
argschema Documentation

Release 1.1.1

Forrest Collman, David Feng

Feb 22, 2019

Contents

1	The User Guide	3
1.1	User Guide	3
1.2	Indices and tables	13
2	API	15
2.1	argschema package	15
3	TESTS	35
3.1	test	35
3.2	fields package	42
4	Indices and tables	47
4.1	Support/Contribute	47
4.2	License	47
	Python Module Index	49

This python module enables python programs to specify and validate their input parameters via a schema, while allowing those parameters to be passed into it in different ways in different contexts.

In particular it will allow you to

1. Specify an input_json file which contains the parameters via the command line
2. OR pass a dictionary directly into the module with the parameters defined
3. AND/OR pass individual parameters via the command line, in a way that will override the input_json or the dictionary given.

In all cases, it will merge these different parameters into a single dictionary and then validate the parameters against your schema.

This is where you should start to understand how to use argschema

1.1 User Guide

1.1.1 Installation

install via source code

```
$ python setup.py install
```

or pip

```
$ pip install argschema
```

1.1.2 Your First Module

Listing 1: mymodule.py

```
import argschema

class MySchema(argschema.ArgSchema):
    a = argschema.fields.Int(default=42, description='my first parameter')

if __name__ == '__main__':
    mod = argschema.ArgSchemaParser(schema_type=MySchema)
    mod.logger.warn("this module does nothing useful")
    print(mod.args)
```

running this code produces

```
$ python mymodule.py
Traceback (most recent call last):
  File "mymodule.py", line 9, in <module>
    mod = argschema.ArgSchemaParser(schema_type=MySchema)
  File "build/bdist.linux-x86_64/egg/argschema/argschema_parser.py", line 108, in __
↪init__
  File "build/bdist.linux-x86_64/egg/argschema/argschema_parser.py", line 256, in_
↪load_schema_with_defaults
  File "build/bdist.linux-x86_64/egg/argschema/utils.py", line 433, in load
AttributeError: 'module' object has no attribute 'EXCLUDE'
```

```
$ python mymodule.py --a 2
Traceback (most recent call last):
  File "mymodule.py", line 9, in <module>
    mod = argschema.ArgSchemaParser(schema_type=MySchema)
  File "build/bdist.linux-x86_64/egg/argschema/argschema_parser.py", line 108, in __
↪init__
  File "build/bdist.linux-x86_64/egg/argschema/argschema_parser.py", line 256, in_
↪load_schema_with_defaults
  File "build/bdist.linux-x86_64/egg/argschema/utils.py", line 433, in load
AttributeError: 'module' object has no attribute 'EXCLUDE'
```

```
$ python mymodule.py --a 2 --log_level WARNING
Traceback (most recent call last):
  File "mymodule.py", line 9, in <module>
    mod = argschema.ArgSchemaParser(schema_type=MySchema)
  File "build/bdist.linux-x86_64/egg/argschema/argschema_parser.py", line 108, in __
↪init__
  File "build/bdist.linux-x86_64/egg/argschema/argschema_parser.py", line 256, in_
↪load_schema_with_defaults
  File "build/bdist.linux-x86_64/egg/argschema/utils.py", line 433, in load
AttributeError: 'module' object has no attribute 'EXCLUDE'
```

```
$ python mymodule.py -h
usage: mymodule.py [-h] [--a A] [--log_level LOG_LEVEL]
                  [--input_json INPUT_JSON] [--output_json OUTPUT_JSON]
                  [--output_json_indent OUTPUT_JSON_INDENT]

optional arguments:
  -h, --help            show this help message and exit

MySchema:
  --a A                  my first parameter (default=42)
  --log_level LOG_LEVEL  set the logging level of the module (default=ERROR)

JsonInputConfigSchema:
  --input_json INPUT_JSON
                        filepath to input_json (REQUIRED)

JsonOutputConfigSchema:
  --output_json OUTPUT_JSON
                        filepath to save output_json (REQUIRED)
  --output_json_indent OUTPUT_JSON_INDENT
                        whether to indent options or not
```

Great you are thinking, that is basically argparse, congratulations!

But there is more.. you can also give your module a dictionary in an interactive session

```
>>> from argschema import ArgSchemaParser
>>> from mymodule import MySchema
>>> d = {'a':5}
>>> mod = ArgSchemaParser(input_data=d, schema_type=MySchema)
>>> print(mod.args)
{'a': 5, 'log_level': u'ERROR'}
```

or you write out a json file and pass it the path on the command line

Listing 2: myinput.json

```
{
  "a":99
}
```

```
$ python mymodule.py --input_json myinput.json
Traceback (most recent call last):
  File "mymodule.py", line 9, in <module>
    mod = argschema.ArgSchemaParser(schema_type=MySchema)
  File "build/bdist.linux-x86_64/egg/argschema/argschema_parser.py", line 94, in __
    ↪init__
  File "build/bdist.linux-x86_64/egg/argschema/argschema_parser.py", line 171, in __
    ↪get_input_data_from_config
  File "build/bdist.linux-x86_64/egg/argschema/sources/source.py", line 137, in get_
    ↪input_from_config
  File "build/bdist.linux-x86_64/egg/argschema/sources/source.py", line 100, in get_
    ↪config
AttributeError: 'module' object has no attribute 'EXCLUDE'
```

or override a parameter if you want

```
$ python mymodule.py --input_json myinput.json --a 100
Traceback (most recent call last):
  File "mymodule.py", line 9, in <module>
    mod = argschema.ArgSchemaParser(schema_type=MySchema)
  File "build/bdist.linux-x86_64/egg/argschema/argschema_parser.py", line 94, in __
    ↪init__
  File "build/bdist.linux-x86_64/egg/argschema/argschema_parser.py", line 171, in __
    ↪get_input_data_from_config
  File "build/bdist.linux-x86_64/egg/argschema/sources/source.py", line 137, in get_
    ↪input_from_config
  File "build/bdist.linux-x86_64/egg/argschema/sources/source.py", line 100, in get_
    ↪config
AttributeError: 'module' object has no attribute 'EXCLUDE'
```

plus, no matter how you give it parameters, they will always be validated, before any of your code runs.

