databuild Documentation

Release 0.0.10

Flavio Curella

October 17, 2014

Contents

`Databuild` is an automation tool for data manipulation.

The general principles in `Databuild` are:

- Low entry barrier
- Easy to install
- Easy to grasp
- Extensible

`Databuild` can be useful for scenarios such as:

- Documenting data transformations in your infoviz project
- Automate data processing in a declarative way

CHAPTER 1

Contents

1.1 Installation

Install Databuild:

```
$ pip install databuild
```

1.2 Quickstart

Run databuild using a *buildfile*:

```
$ data-build.py buildfile.json
```

`buildfile.yaml` contains a list of operations to be performed on data. Think of it as a script for a spreadsheet.

An example of build file could be:

```
- operation: sheets.import_data
  description: Importing data from csv file
  params:
    sheet: dataset1
    format: csv
    filename: dataset1.csv
    skip_last_lines: 1
- operation: columns.add_column
  description: Calculate the gender ratio
  params:
    sheet: dataset1
    name: Gender Ratio
    expression:
      language: python
      content: "return float(row['Totale Maschi']) / float(row['Totale Femmine'])"
- operation: sheets.export_data
  description: save the data
  params:
    sheet: dataset1
    format: csv
    filename: dataset2.csv
```

JSON buildfiles are also supported. `Databuild` will guess the type based on the extension.

1.3 Philosophy

`Databuild` is an alternative to more complex and complete packages like `pandas`, `numpy`, and `R`.

It's aimed at users that are not necessarily data scientist and are looking to a simpler alternative to such softwares.

It's admittely less performant than those and is not optimized for huge datasets. But `Databuild` is much easier to get started with.

The general principles in `Databuild` are:

- Low entry barrier
- Easy to install
- Easy to grasp
- Extensible

Databuild can be useful for scenarios such as:

- Documenting data transformations in your infoviz project
- Automate data processing in a declarative way

1.4 Buildfiles

A `buildfile` contains a list of operations to be performed on data. Think of it as a script for a spreadsheet.

JSON and YAML format are supported. `databuild` will guess the format based on the file extension.

An example of build file could be:

```
[
  {
    "operation": "sheets.import_data",
    "description": "Importing data from csv file",
    "params": {
      "sheet": "dataset1",
      "format": "csv",
      "filename": "dataset1.csv",
      "skip_last_lines": 1
    }
  },
  {
    "operation": "columns.add_column",
    "description": "Calculate the gender ratio",
    "params": {
      "sheet": "dataset1",
      "name": "Gender Ratio",
      "expression": {
        "language": "python",
        "content": "return float(row['Male Total']) / float(row['Female Total'])"
      }
    }
  },
  {
    "operation": "sheets.export_data",
    "description": "save the data",
    "params": {
```

```
      "sheet": "dataset1",
      "format": "csv",
      "filename": "dataset2.csv"
    }
  }
]
```

The same file in yaml:

```
- operation: sheets.import_data
  description: Importing data from csv file
  params:
    sheet: dataset1
    format: csv
    filename: dataset1.csv
    skip_last_lines: 1
- operation: columns.add_column
  description: Calculate the gender ratio
  params:
    sheet: dataset1
    name: Gender Ratio
    expression:
      language: python
      content: "return float(row['Male Total']) / float(row['Female Totale'])"
- operation: sheets.export_data
  description: save the data
  params:
    sheet: dataset1
    format: csv
    filename: dataset2.csv
```

You can split a buildfile in different files, and have `databuild` process the directory that contains them.

Build files will be executed in alphabetical order. It's recommended that you name them starting with a number indicating their order of execution. For example:

```
-- buildfiles
    -- 1_import.yaml
    -- 2_add_column.yaml
    -- 3_export.yaml
    -- data
        -- dataset1.csv
```

1.5 Python API

`Databuild` can be integrated in your python project. Just import the `build` function:

```
from databuild.builder import Builder

Builder().build('buildfile.json')
```

Supported arguments:

- `build_file` Required. Path to the `buildfile`.
- `settings` Optional. Python module path containing the settings. Defaults to `datbuild.settings`
- `echo` Optional. Set this to `True` if you want the operations' description printed to the screen. Defaults to `False`.

