groundwork-spreadsheets Documentation

Release 0.3.0

team useblocks

Aug 25, 2017

Contents

This is a groundwork extension package for reading and writing spreadsheet files.

groundwork is a plugin based Python application framework, which can be used to create various types of applications: console scripts, desktop apps, dynamic websites and more.

Visit groundwork.useblocks.com or read the technical documentation for more information.

CHAPTER 1

Functions

ExcelValidationPattern

Target audience are users who need to read well structured Excel documents with minimal overhead. The Excel documents can be validated in various ways to detect input data problems.

- Uses the library openpyxl
- Can read Excel 2010 files (xlsx, xlsm)
- Configure your sheet using a json file
- Auto detect columns by names. You can move columns without affecting the read routines.
- The data layout can be
  - column based: headers are in a single *row* and the data is below
  - row based: headers are in a single *column* and the data is on the right
- Define column types and verify cell values against them
  - Date
  - Enums (e.g. only the values 'yes' and 'no' are allowed)
  - Floating point numbers with optional min/max check
  - Integer numbers with optional min/max check
  - String with optional regular expression pattern check
- Exclude data row/columns based on filter criteria
- Output is a dictionary of the following form `row or column number -> header name -> cell value`
- Extensive logging of problems

CHAPTER 2

Package content

Usage

An example for the usage can be found in the **example** folder in the root of the package.

Here is the content as a quick reference.

example.py

```
from groundwork_spreadsheets import ExcelValidationPattern
from groundwork import App


def Application():
    app = App(plugins=[], strict=True)
    return app


class ReadCustomExcel(ExcelValidationPattern):
    def __init__(self, app, name=None, *args, **kwargs):
        self.name = name or self.__class__.__name__
        super(ReadCustomExcel, self).__init__(app, *args, **kwargs)

    def activate(self):
        pass

    def deactivate(self):
        pass


if __name__ == '__main__':
    app = App(config_files=['configuration.py'], plugins=[], strict=True)
    plugin = ReadCustomExcel(app)
```

```
    data = plugin.excel_validation.read_excel('config.json', 'example.xlsx')
    for row in data:
        headers = data[row]
        for header in headers:
            print("Row {0}, Header '{1}': {2}".format(row, header, data[row][header]))
```

config.json

```
{
    "sheet_config": "active",
    "orientation": "column_based",
    "headers_index_config": {
        "row_index": {
            "first": "automatic",
            "last": "automatic"
        },
        "column_index": {
            "first": "automatic",
            "last": "automatic"
        }
    },
    "data_index_config": {
        "row_index": {
            "first": "automatic",
            "last": "automatic"
        },
        "column_index": {
            "first": "automatic",
            "last": "automatic"
        }
    },
    "data_type_config": [
        {
            "header": "Date",
            "fail_on_type_error": false,
            "fail_on_empty_cell": false,
            "fail_on_header_not_found": true,
            "type": {
                "base": "date"
            }
        },
        {
            "header": "Enum",
            "fail_on_type_error": true,
            "fail_on_empty_cell": true,
            "fail_on_header_not_found": true,
            "type": {
                "base": "enum",
                "enum_values": ["ape", "dog", "cat"],
                "filter": {
                    "whitelist_values": ["ape", "cat"]
                }
            }
        },
        {
            "header": "Float",
```

```
            "fail_on_type_error": true,
            "fail_on_empty_cell": true,
            "fail_on_header_not_found": true,
            "type": {
                "base": "float",
                "minimum": 1.1,
                "maximum": 334
            }
        },
        {
            "header": "Integer",
            "fail_on_type_error": true,
            "fail_on_empty_cell": true,
            "fail_on_header_not_found": true,
            "type": {
                "base": "integer",
                "minimum": -3,
                "maximum": 30
            }
        },
        {
            "header": "Text",
            "fail_on_type_error": true,
            "fail_on_empty_cell": true,
            "fail_on_header_not_found": true,
            "type": {
                "base": "string",
                "pattern": "^Text [0-9]$"
            }
        }
    ],
    "filter_properties": {
        "excluded_fail_on_empty_cell": false,
        "excluded_fail_on_type_error": false,
        "excluded_enable_logging": false
    }
}
```

