- **`dump_to_string`** – If true any remaining dictionaries will be converted to a string with json.dumps

Returns An event with the field flattened

Examples:

```
# Example #1
event = {'my_field': "{"a": None, "b"}"}
event = flatten_json_1_level(event=input_event, field_name='my_field', field_name_
↪underscore='my_field_', dump_to_string=True)
```

(continues on next page)

(continued from previous page)

```
output_event = {'my_field_a': None,
                'my_field_b': 2}
```

3.12.16 Map Key in Dictionary to Value

managealooma.transformation_functions.**map_key_in_dictionary_to_value**(*event*, *mapping_dict*, *existing_column*, *new_column*, *allow_nulls*)

Adds a column mapping using a dictionary

Parameters

- **event** – A dictionary
- **mapping_dict** – A mapping dict such as {1: 'product A', 2: 'product B'}
- **existing_column** – The column that matches the keys in the mapping dict
- **new_column** – The name of the column to put the values from the mapping dict
- **allow_nulls** – True if the function should let a NULL value in the existing_column pass through. False to throw an error when the existing column has NULL.

Returns An event with the new_column k, v added

Examples:

```
# Example #1
event = {'a_field': 1,
        '_metadata': {'event_type': 'MY_SCHEMA.MY_TABLE'}}
event = map_key_in_dictionary_to_value(event, mapping_dict={1: 'one', 2: 'two'},␣
↪existing_column='a_field', new_column='a_mapped_field', allow_nulls=False)
event = {'_metadata': {'event_type': 'MY_SCHEMA.MY_TABLE'},
        'a_field': 1,
        'a_mapped_field': 'one'}

# Example #2
event = {'a_field': 3,
        '_metadata': {'event_type': 'MY_SCHEMA.MY_TABLE'}}
event = map_key_in_dictionary_to_value(event, mapping_dict={1: 'one', 2: 'two'},␣
↪existing_column='a_field', new_column='a_mapped_field', allow_nulls=False)
Exception: Missing enum transform MY_SCHEMA.MY_TABLE a_field

# Example #3
event = {'a_field': None,
        '_metadata': {'event_type': 'MY_SCHEMA.MY_TABLE'}}
event = map_key_in_dictionary_to_value(event, mapping_dict={1: 'one', 2: 'two'},␣
↪existing_column='a_field', new_column='a_mapped_field', allow_nulls=True)
event = {'_metadata': {'event_type': 'MY_SCHEMA.MY_TABLE'},
        'a_field': None,
        'a_mapped_field': None}
```

3.12.17 Map Value in List to Dictionary Key

`managealooma.transformation_functions.`**`map_value_in_list_to_dictionary_key`**(*event*, *mapping_dict_with_lists*, *existing_column*, *new_column*, *allow_nulls*, *passthrough*)

Maps a value from a list back to the key. Useful to map values to categories.

Parameters

- **`event`** – A dictionary
- **`mapping_dict_with_lists`** – A mapping dict with lists in the values such as {"baked good": ["cake", "croissant"]}
- **`existing_column`** – The column with the existing data
- **`new_column`** – The name of the new column for the added data
- **`allow_nulls`** – True if the existing column can have NULL. If set to False NULLs will throw an error
- **`passthrough`** – True if we should pass through a value of the existing column when there is no mapping value in the list

Returns An altered event

Examples:

```
# Example #1
event = {'_metadata': {'event_type': 'MY_SCHEMA.MY_TABLE'},
        'a_field': 1}
event = map_value_in_list_to_dictionary_key(event, mapping_dict_with_lists={'1-3
↪': [1, 2, 3], '4-6': [4, 5, 6]}, existing_column='a_field', new_column='a_
↪mapped_field', allow_nulls=False, passthrough=False)
event = {'_metadata': {'event_type': 'MY_SCHEMA.MY_TABLE'},
        'a_field': 1,
        'a_mapped_field': '1-3'}

# Example #2
event = {'_metadata': {'event_type': 'MY_SCHEMA.MY_TABLE'},
        'a_field': 7}
event = map_value_in_list_to_dictionary_key(event, mapping_dict_with_lists={'1-3
↪': [1, 2, 3], '4-6': [4, 5, 6]}, existing_column='a_field', new_column='a_
↪mapped_field', allow_nulls=False, passthrough=False)
Exception: Missing map_list transform MY_SCHEMA.MY_TABLE a_field
```