1.6 Operation Functions

Operations functions are regular Python function that perform actions on the book. Examples of operations are: `sheets.import_data`, `columns.add_column`, `columns.update_column`, and more.

They have a `function` name that identifies them, an optional description and a number of parameters that they accept. Different operation functions accept different parameters.

All operations accepts the following arguments:

- `name`: The name idnetifying the operation function
- `description`: Optional. A description to print on screen when the operation gets executed
- `params`: Parameter for the operation. This varies by operation. See below for a detailed list.
- `context`: Optional. An additional set of properties that can be made available to the operation. This can be used in expression or interpolated in strings (by using the `{{ my var }}` sysntax).
- `enabled`: Optional. Set this to `False` to skip execution of the operation. Useful for debug. Defaults to `True`.

1.6.1 Available Operation Functions

`sheets.import_data`

Creates a new sheet importing data from an external source.

params:

- `filename`: Required.
- `sheet`: Optional. Defaults to `filename`'s basename.
- `format`: Values currently supported are `'csv'` and `'json'`.
- `headers`: Optional. Defaults to *null*, meaning that the importer tries to autodetects the header names.
- `encoding`: Optional. Defaults to `'utf-8'`.
- `skip_first_lines`: Optional. Defaults to `0`. Supported only by the CSV importer.
- `skip_Last_lines`: Optional. Defaults to `0`. Supported only by the CSV importer.
- `guess_types`: Optional. If set to `true`, the CSV importer will try to guess the data type. Defaults to `true`.
- `replace`: Optional. if set to `true` and the sheet already exists, its content will be replaced. if set to `false`, the new data will be appended. Defaults to `false`.

`sheets.add`

params:

- `name`
- `headers` (optional)

Adds a new empty sheet named `name`. Optionally sets its headers as specified in `headers`.

sheets.copy

params:

- source
- destination
- headers (optional)

Create a copy of the source sheet named destination. Optionally copies only the headers specified in headers.

sheets.export_data

params:

- sheet
- format
- filename
- headers (optional)

Exports the datasheet named sheet to the file named filename in the specified format. Optionally exports only the headers specified in headers.

sheets.print_data

params:

- sheet

columns.update_column

params:

- sheet
- column
- facets (optional)
- values
- expression

Either values or expression are required.

columns.add_column

params:

- sheet
- name
- values
- expression

Either `values` or `expression` are required.

columns.remove_column

params:

- `sheet`
- `name`

columns.rename_column

params:

- `sheet`
- `old_name`
- `new_name`

columns.to_float

params:

- `sheet`
- `column`
- `facets` (optional)

columns.to_integer

params:

- `sheet`
- `column`
- `facets` (optional)

columns.to_decimal

params:

- `sheet`
- `column`
- `facets` (optional)

columns.to_text

params:

- `sheet`
- `column`

- `facets` (optional)

columns.to_datetime

params:

- `sheet`
- `column`
- `facets` (optional)

system.call_command

Calls an external command with the provided arguments.

params:

- `command`
- `arguments`

system.remove_dir

Deletes a directory and all of its content.

params:

- `path`: The path of the directory. Either absolute or relative to the build file.

system.make_dir

Creates a directory recursively.

params:

- `path`: The path of the directory. Either absolute or relative to the build file.

operations.define_operation

Define an alias to an operation with default params that can be reused.

params:

- `name`: how you want to name your operation. This is name that you will use to call the operation later.
- `operation`: the original path of the operation
- `defaults`: values that will be used as defaults for the operation. You can override them by using the `params` property when you call your operation

operations.define_task

Define an a task, a list of operations with default params that can be reused.

params:

- `name`: How you want to name your task. This is name that you will use to call the task later.
- `operations`: The operations to be applied
- `description`: Optional. A description to be printed when calling the task.
- `defaults`: Optional. values that will be used as defaults for the operations. You can override them by using the `overrides` param when you call your task

operations.call_task

Call a previously defined task

params:

- `name`: the name your task. This is name that you will use to call the operation later.
- `operations`: the operations to be applied
- `description`: Optional. A description to be printed when calling the task.
- `overrides`: values that will override the defaults for the operations.

1.7 Custom Operation

You can add your custom operation and use them in your buildfile.