Whether from the command line

```
$ python mymodule.py --input_json ../examples/myinput.json --a 5!
Traceback (most recent call last):
  File "mymodule.py", line 9, in <module>
    mod = argschema.ArgSchemaParser(schema_type=MySchema)
  File "build/bdist.linux-x86_64/egg/argschema/argschema_parser.py", line 94, in __
    ↪init__
```

(continues on next page)

(continued from previous page)

```

File "build/bdist.linux-x86_64/egg/argschema/argschema_parser.py", line 171, in __
↳get_input_data_from_config
File "build/bdist.linux-x86_64/egg/argschema/sources/source.py", line 137, in get_
↳input_from_config
File "build/bdist.linux-x86_64/egg/argschema/sources/source.py", line 100, in get_
↳config
AttributeError: 'module' object has no attribute 'EXCLUDE'

```

or from a dictionary

```

>>> from argschema import ArgSchemaParser
>>> from mymodule import MySchema
>>> d={'a':'hello'}
>>> mod = ArgSchemaParser(input_data=d, schema_type=MySchema, args=[])
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
File "/Users/forrestcollman/argschema/argschema/argschema_parser.py", line 159,
↳in __init__
    raise mm.ValidationError(json.dumps(result.errors, indent=2))
marshmallow.exceptions.ValidationError: {
  "a": [
    "Not a valid integer."
  ]
}

```

1.1.3 Fields

argschema uses marshmallow (<http://marshmallow.readthedocs.io/>) under the hood to define the parameters schemas. It comes with a basic set of fields that you can use to define your schemas. One powerful feature of Marshmallow is that you can define custom fields that do arbitrary validation. *argschema.fields* contains all the built-in marshmallow fields, but also some useful custom ones, such as *argschema.fields.InputFile*, *argschema.fields.OutputFile*, *argschema.fields.InputDir* that validate that the paths exist and have the proper permissions to allow files to be read or written.

Other fields, such as *argschema.fields.NumpyArray* will deserialize ordered lists of lists directly into a numpy array of your choosing.

Finally, an important Field to know is *argschema.fields.Nested*, which allows you to define heirarchical nested structures. Note, that if you use Nested schemas, your Nested schemas should subclass *argschema.schemas.DefaultSchema* in order that they properly fill in default values, as *marshmallow.Schema* does not do that by itself.

Another common question about *Nested* is how you specify that you want it not to be required, but want it filled in with whatever default values exist in the schema it references. Or alternatively, that you want it not required, and you only want the default values used if there is any reference in the input dictionary. The key to this distinction is including `default={}` (which will cause defaults of the subschemas to be filled in) vs leaving default unspecified, which will only trigger the subschema defaults if the original input contains any references to elements of that subschema.

This example illustrates the difference in the approaches

Listing 3: nested_example.py

```
import argschema
```

(continues on next page)

(continued from previous page)

```

class MyNest(argschema.schemas.DefaultSchema):
    a = argschema.fields.Int(default=1)
    b = argschema.fields.Int(default=2)

class MySchemaFill(argschema.ArgSchema):
    nest = argschema.fields.Nested(MyNest,
                                   required=False,
                                   default={},
                                   description='nested schema that fills in defaults')

class MySchema(argschema.ArgSchema):
    nest = argschema.fields.Nested(MyNest,
                                   required=False,
                                   description='nested schema that does not always_
↳fill defaults')

mod = argschema.ArgSchemaParser(schema_type=MySchema)
print('MySchema')
print(mod.args)
mod2 = argschema.ArgSchemaParser(schema_type=MySchemaFill)
print('MySchemaFill')
print(mod2.args)

```

```

$ python nested_example.py
Traceback (most recent call last):
  File "nested_example.py", line 22, in <module>
    mod = argschema.ArgSchemaParser(schema_type=MySchema)
  File "build/bdist.linux-x86_64/egg/argschema/argschema_parser.py", line 108, in __
↳init__
  File "build/bdist.linux-x86_64/egg/argschema/argschema_parser.py", line 256, in _
↳load_schema_with_defaults
  File "build/bdist.linux-x86_64/egg/argschema/utils.py", line 433, in load
AttributeError: 'module' object has no attribute 'EXCLUDE'

```

```

$ python nested_example.py --nest.a 4
Traceback (most recent call last):
  File "nested_example.py", line 22, in <module>
    mod = argschema.ArgSchemaParser(schema_type=MySchema)
  File "build/bdist.linux-x86_64/egg/argschema/argschema_parser.py", line 108, in __
↳init__
  File "build/bdist.linux-x86_64/egg/argschema/argschema_parser.py", line 256, in _
↳load_schema_with_defaults
  File "build/bdist.linux-x86_64/egg/argschema/utils.py", line 433, in load
AttributeError: 'module' object has no attribute 'EXCLUDE'

```

One important use case for *Nested*, is where you want your json to have a list of dictionaries. You might be tempted to use the field *List*, with a field_type of *Dict*, however you should use *Nested* with *many=True*.

The `template_module` example shows how you might combine these features to define a more complex parameter structure.

Listing 4: template_module.py

```
from argschema import ArgSchemaParser, ArgSchema
from argschema.fields import NumpyArray, Boolean, Int, Str, Nested
from argschema.schemas import DefaultSchema
import numpy as np
import pprint as pp

# these are the core parameters for my module

class MyNestedParameters(DefaultSchema):
    name = Str(required=True, description='name of vector')
    increment = Int(required=True, description='value to increment')
    array = NumpyArray(dtype=np.float, required=True,
                       description='array to increment')
    write_output = Boolean(required=False, default=True)

# but i'm going to nest them inside a subsection called inc

class MyParameters(ArgSchema):
    inc = Nested(MyNestedParameters)

# this is another schema we will use to validate and deserialize our output
class MyOutputParams(DefaultSchema):
    name = Str(required=True, description='name of vector')
    inc_array = NumpyArray(dtype=np.float, required=True,
                           description='incremented array')

if __name__ == '__main__':

    # this defines a default dictionary that will be used if input_json is not
    # specified
    example_input = {
        "inc": {
            "name": "from_dictionary",
            "increment": 5,
            "array": [0, 2, 5],

            "write_output": True
        },
        "output_json": "output_dictionary.json"
    }

    # here is my ArgSchemaParser that processes my inputs
    mod = ArgSchemaParser(input_data=example_input,
                           schema_type=MyParameters,
                           output_schema_type=MyOutputParams)

    # pull out the inc section of the parameters
    inc_params = mod.args['inc']
```

(continues on next page)

(continued from previous page)

```

# do my simple addition of the parameters
inc_array = inc_params['array'] + inc_params['increment']

# define the output dictionary
output = {
    'name': inc_params['name'],
    'inc_array': inc_array
}

# if the parameters are set as such write the output
if inc_params['write_output']:
    mod.output(output)

pp.pprint(mod.args)

```

so now if run the example commands found in run_template.sh

Listing 5: input.json

```

{
  "inc": {
    "name": "from_json",
    "increment": 1,
    "array": [3, 2, 1],
    "write_output": true
  }
}