3.12.18 Mark for Delete

`managealooma.transformation_functions.`**`mark_for_delete`**(*event*)

We created database triggers in our database to write all rows to a polymorphic deleted records table upon hard delete. We log the table_name, the time it was deleted, and the row_to_json This function creates a new row that looks like a soft delete came from the database

Parameters **event** – A dictionary that includes the Alooma _metadata dictionary

Returns A dictionary that looks like a soft deleted row

Examples:

```
# Example #1
event = {'id': 1,
         'table_name': 'alooma_test',
         'primary_key': '123',
         'old_row_json': '{"id":6, "created_at":"2019-01-01"}',
         '_metadata': {'event_type': 'test'}}
event = mark_for_delete(event)
event = {'_metadata': {'event_type': 'alooma_test',
                         'table': 'alooma_test'},
         'created_at': '2019-01-01',
         'id': 6,
         'mark_for_delete': True}
```

3.12.19 Parse List of JSON and Concat

managealooma.transformation_functions.**parse_list_of_json_and_concat**(*event*, *field_name*, *keep_original*, *field_to_keep*)

Iterates through a dictionary and creates a single field with a list of values from the field Output is similar to group_concat and listagg in SQL

Parameters

- **event** – A dictionary
- **field_name** – The name of the field to extract data from
- **keep_original** – True to keep the original field. False to delete the original field and only keep the parsed data.
- **field_to_keep** – The name of the field from within the dictionary

Returns An altered dictionary

Examples:

```
# Example #1
event = {'list_of_dicts': [{'key_to_concat': 123, 'key_to_ignore': 'abc'},
                           {'key_to_concat': 456, 'key_to_ignore': 'def'},
                           {'key_to_concat': 789, 'key_to_ignore': 'ghi'}]}
event =  parse_list_of_json_and_concat(input_event, field_name='list_of_dicts',␣
↪keep_original=True, field_to_keep='key_to_concat')
event = {'list_of_dicts': [{'key_to_concat': 123, 'key_to_ignore': 'abc'},
                           {'key_to_concat': 456, 'key_to_ignore': 'def'},
                           {'key_to_concat': 789, 'key_to_ignore': 'ghi'}],
         'list_of_dicts_key_to_concats': [123, 456, 789]}
```

3.12.20 Remove Duplicate Field

`managealooma.transformation_functions.`**`remove_duplicate_field`**`(`*event*, *field_to_keep*, *field_to_discard*`)`

Remove a field when both fields are present

Parameters

- **`event`** – A dictionary
- **`field_to_keep`** – The field to keep if both keys are present and the value is not None
- **`field_to_discard`** – The field to discard if the field_to_keep is not None

Returns An event with a single bundle ID

Examples:

```
# Example #1
event = {'A_Field': 'another_value', 'a_field': 'a_value '}
event = remove_duplicate_field(event, field_to_keep='a_field', field_to_discard=
↪'A_Field')
event = {'a_field': 'a_value '}
```

3.12.21 Remove Outer Key

`managealooma.transformation_functions.`**`remove_outer_key`**`(`*event*, *key_name*`)`

Removes the outer key from an event

Parameters

- **`event`** – A dict with the entire event
- **`key_name`** – The key to remove from the dictionary

Returns An event with the outer key for the specified dictionary removed

Examples:

```
# Example #1
event = {'outer_dict': {'a_field': 'a_value ',
                        'another_field': 'another_value'}}
event = remove_outer_key(event, key_name='outer_dict')
event = {'a_field': 'a_value ',
         'another_field': 'another_value'}
```

3.12.22 Remove Starting Characters from Keys

`managealooma.transformation_functions.`**`remove_starting_characters_from_keys`**`(`*event*, *starting_characters*, *field_with_json=None*`)`

Removes the specified starting characters from all keys in an event

Parameters

- **`event`** – A dict with the entire event
- **`starting_characters`** – The characters to remove from the beginning of the key

- **field_with_json** – A specific field with nested json from which to remove the characters from its keys

