dagstd

Isaac Harris-Holt

Jan 01, 2023

CONTENTS:

CHAPTER

ONE

DAGSTD

Dagstd is a Python package containing a set of helper modules for use with the Dagster data orchestration tool.

Dagster is a great tool, but there are occasions where you just need to pass in a simple integer or string as input to a Dagster op, but in Dagster, inputs to ops can only be outputs of other ops. This results in a lot of boilerplate functions being written that just return a formatted string or even just an integer. This is why Dagstd was created.

1.1 Features

- Simple ops for common numbers
- Constant value ops
- Helper ops for mathematical and string operations
- Ops for retrieving environment variables
- Sphinx autodoc support for Dagster ops

1.2 Usage

Here's an example of a pure-Dagster graph that downloads a daily zip file and extracts a known file name. Note: the `download_large_file` op has been omitted for brevity.

```
import zipfile

from datetime import datetime

from dagster import op, job


@op
def get_todays_date() -> str:
    return datetime.today().strftime()


@op
def five() -> int:
    return 5
```

(continues on next page)

(continued from previous page)

```
@op
def get_download_file_url(date: str) -> str:
    return f'https://example.com/{date}.csv'


@op
def get_nth_file_name(n: int) -> str:
    return f'file_{n:02}.txt'


@op
def extract_file_from_zip(context, zip_path: str, file_name: str) -> str:
    with zipfile.ZipFile(zip_path) as zip_file:
        with(f'/tmp/{file_name}', 'wb') as f:
            f.write(zip_file.read(file_name))
        context.log.info(f'Extracted {file_name} from {zip_path}')
        return f'/tmp/{file_name}'


@job
def process_data():
    date = get_todays_date()
    url = get_download_file_url(date)
    zip_path = download_large_file(url)

    file_name = get_nth_file_name(five())
    file_path = extract_file_from_zip(zip_path, file_name)
```

And here's the same graph, but with Dagstd ops.

```
import zipfile

from datetime import datetime

from dagster import op, job
from dagstd.constants import Constant, Five
from dagstd.operations import fmt


@op
def get_todays_date_string() -> str:
    return datetime.today().strftime("%Y-%m-%d")


@op
def extract_file_from_zip(context, zip_path: str, file_name: str) -> str:
    with zipfile.ZipFile(zip_path) as zip_file:
        with(f'/tmp/{file_name}', 'wb') as f:
            f.write(zip_file.read(file_name))
        context.log.info(f'Extracted {file_name} from {zip_path}')
        return f'/tmp/{file_name}'
```

(continues on next page)

(continued from previous page)

```
@job
def process_data():
    date = get_todays_date_string()
    url = fmt(Constant('https://example.com/{}.csv'), [date])
    zip_path = download_large_file(url)

    file_name = fmt(Constant('file_{}.txt'), [Five()])
    file_path = extract_file_from_zip(zip_path, file_name)
```

This was just a small example, but it serves to show how much boilerplate can be avoided when using Dagstd.

1.3 Sphinx Autodoc Plugin

Dagstd includes a Sphinx autodoc plugin that can be used to generate documentation for Dagster ops. To use the autodoc plugin, add the following to your `conf.py` file:

```
extensions = [
    ...
    'dagstd.sphinx.parser',
]
```

By default, this will prefix all op documentation with `(op)`. To change this, add the following to your `conf.py` file:

```
dagstd_op_prefix = 'My Op'
```

1.4 Documentation

Documentation can be found at https://dagstd.readthedocs.io/en/latest/readme.html.

1.5 Installation

Install Dagstd with pip:

```
pip install dagstd
```

1.6 Dependencies

- dagster >= 0.14.17

1.7 Contribute

I'm always looking for more ideas to add to Dagstd. If you have an idea, please open an issue or pull request, or message me on GitHub.

- Issue Tracker: https://github.com/isaacharrisholt/dagstd/issues
- Source Code: https://github.com/isaacharrisholt/dagstd

1.8 Support

If you are having issues, please let me know.

1.9 License

The project is licensed under the GNU GPLv3 license.

CHAPTER

TWO

DAGSTD PACKAGE

2.1 Subpackages

2.1.1 dagstd.constants package

Submodules

dagstd.constants.constant module

constant.py contains a Constant function that acts as an op that returns whatever value is passed to it.

dagstd.constants.constant.**Constant**(*value: Any*, *suffix: Optional[str] = None*, *make_unique: bool = True*) → Any

Acts as an op that returns whatever value is passed to it on creation.

Parameters

- **value** (*Any*) – The value to return.
- **suffix** (*str, default=None*) – A suffix to append to the op name.
- **make_unique** (*bool, default=True*) – If True, the op name will be suffixed with a UUID. Is overridden if the suffix argument is provided.
- **The name of the op is the value passed to it plus a unique ID to allow**
- **having multiple constants with the same value. Alternatively, you can**
- **pass a suffix to the Constant function to be used instead of the unique ID.**

dagstd.constants.numbers module

numbers.py contains ops that return common numbers. These ops are capitalised to make them seem more like classes, which helps with readability.