```

```

$ python template_module.py
--output_json output_command.json
--inc.name from_command
--inc.increment 2
Traceback (most recent call last):
  File "template_module.py", line 50, in <module>
    output_schema_type=MyOutputParams)
  File "build/bdist.linux-x86_64/egg/argschema/argschema_parser.py", line 103, in __
↪init__
  File "build/bdist.linux-x86_64/egg/argschema/argschema_parser.py", line 138, in __
↪get_output_sink_from_config
  File "build/bdist.linux-x86_64/egg/argschema/sources/source.py", line 100, in get_
↪config
AttributeError: 'module' object has no attribute 'EXCLUDE'

```

```

$ python template_module.py
--input_json input.json
--output_json output_fromjson.json
Traceback (most recent call last):
  File "template_module.py", line 50, in <module>
    output_schema_type=MyOutputParams)
  File "build/bdist.linux-x86_64/egg/argschema/argschema_parser.py", line 94, in __
↪init__
  File "build/bdist.linux-x86_64/egg/argschema/argschema_parser.py", line 171, in __
↪get_input_data_from_config
  File "build/bdist.linux-x86_64/egg/argschema/sources/source.py", line 137, in get_
↪input_from_config
  File "build/bdist.linux-x86_64/egg/argschema/sources/source.py", line 100, in get_
↪config

```

(continues on next page)

(continued from previous page)

```
AttributeError: 'module' object has no attribute 'EXCLUDE'
```

```
$ python template_module.py
Traceback (most recent call last):
  File "template_module.py", line 50, in <module>
    output_schema_type=MyOutputParams)
  File "build/bdist.linux-x86_64/egg/argschema/argschema_parser.py", line 103, in __
  ↪init__
  File "build/bdist.linux-x86_64/egg/argschema/argschema_parser.py", line 138, in __
  ↪get_output_sink_from_config
  File "build/bdist.linux-x86_64/egg/argschema/sources/source.py", line 100, in get_
  ↪config
AttributeError: 'module' object has no attribute 'EXCLUDE'
```

1.1.4 Command-Line Specification

As mentioned in the section *Your First Module*, argschema supports setting arguments at the command line, along with providing arguments either in an input json or directly passing a dictionary as *input_data*. Values passed at the command line will take precedence over those passed to the parser or in the input json.

Arguments are specified with `--argument_name <value>`, where value is passed by the shell. If there are spaces in the value, it will need to be wrapped in quotes, and any special characters will need to be escaped with `\`. Booleans are set with `True` or `1` for true and `False` or `0` for false.

We can explore some typical examples of command line usage with the following script:

Listing 6: cli_example.py

```
from argschema import ArgSchema, ArgSchemaParser
from argschema.fields import List, NumpyArray, Bool, Int, Nested, Str
from argschema.schemas import DefaultSchema

class MyNestedSchema(DefaultSchema):
    a = Int(default=42, description="my first parameter")
    b = Bool(default=True, description="my boolean")

class MySchema(ArgSchema):
    array = NumpyArray(default=[[1, 2, 3], [4, 5, 6]], dtype="uint8",
                        description="my example array")
    string_list = List(List(Str,
                            default=[["hello", "world"], ["lists!"]],
                            description="list of lists of strings"))
    int_list = List(Int, default=[1, 2, 3],
                    description="list of ints")
    nested = Nested(MyNestedSchema, required=True)

if __name__ == '__main__':
    mod = ArgSchemaParser(schema_type=MySchema)
    print(mod.args)
```

```
$ python cli_example.py --help
usage: cli_example.py [-h] [--string_list STRING_LIST] [--int_list INT_LIST]
```

(continues on next page)

(continued from previous page)

```

        [--log_level LOG_LEVEL] [--array ARRAY]
        [--nested.a NESTED.A] [--nested.b NESTED.B]
        [--input_json INPUT_JSON] [--output_json OUTPUT_JSON]
        [--output_json_indent OUTPUT_JSON_INDENT]

optional arguments:
  -h, --help            show this help message and exit

MySchema:
  --string_list STRING_LIST
                        list of lists of strings (default=[['hello', 'world'],
                        ['lists!']])
  --int_list INT_LIST   list of ints (default=[1, 2, 3])
  --log_level LOG_LEVEL
                        set the logging level of the module (default=ERROR)
  --array ARRAY         my example array (default=[[1, 2, 3], [4, 5, 6]])

nested:
  --nested.a NESTED.A   my first parameter (default=42)
  --nested.b NESTED.B   my boolean (default=True)

JsonInputConfigSchema:
  --input_json INPUT_JSON
                        filepath to input_json (REQUIRED)

JsonOutputConfigSchema:
  --output_json OUTPUT_JSON
                        filepath to save output_json (REQUIRED)
  --output_json_indent OUTPUT_JSON_INDENT
                        whether to indent options or not

```

We can set some values and observe the output:

```

$ python cli_example.py --nested.b 0 --string_list "[['foo','bar'], ['baz','buz']]"
Traceback (most recent call last):
  File "cli_example.py", line 23, in <module>
    mod = ArgSchemaParser(schema_type=MySchema)
  File "build/bdist.linux-x86_64/egg/argschema/argschema_parser.py", line 108, in __
↳ init__
  File "build/bdist.linux-x86_64/egg/argschema/argschema_parser.py", line 256, in _
↳ load_schema_with_defaults
  File "build/bdist.linux-x86_64/egg/argschema/utils.py", line 433, in load
AttributeError: 'module' object has no attribute 'EXCLUDE'

```

If we try to set a field in a way the parser can't cast the variable (for example, having an invalid literal) we will see a casting validation error:

```

$ python cli_example.py --array [1,foo,3]
Traceback (most recent call last):
  File "cli_example.py", line 23, in <module>
    mod = ArgSchemaParser(schema_type=MySchema)
  File "build/bdist.linux-x86_64/egg/argschema/argschema_parser.py", line 82, in __
↳ init__
  File "build/bdist.linux-x86_64/egg/argschema/utils.py", line 163, in args_to_dict
marshmallow.exceptions.ValidationError: {
  "array": [

```

(continues on next page)

(continued from previous page)

```

    "Command-line argument can't cast to NumpyArray"
  ]
}

```

argschema does not support setting `Dict` at the command line.

1.1.5 Alternate Sources/Sinks

Json files are just one way that you might decide to serialize module parameters or outputs. Argschema by default provides json support because that is what we use most frequently at the Allen Institute, however we have generalized the concept to allow `argschema.ArgSchemaParser` to plugin alternative “sources” and “sinks” of dictionary inputs and outputs.

For example, `yaml` is another reasonable choice for storing nested key-value stores. `argschema.argschema_parser.ArgSchemaYamlParser` demonstrates just that functionality. So now `input_yaml` and `output_yaml` can be specified instead.

Furthermore, you can pass an `ArgSchemaParser` an `argschema.sources.ArgSource` object which implements a `get_dict` method, and any `argschema.ArgSchemaParser` will get its input parameters from that dictionary. Importantly, this is true even when the original module author didn’t explicitly support passing parameters from that mechanism, and the parameters will still be deserialized and validated in a uniform manner.