Configuration

This is the base structure of the configuration .json file:

```
{
    "sheet_config": "active",
    "orientation": "column_based",
    "headers_index_config": {
        "row_index": {
            "first": "automatic",
            "last": "automatic"
        },
        "column_index": {
            "first": "automatic",
            "last": "automatic"
        }
    },
```

```
    "data_index_config": {
        "row_index": {
            "first": "automatic",
            "last": "automatic"
        },
        "column_index": {
            "first": "automatic",
            "last": "automatic"
        }
    },
    "data_type_config": [
        {
            "header": "Text Column",
            "type": {
                "base": "string",
                "convert_numbers": true,
                "pattern": "^Text [0-9]$"
            }
        },
        {
            "header": "Animals Column",
            "fail_on_type_error": true,
            "fail_on_empty_cell": true,
            "fail_on_header_not_found": true,
            "type": {
                "base": "enum",
                "enum_values": ["ape", "dog", "cat"],
                "filter": {
                    "whitelist_values": ["ape", "cat"]
                }
            }
        }
    ],
    "filter_properties": {
        "excluded_fail_on_empty_cell": false,
        "excluded_fail_on_type_error": false,
        "excluded_enable_logging": false
    }
}
```

sheet_config

This parameter configures which worksheet to choose. Only one worksheet can be read in one call.

Possible values are:

Value	Type	Example	Meaning
active	string	“sheet_config”: “active”	Chooses the active worksheet, that is the one that was active when last saving the workbook
first	string	“sheet_config”: “first”	Chooses the first worksheet
last	string	“sheet_config”: “last”	Chooses the last worksheet
name:<sheet_name>	string	“sheet_config”: “name:sheet2”	Chooses the worksheet with the name <sheet_name>
<index>	integer	“sheet_config”: 2	The index of the worksheet. The first sheet gets index 1.

orientation

This parameter specifies the layout of the worksheet.

Possible values are:

Value	Type	Example	Meaning
column_based	string	“orientation”: “column_based”	Each header and its data is in one column
row_based	string	“orientation”: “row_based”	Each header and its data is in one row

ASCII art for column_based

```
*******************************
* header1   header2   header3 *
*******************************
* value1    value1    value1  *
*                             *
* value2    value2    value2  *
*                             *
* value3    value3    value3  *
*******************************
```

ASCII art for row_based

```
*************************************
* header1 * value1   value2   value3 *
*         *                         *
* header2 * value1   value2   value3 *
*         *                         *
* header3 * value1   value2   value3 *
*************************************
```

headers_index_config

This dictionary specifies the location of the headers in the worksheet.

The dictionary always has 2 keys:

Value	Type	Meaning
row_index	dictionary	Defines the row cells for the headers in the chosen orientation.
column_index	dictionary	Defines the column cells for the headers in the chosen orientation.

Note: Note that all row and column indices are 1-based. That means the upper left cell of a worksheet is in row 1 and column 1, just like in Excel.

row_index

This dictionary specifies the row cells for the headers in the chosen orientation. In column based orientation, the row_index matrix spans several cells in one row. In row based orientation,

The dictionary always has 2 keys:

Value	Type	Meaning
first	string or integer	Defines the first header row.
last	string or integer	Defines the last header row.

first

Possible values are:

Value	Type	Example	Meaning
automatic	string	“first”: “automatic”	A default of 1 is chosen.
<row_index>	integer	“first”: 2	A manually integer value greater than 1.

Note: For column_based orientation, the first and last row must be identical if both are set manually.

last

Possible values are:

Value	Type	Example	Meaning
automatic	string	“last”: “automatic”	**column_based** The value of the entry in ‘first’ is chosen. If it’s set to ‘automatic’, 1 is chosen. **row_based** The worksheet dimensions are read by the library openpyxl. The greatest row index (of all rows) is chosen.
<row_index>	integer	“last”: 2	A manually set integer value not smaller than 1.
severalEmptyCells:<cell_count>	string	“last”: “severalEmptyCells:3”	**column_based** Same as ‘automatic’. The given <cell_count> value has no meaning. **row_based** The last header row will be chosen using a search algorithm. If, after a non-empty row, several (<cell_count>) directly following empty cells are found, the last non-empty row is considered the last row.

column_index

This dictionary specifies the column cells for the headers in the chosen orientation.

first

Possible values are:

Value	Type	Example	Meaning
automatic	string	“first”: “automatic”	A default of 1 is chosen.
<row_index>	integer	“first”: 2	A manually integer value greater than 1.

Note: For row_based orientation, the first and last row must be identical if both are set manually.

last

Possible values are:

Value	Type	Example	Meaning
automatic	string	“last”: “automatic”	**column_based** The worksheet dimensions are read by the library openpyxl. The greatest column index (of all columns) is chosen. **row_based** The value of the entry in ‘first’ is chosen. If it’s set to ‘automatic’, 1 is chosen.
<row_index>	integer	“last”: 2	A manually set integer value not smaller than 1.
severalEmptyCells:<cell_count>	string	“last”: “severalEmptyCells:3”	**column_based** The last header column will be chosen using a search algorithm. If, after a non-empty column, several (<cell_count>) directly following empty cells are found, the last non-empty column is considered the last column. **row_based** Same as ‘automatic’. The given <cell_count> value has no meaning.

data_index_config

This dictionary specifies the location of the data in the worksheet.