An Operation is just a regular python function. The first arguments has to be the `context`, but the remaining arguments will be pulled in from the `params` property of the operation in the buildfile.

By default, `context` is a `dict` with following keys:

- `workbook`: a reference the workbook object
- `buildfile`: a reference to the build file the operation has been read from.

```
def myoperation(context, foo, bar, baz):
    pass
```

Operations are defined in modules, which are just regulare Python files.

As long as your operation modules are in your `PYTHONPATH`, you can add them to your `OPERATION_MODULES` setting (see *operation-modules-setting*) and then call the operation in your buildfile by referencing its import path:

```
[
    ...,
    {
        "operation": "mymodule.myoperation",
        "description": "",
        "params": {
            "foo": "foos",
            "bar": "bars",
            "baz": "bazes"
        }
```

```
    }
]
```

1.8 Expressions

Expressions are objects encapsulating code for situations such as filtering or calculations.

An expression has the following properties:

- `language`: The name of the environment where the expression will be executed, as specified in `settings.LANGUAGES`. See *LANGUAGES*).
- `content`: The actual code to run, or
- `path`: path to a file containing the code to run

The expression will be evaluated inside a function and run against every row in the datasheet. The following context variables will be avalaible:

```
* ``row``: A dictionary representing the currently selected row.
```

1.9 Environments

Expressions are evaluated in the environment specified by their `language` property.

The value maps to a specific environment as specified in `settings.LANGUAGES` (See the *LANGUAGES* setting).

1.9.1 Included Environments

Currently, the following environments are shipped with `databuild`:

Python

Unsafe Python environment. Use only with trusted build files.

1.9.2 Writing Custom Environments

An `Environment` is a subclass of `databuild.environments.base.BaseEnvironment` that implements the following methods:

- `__init__(self, book)`: Initializes the environment with the appropriate global variables.
- `copy(self, iterable)`: Copies a variable from the databuild process to the hosted environment.
- `eval(self, expression, context)`: Evaluates the string *expression* to an actual functions and returns it.

1.9.3 Add-on Environments

Lua

An additional Lua environment is available at http://github.com/databuild/databuild-lua

Requires `Lua` or `LuaJIT` (Note: `LuaJIT` is currently unsupported on OS X).

1.10 Functions

Functions are additional methods that can be used inside *Expressions*.

1.10.1 Available Functions

`cross`

Returns a single value from a column in a different sheet.

arguments:

- `row` reference to the current row
- `sheet_source` name of the sheet that you want to get the data from
- `column_source` name of the column that you want to get the data from
- `column_key` name of the sheet that you want to match the data between the sheets.

`column`

Returns an array of values from column from a different dataset, ordered as the key.

arguments:

- `sheet_name` name of the current sheet
- `sheet_source` name of the sheet that you want to get the data from
- `column_source` name of the column that you want to get the data from
- `column_key` name of the sheet that you want to match the data between the sheets.

1.10.2 Custom Functions Modules

You can write your own custom functions modules.

A function module is a regulare Python module containing Python functions with the following signature:

```
def myfunction(environment, book, **kwargs)
```

Function must accept the `environment` and `book` positional arguments. After them, everything other argument is up the the function.

Another reuqirement is that the function must return a value wrapped into the environment's copy method:

```
return environment.copy(my_return_value)
```

Function modules must be made available by adding them to the `FUNCTION_MODULES` *Settings* variable.

1.11 Settings

1.11.1 ADAPTER

Classpath of the adapter class. Defaults to `'databuild.adapters.locmem.models.LocMemBook'`.

1.11.2 LANGUAGES

A `dict` mapping languages to *Environments*. Default to:

```
LANGUAGES = {
    'python': 'databuild.environments.python.PythonEnvironment',
}
```

1.11.3 FUNCTION_MODULES

A `tuple` of module paths to import *Functions* from. Defaults to:

```
FUNCTION_MODULES = (
    'databuild.functions.data',
)
```

1.11.4 OPERATION_MODULES

A `tuple` of module paths to import *Operation Functions* from. Defaults to:

```
OPERATION_MODULES = (
    "databuild.operations.sheets",
    "databuild.operations.columns",
    "databuild.operations.operations",
)
```

CHAPTER 2

Indices and tables

- *genindex*
- *modindex*
- *search*