Returns a modified event

Examples:

```
# Example #1
event = {'_metadata': {},
         '$a_field': 'a_value',
         '$another_field': 'another_value'}
event = remove_starting_characters_from_keys(event, starting_characters='$')
event = {'_metadata': {},
         'another_field': 'another_value',
         'field': 'a_value'}

# Example #2
event = {'_metadata': {},
         'a_dict': {'$a_field': 'a_value',
                    '$another_field': 'another_value'},
         '$outer_field': 'some value'}
event = remove_starting_characters_from_keys(event, starting_characters='$',␣
↪field_with_json='a_dict')
event = {'$outer_field': 'some value',
         '_metadata': {},
         'a_dict': {'a_field': 'a_value',
                    'another_field': 'another_value'}}
```

3.12.23 Remove Whitespace

managealooma.transformation_functions.**remove_whitespace**(*event*, *field_or_field_list*)

Remove leading and trailing whitespace

Parameters

- **event** – A dictionary
- **field_or_field_list** – A field or list of fields to trim the whitespace from

Returns A trimmed string

Examples:

```
# Example #1
event = {'a_field': '  should not have whitespace at ends    '}
event = remove_whitespace(event, field_or_field_list='a_field')
event = {'a_field': 'should not have whitespace at ends'}

# Example #2
event = {'a_field': '  should not have whitespace at ends    ',
         'another_field': '  also should not have whitespace at ends    '}
event = remove_whitespace(event, field_or_field_list=['a_field', 'another_field'])
event = {'a_field': 'should not have whitespace at ends',
         'another_field': 'also should not have whitespace at ends'}
```

3.12.24 Rename Fields

managealooma.transformation_functions.**rename_fields**(*event*, *field_dict*)

Renames fields from the key to value :param event: A dict with the entire event :param field_dict: A dict with the rename mapping with the key as the existing field and the value as the new field :return: An altered event with the renamed fields

Examples:

```
# Example #1
event = {'a_field': 'a_value',
         'another_field': 'another_value',
         'no_change': 'same'}
event = rename_fields(event, field_dict={'a_field': 'field_one', 'another_field':
↪'field_two'})
event = {'field_one': 'a_value',
         'field_two': 'another_value',
         'no_change': 'same'}
```

3.12.25 Split Event to Multiple Events

managealooma.transformation_functions.**split_event_to_multiple_events**(*event*, *table_name_list*)

Splits events into a list of events with a schema_name.table_name

Parameters

- **event** – A dict with a single event
- **table_name_list** – The table names for the new events

Returns A list of the new events. If an event has already been split it will not re-split and returns the original event.

Examples:

```
# Example #1
event = {'_metadata': {'@uuid': '123-abc', 'event_type':
                       'my_schema.my_table'},
         'a_field': 'a_value'}
event = split_event_to_multiple_events(event, table_name_list=['table_one',
↪'table_two'])
# A parent UUID is added in Alooma when events are split.  Local testing won't␣
↪add a parent UUID.
event = [{'_metadata': {'@parent_uuid': '123-abc',
                       '@uuid': '456-def',
                       'event_type': 'my_schema.table_one'},
         'a_field': 'a_value'},
         {'_metadata': {'@uuid': '123-abc',
                       '@uuid': '789-ghi',
                       'event_type': 'my_schema.table_two'},
         'a_field': 'a_value'}]

# Example #2
event = {'_metadata': {'@parent_uuid': '123-abc',
                       '@uuid': '456-def',
                       'event_type': 'my_schema.table_one'},
```

(continues on next page)

(continued from previous page)

```
        'a_field': 'a_value'}
event = split_event_to_multiple_events(event, table_name_list=['table_one',
↪'table_two'])
# If the event has a parent_uuid it will not be re-split
event = {'_metadata': {'@parent_uuid': '123-abc',
                        '@uuid': '456-def',
                        'event_type': 'my_schema.table_one'},
        'a_field': 'a_value'}
```

3.12.26 Split Field List to Multiple Events

managealooma.transformation_functions.**split_field_list_to_multiple_events**(*event*, *fields_to_split*, *add_counter*, *counter_name*, *reverse=False*)

Take an event that has columns that are lists (of same length) and break into multiple rows

Parameters

- **event** – A dictionary
- **fields_to_split** – The field with to split
- **add_counter** – True to add a counter field to the event
- **counter_name** – The name of the counter field
- **reverse** – True to start the counter in reverse