Similarly you can pass an `argschema.sources.ArgSink` object which implements a `put_dict` method, and `argschema.ArgSchemaParser.output` will output the dictionary however that `argschema.sources.ArgSink` specifies it should.

Finally, both `argschema.sources.ArgSource` and `argschema.sources.ArgSink` have a property called `ConfigSchema`, which is a `marshmallow.Schema` for how to deserialize the kwargs to it’s init class.

For example, the default `argschema.sources.json_source.JsonSource` has one string field of ‘input_json’. This is how `argschema.ArgSchemaParser` is told what keys and values should be read to initialize a `argschema.sources.ArgSource` or

`argschema.sources.ArgSink` instance.

So for example, if you wanted to define a `argschema.sources.ArgSource` which loaded a dictionary from a particular host, port and url, and a module which had a command line interface for setting that host port and url you could do so like this.

```

from argschema.sources import ArgSource, ArgSink
from argschema.schemas import DefaultSchema
from argschema.fields import Str, Int
from argschema import ArgSchemaParser
from test_classes import MySchema
import requests
try:
    from urllib.parse import urlunparse
except:
    from urlparse import urlunparse

class UrlSourceConfig(DefaultSchema):
    input_host = Str(required=True, description="host of url")
    input_port = Int(required=False, default=None, description="port of url")
    input_url = Str(required=True, description="location on host of input")
    input_protocol = Str(required=False, default='http', description="url protocol to_
↪use")

```

(continues on next page)

(continued from previous page)

```

class UrlSource(ArgSource):
    ConfigSchema = UrlSourceConfig

    def get_dict(self):
        if self.input_port is None:
            netloc = self.input_host
        else:
            netloc = "{}:{}".format(self.input_host, self.input_port)
        url = urlunparse((self.input_protocol, netloc, self.input_url, None, None, None))
        response = requests.get(url)
        return response.json()

class UrlArgSchemaParser(ArgSchemaParser):
    default_configurable_sources = [UrlSource]
    default_schema = MySchema

```

so now a `UrlArgSchemaParser` would expect command line flags of ‘`--input_host`’ and ‘`--input_url`’, and optionally ‘`--input_port`’, ‘`--input_protocol`’ (or look for them in `input_data`) and will look to download the json from that http location via requests. In addition, an existing `argschema.ArgSchemaParser` module could be simply passed a configured `UrlSource` via `input_source`, and it would get its parameters from there.

1.1.6 Sphinx Documentation

argschema comes with a autodocumentation feature for Sphinx which will help you automatically add documentation of your Schemas and `argschema.ArgSchemaParser` classes in your project. This is how the documentation of the *test* suite included here was generated.

To configure sphinx to use this function, you must be using the sphinx autodoc module and add the following to your `conf.py` file

```

from argschema.autodoc import process_schemas

def setup(app):
    app.connect('autodoc-process-docstring', process_schemas)

```

1.2 Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)

This contains the complete documentation of the api

2.1 argschema package

2.1.1 Subpackages

argschema.fields package

Submodules

argschema.fields.files module

marshmallow fields related to validating input and output file paths

```
class argschema.fields.files.InputDir (default=<marshmallow.missing>,          at-
                                     tribute=None, data_key=None, error=None, val-
                                     idate=None, required=False, allow_none=None,
                                     load_only=False, dump_only=False,
                                     missing=<marshmallow.missing>,          er-
                                     ror_messages=None, **metadata)
```

Bases: `marshmallow.fields.String`

`InputDir` is `marshmallow.fields.Str` subclass which is a path to a directory that exists and that the user can access (presently checked with `os.access`)

```
class argschema.fields.files.InputFile (default=<marshmallow.missing>,          at-
                                     tribute=None, data_key=None, error=None,
                                     validate=None, required=False, allow_none=None,
                                     load_only=False, dump_only=False,
                                     missing=<marshmallow.missing>,          er-
                                     ror_messages=None, **metadata)
```

Bases: `marshmallow.fields.String`

`InputDile` is a `marshmallow.fields.Str` subclass which is a path to a file location which can be read by the user (presently passes `os.path.isfile` and `os.access = R_OK`)

class `argschema.fields.files.OutputDir` (*mode=None, *args, **kwargs*)

Bases: `marshmallow.fields.String`

`OutputDir` is a `marshmallow.fields.Str` subclass which is a path to a location where this module will write files. Validation will check that the directory exists and create the directory if it is not present, and will fail validation if the directory cannot be created or cannot be written to.

Parameters

- **mode** (*str*) – mode to create directory
- ***args** – same as passed to `marshmallow.fields.Str`
- ****kwargs** – same as passed to `marshmallow.fields.Str`

class `argschema.fields.files.OutputFile` (*default=<marshmallow.missing>, attribute=None, data_key=None, error=None, validate=None, required=False, allow_none=None, load_only=False, dump_only=False, missing=<marshmallow.missing>, error_messages=None, **metadata*)

Bases: `marshmallow.fields.String`

`OutputFile` `marshmallow.fields.Str` subclass which is a path to a file location that can be written to by the current user (presently tested by opening a temporary file to that location)

`argschema.fields.files.validate_input_path` (*value*)

`argschema.fields.files.validate_outpath` (*path*)

argschema.fields.loglevel module

marshmallow fields related to setting logging levels

class `argschema.fields.loglevel.LogLevel` (***kwargs*)

Bases: `marshmallow.fields.String`

`LogLevel` is a field type that provides a setting for the loglevel of `python.logging`. This class will both validate the input and also *set* the input globally. In simple scenarios, a module will not have to do any manipulation of loglevel.

options = ['FATAL', 'CRITICAL', 'ERROR', 'WARN', 'WARNING', 'INFO', 'DEBUG']

argschema.fields.numpyarrays module

marshmallow fields related to reading in numpy arrays

class `argschema.fields.numpyarrays.NumpyArray` (*dtype=None, *args, **kwargs*)

Bases: `marshmallow.fields.List`

`NumpyArray` is a `marshmallow.fields.List` subclass which will convert any numpy compatible set of lists into a numpy array after deserialization and convert it back to a list when serializing,

Parameters **dtype** (*numpy.Dtype*) – dtype specifying the desired data type. if dtype is given the array will be converted to the type, otherwise numpy will decide what type it should be. (Default=None)

argschema.fields.slice module

class argschema.fields.slice.**Slice**(**kwargs)

Bases: `marshmallow.fields.String`

Slice is a `:class:'marshmallow.fields.Str'` field that supports a range or slice argument for selecting some subset of a larger dataset. The syntax is identical to numpy slicing. Examples: "10:20", "40", ":30", "10:2:40"

Parameters **kwargs** – the same as any `Str` receive

Module contents

sub-module for custom marshmallow fields of general utility

class argschema.fields.**Field**(default=<marshmallow.missing>, attribute=None, data_key=None, error=None, validate=None, required=False, allow_none=None, load_only=False, dump_only=False, missing=<marshmallow.missing>, error_messages=None, **metadata)

Bases: `marshmallow.base.FieldABC`

Basic field from which other fields should extend. It applies no formatting by default, and should only be used in cases where data does not need to be formatted before being serialized or deserialized. On error, the name of the field will be returned.