The dictionary always has 2 keys:

Value	Type	Meaning
row_index	dictionary	Defines the row cells for the data in the chosen orientation.
column_index	dictionary	Defines the column cells for the data in the chosen orientation.

Note: Note that all row and column indices are 1-based. That means the upper left cell of a worksheet is in row 1 and column 1 (==’A’), just like in Excel.

row_index

This dictionary specifies the row cells for the data in the chosen orientation.

The dictionary always has 2 keys:

Value	Type	Meaning
first	string or integer	Defines the first data row.
last	string or integer	Defines the last data row.

first

Possible values are:

Value	Type	Example	Meaning
automatic	string	“first”: “automatic”	**column_based** A default of <first_header_row> + 1 is chosen. That means the data comes directly after the headers. **row_based** The <first_header_row> is chosen.
<row_index>	integer	“first”: 2	A manually integer value greater than 1.

last

Possible values are:

Value	Type	Example	Meaning
automatic	string	“last”: “automatic”	**column_based** The worksheet dimensions are read by the library openpyxl. The greatest row index (of all rows) is chosen. **row_based** The value of the entry in ‘last header row’ is chosen or its automatically calculated value is taken.
<row_index>	integer	“last”: 2	A manually set integer value not smaller than 1.
severalEmptyCells:<cell_count>	string	“last”: “severalEmptyCells:3”	**column_based** The last data row will be chosen using a search algorithm. If, after a non-empty row, several (<cell_count>) directly following empty rows are found, the last non-empty row is considered the last row. A row is empty if all columns in that row are empty. **row_based** The value of the entry in ‘last header row’ is chosen or its automatically calculated value is taken.

column_index

This dictionary specifies the column cells for the data in the chosen orientation.

first

Possible values are:

Value	Type	Example	Meaning
automatic	string	“first”: “automatic”	**column_based** The <first_header_column> is chosen. **row_based** A default of <first_header_column> + 1 is chosen. That means the data comes directly after the headers.
<row_index>	integer	“first”: 2	A manually integer value greater than 1.

last

Possible values are:

Value	Type	Example	Meaning
automatic	string	“last”: “automatic”	**column_based** The value of the entry in ‘last header column’ is chosen or its automatically calculated value is taken. **row_based** The worksheet dimensions are read by the library openpyxl. The greatest column index (of all columns) is chosen.
<row_index>	integer	“last”: 2	A manually set integer value not smaller than 1.
severalEmptyCells:<cell_count>	string	“last”: “severalEmptyCells:3”	**column_based** The value of the entry in ‘last header column’ is chosen or its automatically calculated value is taken. **row_based** The last data column will be chosen using a search algorithm. If, after a non-empty column, several (<cell_count>) directly following empty columns are found, the last non-empty column is considered the last column. A column is empty if all rows in that column are empty.

data_type_config

This array specifies the data type for each header. The validation is done against this specification.

Possible base types are:

Value	Meaning
automatic	No validation is done. The value is passed as read by openpyxl.
date	A Python datetime.datetime instance. The validation fails if a cell does not contain a date.
enum	A set of allowed strings can be specified. The validation fails if a cell contains text which is not part of the allowed string list.
float	A floating point number is expected. Minimum and maximum can optionally be specified.
integer	An integer number is expected. Minimum and maximum can optionally be specified.
string	A string is expected. A regular expression pattern can optionally be specified. The Python re.search implementation is used.

Common parameters

Parameter	Type	Meaning
header	string	The name of the header. The worksheet will be searched for this name.
fail_on_type_error	bool	If true, a ValueError exception is raised if the type does not fit or a constraint fails. If false, just a log message (log level error) is dumped if the type does not fit or a constraint fails.
fail_on_empty_cell	bool	If true, a ValueError exception is raised if an empty cell is found. If false, just a log message (log level error) is dumped if an empty cell is found.
fail_on_header_not_found	bool	If true, a ValueError exception is raised if the corresponding header cannot be found in the spreadsheet. If false, just a log message (log level error) is dumped if the corresponding header cannot be found in the spreadsheet. The output data dictionary will not contain that header.

The field `header` is mandatory for all types.

The following fields are optional for all types. If not given, a default of `true` is chosen for these.