(op)dagstd.constants.numbers.**Eight**() → int

(op)dagstd.constants.numbers.**Fifty**() → int

(op)dagstd.constants.numbers.**Five**() → int

(op)dagstd.constants.numbers.**FiveHundred**() → int

(op)dagstd.constants.numbers.**Forty**() → int

(op)dagstd.constants.numbers.**Four**() → int

(op)dagstd.constants.numbers.**Nine**() → int

(op)dagstd.constants.numbers.**One**() → int

(op)dagstd.constants.numbers.**OneHundred**() → int

(op)dagstd.constants.numbers.**OneThousand**() → int

(op)dagstd.constants.numbers.**Seven**() → int

(op)dagstd.constants.numbers.**Six**() → int

(op)dagstd.constants.numbers.**Ten**() → int

(op)dagstd.constants.numbers.**Thirty**() → int

(op)dagstd.constants.numbers.**Three**() → int

(op)dagstd.constants.numbers.**Twenty**() → int

(op)dagstd.constants.numbers.**Two**() → int

(op)dagstd.constants.numbers.**Zero**() → int

Module contents

2.1.2 dagstd.operations package

Submodules

dagstd.operations.maths module

maths.py contains ops that perform common mathematical operations.

(op)dagstd.operations.maths.**add**(*args: List*)
Adds the given arguments.

(op)dagstd.operations.maths.**divide**(*x*, *args: List*)
Divides the first argument by the given arguments.

(op)dagstd.operations.maths.**multiply**(*args: List*)
Multiplies the given arguments.

(op)dagstd.operations.maths.**subtract**(*x*, *args: List*)
Subtracts the given arguments from the first argument.

dagstd.operations.strings module

strings.py contains helper ops for working with strings.

(op)dagstd.operations.strings.**fmt**(*string: str*, *args: List*) → str

Formats a string with the given arguments.

Parameters

- **string** (*str*) – The string to format.
- **args** (*List*) – The arguments to format into the string.

Returns
The formatted string.

Return type
str

Module contents

2.1.3 dagstd.sphinx package

Submodules

dagstd.sphinx.parser module

class dagstd.sphinx.parser.**OpDirective**(*name*, *arguments*, *options*, *content*, *lineno*, *content_offset*, *block_text*, *state*, *state_machine*)

Bases: PyFunction

Sphinx op directive.

get_signature_prefix(*sig*)

May return a prefix to put before the object name in the signature.

class dagstd.sphinx.parser.**OpDocumenter**(*directive: DocumenterBridge*, *name: str*, *indent: str = ''*)

Bases: FunctionDocumenter

Document op definitions.

classmethod can_document_member(*member*, *membername*, *isattr*, *parent*)

Called to see if a member can be documented by this Documenter.

check_module()

Check if *self.object* is really defined in the module given by *self.modname*.

document_members(*all_members=False*)

Generate reST for member documentation.

If *all_members* is True, document all members, else those given by *self.options.members*.

format_args(***kwargs: Any*)

Format the argument signature of *self.object*.

Should return None if the object does not have a signature.

member_order = 11

order if autodoc_member_order is set to 'groupwise'

objtype = 'op'

name by which the directive is called (auto...) and the default generated directive name

dagstd.sphinx.parser.**setup**(*app*)

Setup Sphinx extension.

Module contents

2.2 Submodules

2.3 dagstd.env module

env.py contains ops for working with environment variables.

(op)dagstd.env.**or_default**(*key: str*, *default: str*) → str

Returns the value of the environment variable with the given key, or the given default if the variable is not set.

(op)dagstd.env.**or_empty**(*key: str*) → str

Returns the value of the environment variable with the given key, or an empty string if the variable is not set.

(op)dagstd.env.**or_none**(*key: str*) → Union[str, NoneType]

Returns the value of the environment variable with the given key, or None if the variable is not set.

(op)dagstd.env.**or_raise**(*key: str*) → str

Returns the value of the environment variable with the given key, or raises an error if the variable is not set.

2.4 dagstd.utils module

utils.py contains useful helper ops that are not specific to any part of the Dagstd library.

(op)dagstd.utils.**no_op**(*value: Any*) → Any

Does nothing. Allows unlinked dependencies for graphs.

Examples

In this example, even though graph_2 has no data dependencies on graph_1, it will still wait for graph_1 to complete before starting due to the use of the *no_op* operation.

This is useful when you have graphs that need to activate in sequence, but the earlier graphs don't pass data. For example, you might run a data sync with Airbyte, and then run a dbt project to transform that data in your warehouse.

```
from dagster import graph, job, In

from dagstd.constants import Constant, One, Two
from dagstd.operations import add, fmt
from dagstd.utils import no_op

@graph
def graph_1():
```

(continues on next page)

(continued from previous page)

```
    name = Constant('Isaac')
    age = Constant(20)
    return fmt(Constant('{} is {} years old'), name, age)

@graph(ins={'arg_1': In(int), 'arg_2': In(int), 'deps': In(Any)})
def graph_2(arg_1, arg_2, deps):
    no_op(deps)
    return add([arg_1, arg_2])


@job
def my_job():
    string = graph_1()
    num = graph_2(One(), Two(), deps=string)
```

2.5 Module contents

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

d

INDEX

S

T

Z