Parameters

- **default** – If set, this value will be used during serialization if the input value is missing. If not set, the field will be excluded from the serialized output if the input value is missing. May be a value or a callable.
- **attribute** (*str*) – The name of the attribute to get the value from when serializing. If *None*, assumes the attribute has the same name as the field.
- **data_key** (*str*) – The name of the key to get the value from when deserializing. If *None*, assumes the key has the same name as the field.
- **validate** (*callable*) – Validator or collection of validators that are called during deserialization. Validator takes a field's input value as its only parameter and returns a boolean. If it returns *False*, an `ValidationError` is raised.
- **required** – Raise a `ValidationError` if the field value is not supplied during deserialization.
- **allow_none** – Set this to *True* if *None* should be considered a valid value during validation/deserialization. If *missing=None* and *allow_none* is unset, will default to *True*. Otherwise, the default is *False*.
- **load_only** (*bool*) – If *True* skip this field during serialization, otherwise its value will be present in the serialized data.
- **dump_only** (*bool*) – If *True* skip this field during deserialization, otherwise its value will be present in the deserialized object. In the context of an HTTP API, this effectively marks the field as "read-only".
- **missing** – Default deserialization value for the field if the field is not found in the input data. May be a value or a callable.
- **error_messages** (*dict*) – Overrides for `Field.default_error_messages`.
- **metadata** – Extra arguments to be stored as metadata.

Changed in version 2.0.0: Removed *error* parameter. Use *error_messages* instead.

Changed in version 2.0.0: Added *allow_none* parameter, which makes validation/deserialization of *None* consistent across fields.

Changed in version 2.0.0: Added *load_only* and *dump_only* parameters, which allow field skipping during the (de)serialization process.

Changed in version 2.0.0: Added *missing* parameter, which indicates the value for a field if the field is not found during deserialization.

Changed in version 2.0.0: *default* value is only used if explicitly set. Otherwise, missing values inputs are excluded from serialized output.

Changed in version 3.0.0b8: Add *data_key* parameter for the specifying the key in the input and output data. This parameter replaced both *load_from* and *dump_to*.

context

The context dictionary for the parent *Schema*.

default_error_messages = {u'null': u'Field may not be null.', u'required': u'Missing
Default error messages for various kinds of errors. The keys in this dictionary are passed to *Field.fail*. The values are error messages passed to *marshmallow.exceptions.ValidationError*.

deserialize (*value*, *attr=None*, *data=None*, ***kwargs*)
Deserialize *value*.

Parameters

- **value** – The value to be deserialized.
- **attr** (*str*) – The attribute/key in *data* to be deserialized.
- **data** (*dict*) – The raw input data passed to the *Schema.load*.
- **kwargs** (*dict*) – Field-specific keyword arguments.

Raises *ValidationError* – If an invalid value is passed or if a required value is missing.

fail (*key*, ***kwargs*)
A helper method that simply raises a *ValidationError*.

get_value (*obj*, *attr*, *accessor=None*, *default=<marshmallow.missing>*)
Return the value for a given key from an object.

Parameters

- **obj** (*object*) – The object to get the value from
- **attr** (*str*) – The attribute/key in *obj* to get the value from.
- **accessor** (*callable*) – A callable used to retrieve the value of *attr* from the object *obj*. Defaults to *marshmallow.utils.get_value*.

root

Reference to the *Schema* that this field belongs to even if it is buried in a *List*. Return *None* for unbound fields.

serialize (*attr*, *obj*, *accessor=None*, ***kwargs*)
Pulls the value for the given key from the object, applies the field's formatting and returns the result.

Parameters

- **attr** (*str*) – The attribute or key to get from the object.
- **obj** (*str*) – The object to pull the key from.

- **accessor** (*callable*) – Function used to pull values from `obj`.
- **kwargs** (*dict*) – Field-specific keyword arguments.

Raises `ValidationError` – In case of formatting problem

```
class argschema.fields.Raw(default=<marshmallow.missing>, attribute=None, data_key=None,
                           error=None, validate=None, required=False, al-
                           low_none=None, load_only=False, dump_only=False, miss-
                           ing=<marshmallow.missing>, error_messages=None, **metadata)
```

Bases: `marshmallow.fields.Field`

Field that applies no formatting or validation.

```
class argschema.fields.Nested(nested, default=<marshmallow.missing>, exclude=(),
                              only=None, **kwargs)
```

Bases: `marshmallow.fields.Field`

Allows you to nest a `Schema` inside a field.

Examples:

```
user = fields.Nested(UserSchema)
user2 = fields.Nested('UserSchema') # Equivalent to above
collaborators = fields.Nested(UserSchema, many=True, only=('id',))
parent = fields.Nested('self')
```

When passing a *Schema* `<marshmallow.Schema>` instance as the first argument, the instance's `exclude`, `only`, and `many` attributes will be respected.

Therefore, when passing the `exclude`, `only`, or `many` arguments to `fields.Nested`, you should pass a *Schema* `<marshmallow.Schema>` class (not an instance) as the first argument.

```
# Yes
author = fields.Nested(UserSchema, only=('id', 'name'))

# No
author = fields.Nested(UserSchema(), only=('id', 'name'))
```

Parameters

- **nested** (*Schema*) – The `Schema` class or class name (string) to nest, or `"self"` to nest the `Schema` within itself.
- **exclude** (*tuple*) – A list or tuple of fields to exclude.
- **only** – A tuple or string of the field(s) to marshal. If `None`, all fields will be marshalled. If a field name (string) is given, only a single value will be returned as output instead of a dictionary. This parameter takes precedence over `exclude`.
- **many** (*bool*) – Whether the field is a collection of objects.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use `EXCLUDE`, `INCLUDE` or `RAISE`.
- **kwargs** – The same keyword arguments that `Field` receives.

```
default_error_messages = {u'type': u'Invalid type.'}
```

schema

The nested `Schema` object.

Changed in version 1.0.0: Renamed from *serializer* to *schema*

```
class argschema.fields.Mapping (keys=None, values=None, **kwargs)  
    Bases: marshmallow.fields.Field
```

An abstract class for objects with key-value pairs.