- fail_on_type_error
- fail_on_empty_cell

- fail_on_header_not_found

type

The field `base` is mandatory for all types.

automatic

Specification:

```
"type": {
    "base": "automatic"
}
```

date

Specification:

```
"type": {
    "base": "date"
}
```

enum

Specification:

```
"type": {
    "base": "enum",
    "enum_values": [<list_of_string_values>]
}
```

The `enum_values` field is mandatory.

The enum type supports filtering using a whitelist of enum values:

```
"type": {
    "base": "enum",
    "enum_values": [<list_of_string_values>],
    "filter": {
        "whitelist_values": [<list_of_allowed_values>]
    }
}
```

Within the `filter` property, the `whitelist_values` field is mandatory.

The target data will only returns data rows/columns containing the specified allowed values. It's possible to have filters on several enum types. In this case, only rows/columns are returned which are contained by both filters.

float

Specification:

```
"type": {
    "base": "float",
    "minimum": <min_value>,
    "maximum": <max_value>
}
```

The `minimum` field is optional. The `maximum` field is optional. If the minimum or maximum constraint fails, it will be handled as a type error (see *Common parameters*).

Note: For documents saved by MS Excel, openpyxl returns integer values with the 'float' data type (e.g. 33345.0). The ExcelValidationPattern checks if the float can be converted to int without precision loss (using 'value.is_integer()'). If yes a type cast to int is done, that means you can always expect the 'int' type. If no it is a type error.

---

integer

Specification:

```
"type": {
    "base": "integer",
    "minimum": <min_value>,
    "maximum": <max_value>
}
```

The `minimum` field is optional. The `maximum` field is optional. If the minimum or maximum constraint fails, it will be handled as a type error (see *Common parameters*).

Note: For Python 2.7, openpyxl returns integers with the 'long' data type. For Python 3, openpyxl returns text with the 'int' data type. Both are accepted by the above integer type. No type conversion is done by ExcelValidationPattern.

---

string

Specification:

```
"type": {
    "base": "string",
    "convert_numbers": <bool>,
    "pattern": "<regex_pattern>"
}
```

The `pattern` field is optional. If the pattern constraint fails, it will be handled as a type error (see *Common parameters*).

The `pattern` will be checked using the Python re.search routine. If you would like to check the whole cell value, use the anchors `^` and `$`.

Note: Both types 'unicode' and 'str' are accepted by above string type. No type conversion is done by ExcelValidationPattern.

The `convert_numbers` field is optional.

If it is **true**, the numeric types 'int', 'long' and 'float' are casted to a string using the str() routine. As a consequence, no type error will occur.

If it is **false**, the numeric types 'int', 'long' and 'float' lead to a type error. This is helpful in situation where the type can either be a number or text.

filter_properties

This dictionary specifies how filters affect errors of excluded rows/columns. Excluded rows/columns are commonly not of primary interest to the user, so it makes sense to mask errors that might arise there. The variables set here can overwrite the data type definitions.

Parameter	Type	Meaning
excluded_fail_on_empty_cell	bool	If **true**, an empty cell in an excluded row/column will **still raise an exception**. If **false**, an empty cell in an excluded row/column will **not raise an exception**.
excluded_fail_on_type_error	bool	If **true**, a type error in an excluded row/column will **still raise an exception**. If **false**, a type error in an excluded row/column will **not raise an exception**.
excluded_enable_logging	bool	If **true**, empty cell and type errors which are configured not to raise an exception, will still be logged. If **false**, empty cell and type errors which are configured not to raise an exception, will not be logged.

Example 1:

```
"filter_properties": {
    "excluded_fail_on_empty_cell": false,
    "excluded_fail_on_type_error": false,
    "excluded_enable_logging": false
}
```

In above case, errors in excluded rows/columns will neither raise an exception nor be logged.

Example 2:

```
"filter_properties": {
    "excluded_fail_on_empty_cell": true,
    "excluded_fail_on_type_error": false,
```

```
    "excluded_enable_logging": true
}
```

In above case, empty cell errors in excluded rows/columns will raise exceptions. Type errors in excluded rows/columns, however, will just be logged.

Changelog

0.3.0

- Added 'convert_numbers' key to string type. This accepts numbers for string types too.
- Added coveralls.io supprt
- Added scrutinizer-ci.com support
- Fixed pylint issues

0.2.0

- Added exclusion function for data row/columns based on filter criteria. Currently only enums whitelisting is supported.
- Expanded test cases with documents saved by MS Excel 2013

0.1.2

- Added debug log messages for automatic last row/column detection
- Fixed a small logging bug
- Example: Added a configuration file to enable logging
- Example: Added a readme
- Description: Some rewriting

0.1.1

- Added description in setup.py for PyPi

0.1.0

Initial version