Returns A list of events

Examples:

```
# Example #1
event = {'id': 1,
        'names': ['first', 'second'],
        '_metadata': {'uuid': '1a'}}
event =  split_field_list_to_multiple_events(event=input, fields_to_split=['names
↪'], add_counter=True, counter_name='counter', reverse=False)
event = [{'id': 1,
           'name': 'first',
           'counter': 1,
           '_metadata': {'@parent_uuid': '1a',
                        '@uuid': '456-def'}},
          {'id': 1,
           'name': 'second',
           'counter': 2,
           '_metadata': {'@parent_uuid': '1a',
                        '@uuid': '789-ghi'}}]
```

3.12.27 Whitelist or Blacklist Columns

`managealooma.transformation_functions.`**`whitelist_or_blacklist_columns`**(*event*, *field_list*, *white_or_black_list='whitelist'*)

Allows you to remove a list of fields (blacklist) or limit an event to a list of fields (whitelist)

Parameters

- **`event`** – A dictionary
- **`field_list`** – A list of fields to keep or remove
- **`white_or_black_list`** – whitelist = Only let a particular list of columns through the event and remove other columns. blacklist = Don't allow a particular list of columns through. Leave all other columns

Returns An altered dictionary

Examples:

```
# Example #1
event = {'_metadata': {},
        'keep_me': 'i stay',
        'keep_me_too': 'i stay too',
        'remove_me': 'im gone'}
event = whitelist_or_blacklist_columns(event, field_list=['keep_me', 'keep_me_too
↪'], white_or_black_list='whitelist')
event = {'_metadata': {},
        'keep_me': 'i stay',
        'keep_me_too': 'i stay too'}

# Example #2
event = {'_metadata': {},
        'keep_me': 'i stay',
        'keep_me_too': 'i stay too',
        'remove_me': 'im gone'}
event = whitelist_or_blacklist_columns(event, field_list=['remove_me'], white_or_
↪black_list='blacklist')
event = {'_metadata': {},
        'keep_me': 'i stay',
        'keep_me_too': 'i stay too'}
```

3.13 Transformation Test

Alooma takes an event (a dictionary) and performs transformations on it. This class helps to test single events to inspect the transformation or run multiple samples through the tests for integration testing.

`TransformationTest.`**`__init__`**(*api=None*, *code_package=None*, *preview_full_events=True*, *preview_difference_dicts=True*, *local_or_api='local'*, *pprint_indent=2*, *pprint_width=250*, *pprint_depth=5*)

Tests the Alooma events

Parameters

- **`api`** – api authentication using the Alooma package
- **`preview`** – True to print the transformations for visual inspection

- **code_package** – Your specific transformation code to use as a package. This is needed for local testing
- **local_or_api** – 'local' to test on local code or 'api' to test on code deployed to Alooma
- **pprint_indent** – The indent value to pprint dictionaries
- **pprint_width** – The width value to pprint dictionaries

3.13.1 Test Single Event

TransformationTest.**test_single_event**(*sample*)

Tests a single event using the local code and pretty prints the before and after dictionaries to the console

Parameters **sample** – A single sample event dictionary

Returns The output dictionary

3.13.2 Loop Through Events to Test

TransformationTest.**loop_through_events_to_test**(*sample_event_list*)

Loops through the event files to test each sample

Parameters **sample_event_list** – A list of dictionaries to test. The expected format from Alooma is a dictionary as formatted below

Returns None

Alooma samples are in a list of dictionaries with the sample inside of a key of sample:

```
[{'sample': {}
{'sample': {}
{'sample': {}]
```

3.13.3 Test All Events

TransformationTest.**test_all_events**(*sample_event_directory*, *file_name*, *file_prefix='input'*)

Tests all events from a specified file or all the saved input sample.

Parameters

- **sample_event_directory** – The name of the directory where sample files are saved.
- **file_name** – The name of a specific file to test. A file should be a list with dictionaries.
- **file_prefix** – Specific a file prefix to test all the events in files with a similar name. The defaul will test all the samples for all inputs that have been saved using the Samples.save_alooma_samples_to_files function.

Returns True if the samples should be printed to the console

3.14 Credits

Thanks to my colleagues who helped write, review, and test the code and docs.

– Jay

3.14.1 Contributors

- Frances Fang
- Justin Grilli
- Jeremy Holtzman
- Beau Rothrock
- The Alooma Team

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index