Parameters

- **keys** (`Field`) – A field class or instance for dict keys.
- **values** (`Field`) – A field class or instance for dict values.
- **kwargs** – The same keyword arguments that `Field` receives.

Note: When the structure of nested data is not known, you may omit the *keys* and *values* arguments to prevent content validation.

New in version 3.0.0rc4.

```
default_error_messages = {'invalid': 'Not a valid mapping type.'}
```

mapping_type
alias of `__builtin__.dict`

```
class argschema.fields.Dict (keys=None, values=None, **kwargs)  
    Bases: marshmallow.fields.Mapping
```

A dict field. Supports dicts and dict-like objects. Extends Mapping with dict as the mapping_type.

Example:

```
numbers = fields.Dict(keys=fields.Str(), values=fields.Float())
```

Parameters **kwargs** – The same keyword arguments that `Mapping` receives.

New in version 2.1.0.

mapping_type
alias of `__builtin__.dict`

```
class argschema.fields.List (cls_or_instance, **kwargs)  
    Bases: marshmallow.fields.Field
```

A list field, composed with another `Field` class or instance.

Example:

```
numbers = fields.List(fields.Float())
```

Parameters

- **cls_or_instance** (`Field`) – A field class or instance.
- **default** (`bool`) – Default value for serialization.
- **kwargs** – The same keyword arguments that `Field` receives.

Changed in version 2.0.0: The `allow_none` parameter now applies to deserialization and has the same semantics as the other fields.

```
default_error_messages = {'invalid': 'Not a valid list.'}
```

get_value (*obj*, *attr*, *accessor=None*)
 Return the value for a given key from an object.

class argschema.fields.**Tuple** (*tuple_fields*, **args*, ***kwargs*)
 Bases: `marshmallow.fields.Field`

A tuple field, composed of a fixed number of other *Field* classes or instances

Example:

```
row = Tuple((fields.String(), fields.Integer(), fields.Float()))
```

Note: Because of the structured nature of *collections.namedtuple* and *typing.NamedTuple*, using a Schema within a Nested field for them is more appropriate than using a *Tuple* field.

Parameters

- **tuple_fields** (*Iterable[Field]*) – An iterable of field classes or instances.
- **kwargs** – The same keyword arguments that *Field* receives.

New in version 3.0.0rc4.

default_error_messages = {*u'invalid'*: *u'Not a valid tuple.'*}

get_value (*obj*, *attr*, *accessor=None*)
 Return the value for a given key from an object.

class argschema.fields.**String** (*default=<marshmallow.missing>*, *attribute=None*,
data_key=None, *error=None*, *validate=None*, *required=False*,
allow_none=None, *load_only=False*, *dump_only=False*, *missing=<marshmallow.missing>*, *error_messages=None*, ***meta-*
data)
 Bases: `marshmallow.fields.Field`

A string field.

Parameters **kwargs** – The same keyword arguments that *Field* receives.

default_error_messages = {*u'invalid'*: *u'Not a valid string.'*, *u'invalid_utf8'*: *u'Not*

class argschema.fields.**UUID** (*default=<marshmallow.missing>*, *attribute=None*, *data_key=None*,
error=None, *validate=None*, *required=False*, *allow_none=None*, *load_only=False*, *dump_only=False*, *missing=<marshmallow.missing>*, *error_messages=None*, ***metadata*)
 Bases: `marshmallow.fields.String`

A UUID field.

default_error_messages = {*u'invalid_uuid'*: *u'Not a valid UUID.'*}

class argschema.fields.**Number** (*as_string=False*, ***kwargs*)
 Bases: `marshmallow.fields.Field`

Base class for number fields.

Parameters

- **as_string** (*bool*) – If True, format the serialized value as a string.
- **kwargs** – The same keyword arguments that *Field* receives.

default_error_messages = {*u'invalid'*: *u'Not a valid number.'*}

num_type
alias of `__builtin__.float`

class `argschema.fields.Integer` (*strict=False*, ***kwargs*)

Bases: `marshmallow.fields.Number`

An integer field.

Parameters *kwargs* – The same keyword arguments that *Number* receives.

default_error_messages = `{u'invalid': u'Not a valid integer.'}`

num_type
alias of `__builtin__.int`

class `argschema.fields.Decimal` (*places=None*, *rounding=None*, *allow_nan=False*,
as_string=False, ***kwargs*)

Bases: `marshmallow.fields.Number`

A field that (de)serializes to the Python `decimal.Decimal` type. It's safe to use when dealing with money values, percentages, ratios or other numbers where precision is critical.

Warning: This field serializes to a *decimal.Decimal* object by default. If you need to render your data as JSON, keep in mind that the *json* module from the standard library does not encode *decimal.Decimal*. Therefore, you must use a JSON library that can handle decimals, such as *simplejson*, or serialize to a string by passing *as_string=True*.

Warning: If a JSON *float* value is passed to this field for deserialization it will first be cast to its corresponding *string* value before being deserialized to a *decimal.Decimal* object. The default `__str__` implementation of the built-in Python *float* type may apply a destructive transformation upon its input data and therefore cannot be relied upon to preserve precision. To avoid this, you can instead pass a JSON *string* to be deserialized directly.

Parameters

- **places** (*int*) – How many decimal places to quantize the value. If *None*, does not quantize the value.
- **rounding** – How to round the value during quantize, for example *decimal.ROUND_UP*. If *None*, uses the rounding value from the current thread's context.
- **allow_nan** (*bool*) – If *True*, *NaN*, *Infinity* and *-Infinity* are allowed, even though they are illegal according to the JSON specification.
- **as_string** (*bool*) – If *True*, serialize to a string instead of a Python *decimal.Decimal* type.
- **kwargs** – The same keyword arguments that *Number* receives.

New in version 1.2.0.

default_error_messages = `{u'special': u'Special numeric values (nan or infinity) are not allowed'}`

num_type
alias of `decimal.Decimal`

class `argschema.fields.Boolean` (*truthy=None*, *falsy=None*, ***kwargs*)

Bases: `marshmallow.fields.Field`

A boolean field.

Parameters

- **truthy** (*set*) – Values that will (de)serialize to *True*. If an empty set, any non-falsy value will deserialize to *True*. If *None*, *marshmallow.fields.Boolean.truthy* will be used.
- **falsy** (*set*) – Values that will (de)serialize to *False*. If *None*, *marshmallow.fields.Boolean.falsy* will be used.
- **kwargs** – The same keyword arguments that *Field* receives.

```
default_error_messages = {u'invalid': u'Not a valid boolean.'}
```

```
falsy = set([u'OFF', 0, u'False', u'0', u'F', u'f', u'off', u'FALSE', u'n', u'No', u'N'])
Default falsy values.
```

```
truthy = set([u'on', 1, u'On', u'true', u'YES', u'1', u'Y', u't', u'ON', u'y', u'yes', u'Yes'])
Default truthy values.
```

```
class argschema.fields.FormattedString(src_str, *args, **kwargs)
```

Bases: *marshmallow.fields.Field*

Interpolate other values from the object into this field. The syntax for the source string is the same as the string *str.format* method from the python stdlib.

```
class UserSchema(Schema):
    name = fields.String()
    greeting = fields.FormattedString('Hello {name}')

ser = UserSchema()
res = ser.dump(user)
res.data # => {'name': 'Monty', 'greeting': 'Hello Monty'}
```

```
default_error_messages = {u'format': u'Cannot format string with given data.'}
```

```
class argschema.fields.Float(allow_nan=False, as_string=False, **kwargs)
```

Bases: *marshmallow.fields.Number*

A double as IEEE-754 double precision string.

Parameters

- **allow_nan** (*bool*) – If *True*, *NaN*, *Infinity* and *-Infinity* are allowed, even though they are illegal according to the JSON specification.
- **as_string** (*bool*) – If *True*, format the value as a string.
- **kwargs** – The same keyword arguments that *Number* receives.

```
default_error_messages = {u'special': u'Special numeric values (nan or infinity) are not allowed'}
```

num_type

alias of *__builtin__.float*

```
class argschema.fields.DateTime(format=None, **kwargs)
```

Bases: *marshmallow.fields.Field*

A formatted datetime string in UTC.

Example: '2014-12-22T03:12:58.019077+00:00'

Timezone-naive *datetime* objects are converted to UTC (+00:00) by *Schema.dump*. *Schema.load* returns *datetime* objects that are timezone-aware.

Parameters

- **format** (*str*) – Either "rfc" (for RFC822), "iso" (for ISO8601), or a date format string. If *None*, defaults to "iso".
- **kwargs** – The same keyword arguments that *Field* receives.

```
DEFAULT_FORMAT = u'iso'
```

```
DESERIALIZATION_FUNCS = {u'iso': <function from_iso_datetime at 0x7fe843a21050>, u'iso8601': <
```

```
OBJ_TYPE = u'datetime'
```

```
SCHEMA_OPTS_VAR_NAME = u'datetimeformat'
```

```
SERIALIZATION_FUNCS = {u'iso': <function isoformat at 0x7fe843a1aed8>, u'iso8601': <
```

```
default_error_messages = {u'format': u'"{input}" cannot be formatted as a {obj_type}.'
```

```
localtime = False
```

```
class argschema.fields.LocalDateTime (format=None, **kwargs)
```

```
Bases: marshmallow.fields.DateTime
```

A formatted datetime string in localized time, relative to UTC.

ex. "Sun, 10 Nov 2013 08:23:45 -0600"

Takes the same arguments as *DateTime*.

```
localtime = True
```

```
class argschema.fields.Time (default=<marshmallow.missing>, attribute=None, data_key=None,
                             error=None, validate=None, required=False, allow_none=None,
                             load_only=False, dump_only=False, missing=<marshmallow.missing>,
                             error_messages=None, **metadata)
```

```
Bases: marshmallow.fields.Field
```

ISO8601-formatted time string.

Parameters **kwargs** – The same keyword arguments that *Field* receives.

```
default_error_messages = {u'format': u'"{input}" cannot be formatted as a time.', u'iso8601':
```

```
class argschema.fields.Date (format=None, **kwargs)
```

```
Bases: marshmallow.fields.DateTime
```

ISO8601-formatted date string.

Parameters

- **format** – Either "iso" (for ISO8601) or a date format string. If *None*, defaults to "iso".
- **kwargs** – The same keyword arguments that *Field* receives.

```
DEFAULT_FORMAT = u'iso'
```

```
DESERIALIZATION_FUNCS = {u'iso': <function from_iso_date at 0x7fe843a21140>, u'iso8601': <
```

```
OBJ_TYPE = u'date'
```

```
SCHEMA_OPTS_VAR_NAME = u'dateformat'
```

```
SERIALIZATION_FUNCS = {u'iso': <function to_iso_date at 0x7fe843a211b8>, u'iso8601': <
```

```
default_error_messages = {u'format': u'"{input}" cannot be formatted as a date.', u'iso8601':
```

```
class argschema.fields.TimeDelta (precision=u'seconds', error=None, **kwargs)
```

```
Bases: marshmallow.fields.Field
```

A field that (de)serializes a `datetime.timedelta` object to an integer and vice versa. The integer can represent the number of days, seconds or microseconds.

Parameters

- **precision** (*str*) – Influences how the integer is interpreted during (de)serialization. Must be ‘days’, ‘seconds’, ‘microseconds’, ‘milliseconds’, ‘minutes’, ‘hours’ or ‘weeks’.
- **error** (*str*) – Error message stored upon validation failure.
- **kwargs** – The same keyword arguments that *Field* receives.

Changed in version 2.0.0: Always serializes to an integer value to avoid rounding errors. Add *precision* parameter.

DAYS = u'days'

HOURS = u'hours'

MICROSECONDS = u'microseconds'

MILLISECONDS = u'milliseconds'

MINUTES = u'minutes'

SECONDS = u'seconds'

WEEKS = u'weeks'

default_error_messages = {'format': u'{input!r} cannot be formatted as a timedelta.'}

```
class argschema.fields.Url (relative=False, schemes=None, require_tld=True, **kwargs)
    Bases: marshmallow.fields.String
```

A validated URL field. Validation occurs during both serialization and deserialization.

Parameters

- **default** – Default value for the field if the attribute is not set.
- **attribute** (*str*) – The name of the attribute to get the value from. If *None*, assumes the attribute has the same name as the field.
- **relative** (*bool*) – Allow relative URLs.
- **kwargs** – The same keyword arguments that *String* receives.

default_error_messages = {'invalid': u'Not a valid URL.'}

```
argschema.fields.URL
    alias of marshmallow.fields.Url
```

```
class argschema.fields.Email (*args, **kwargs)
    Bases: marshmallow.fields.String
```

A validated email field. Validation occurs during both serialization and deserialization.

Parameters

- **args** – The same positional arguments that *String* receives.
- **kwargs** – The same keyword arguments that *String* receives.

default_error_messages = {'invalid': u'Not a valid email address.'}

```
class argschema.fields.Method (serialize=None, deserialize=None, **kwargs)
    Bases: marshmallow.fields.Field
```

A field that takes the value returned by a *Schema* method.

Parameters

- **serialize** (*str*) – The name of the Schema method from which to retrieve the value. The method must take an argument *obj* (in addition to *self*) that is the object to be serialized.
- **deserialize** (*str*) – Optional name of the Schema method for deserializing a value. The method must take a single argument *value*, which is the value to deserialize.

Changed in version 2.0.0: Removed optional *context* parameter on methods. Use *self.context* instead.

Changed in version 2.3.0: Deprecated *method_name* parameter in favor of *serialize* and allow *serialize* to not be passed at all.

Changed in version 3.0.0: Removed *method_name* parameter.

class argschema.fields.**Function** (*serialize=None, deserialize=None, func=None, **kwargs*)

Bases: `marshmallow.fields.Field`

A field that takes the value returned by a function.

Parameters

- **serialize** (*callable*) – A callable from which to retrieve the value. The function must take a single argument *obj* which is the object to be serialized. It can also optionally take a *context* argument, which is a dictionary of context variables passed to the serializer. If no callable is provided then the ``load_only`` flag will be set to `True`.
- **deserialize** (*callable*) – A callable from which to retrieve the value. The function must take a single argument *value* which is the value to be deserialized. It can also optionally take a *context* argument, which is a dictionary of context variables passed to the deserializer. If no callable is provided then ``value`` will be passed through unchanged.

Changed in version 2.3.0: Deprecated *func* parameter in favor of *serialize*.

Changed in version 3.0.0a1: Removed *func* parameter.

argschema.fields.**Str**

alias of `marshmallow.fields.String`

argschema.fields.**Bool**

alias of `marshmallow.fields.Boolean`

argschema.fields.**Int**

alias of `marshmallow.fields.Integer`

class argschema.fields.**Constant** (*constant, **kwargs*)

Bases: `marshmallow.fields.Field`

A field that (de)serializes to a preset constant. If you only want the constant added for serialization or deserialization, you should use `dump_only=True` or `load_only=True` respectively.

Parameters **constant** – The constant to return for the field attribute.

New in version 2.0.0.

class argschema.fields.**OutputFile** (*default=<marshmallow.missing>, attribute=None, data_key=None, error=None, validate=None, required=False, allow_none=None, load_only=False, dump_only=False, missing=<marshmallow.missing>, error_messages=None, **metadata*)

Bases: `marshmallow.fields.String`

OutputFile `marshmallow.fields.Str` subclass which is a path to a file location that can be written to by the current user (presently tested by opening a temporary file to that location)

```
class argschema.fields.InputDir (default=<marshmallow.missing>,          attribute=None,
                                data_key=None, error=None, validate=None, required=False,
                                allow_none=None, load_only=False, dump_only=False,
                                missing=<marshmallow.missing>, error_messages=None,
                                **metadata)
```

Bases: `marshmallow.fields.String`

InputDir is `marshmallow.fields.Str` subclass which is a path to a directory that exists and that the user can access (presently checked with `os.access`)

```
class argschema.fields.InputFile (default=<marshmallow.missing>,          attribute=None,
                                  data_key=None, error=None, validate=None, re-
                                  quired=False, allow_none=None, load_only=False,
                                  dump_only=False, missing=<marshmallow.missing>,
                                  error_messages=None, **metadata)
```

Bases: `marshmallow.fields.String`

InputFile is a `marshmallow.fields.Str` subclass which is a path to a file location which can be read by the user (presently passes `os.path.isfile` and `os.access = R_OK`)

```
class argschema.fields.OutputDir (mode=None, *args, **kwargs)
```

Bases: `marshmallow.fields.String`

OutputDir is a `marshmallow.fields.Str` subclass which is a path to a location where this module will write files. Validation will check that the directory exists and create the directory if it is not present, and will fail validation if the directory cannot be created or cannot be written to.

Parameters

- **mode** (*str*) – mode to create directory
- ***args** – same as passed to `marshmallow.fields.Str`
- ****kwargs** – same as passed to `marshmallow.fields.Str`

```
class argschema.fields.NumpyArray (dtype=None, *args, **kwargs)
```

Bases: `marshmallow.fields.List`

NumpyArray is a `marshmallow.fields.List` subclass which will convert any numpy compatible set of lists into a numpy array after deserialization and convert it back to a list when serializing,

Parameters **dtype** (*numpy.Dtype*) – dtype specifying the desired data type. if dtype is given the array will be converted to the type, otherwise numpy will decide what type it should be. (Default=None)

```
class argschema.fields.LogLevel (**kwargs)
```

Bases: `marshmallow.fields.String`

LogLevel is a field type that provides a setting for the loglevel of `python.logging`. This class will both validate the input and also *set* the input globally. In simple scenarios, a module will not have to do any manipulation of loglevel.

options = ['FATAL', 'CRITICAL', 'ERROR', 'WARN', 'WARNING', 'INFO', 'DEBUG']

```
class argschema.fields.Slice (**kwargs)
```

Bases: `marshmallow.fields.String`

Slice is a `marshmallow.fields.Str` field that supports a range or slice argument for selecting some subset of a larger dataset. The syntax is identical to numpy slicing. Examples: “10:20”, “40”, “:30”, “10:2:40”

Parameters **kwargs** – the same as any `Str` receive

2.1.2 Submodules

2.1.3 argschema.argschema_parser module

Module that contains the base class ArgSchemaParser which should be subclassed when using this library

```
class argschema.argschema_parser.ArgSchemaParser (input_data=None,
                                                    schema_type=None,          out-
                                                    put_schema_type=None,
                                                    args=None,      input_source=None,
                                                    output_sink=None,          log-
                                                    ger_name='argschema.argschema_parser')
```

Bases: object

The main class you should sub-class to write your own argschema module. Takes input_data, reference to a input_json and the command line inputs and parses out the parameters and validates them against the schema_type specified.

To subclass this and make a new schema be default, simply override the default_schema and default_output_schema attributes of this class.

Parameters

- **input_data** (*dict or None*) – dictionary parameters to fall back on if not source is given or configured via command line
- **schema_type** (*schemas.ArgSchema*) – the schema to use to validate the parameters
- **output_schema_type** (*marshmallow.Schema*) – the schema to use to validate the output, used by self.output
- **input_source** (*argschema.sources.source.Source*) – a generic source of a dictionary
- **output_sink** (*argschema.sources.source.Source*) – a generic sink to write output dictionary to
- **args** (*list or None*) – command line arguments passed to the module, if None use argparse to parse the command line, set to [] if you want to bypass command line parsing
- **logger_name** (*str*) – name of logger from the logging module you want to instantiate default ('argschema')

Raises **marshmallow.ValidationError** – If the combination of input_json, input_data and command line arguments do not pass the validation of the schema

Note: This class takes a ArgSchema as an input to parse inputs , with a default schema of type *ArgSchema*

```
default_configurable_sinks = [<class 'argschema.sources.json_source.JsonSink'>]
default_configurable_sources = [<class 'argschema.sources.json_source.JsonSource'>]
default_output_schema = None
default_schema
    alias of argschema.schemas.ArgSchema
get_output_json (d)
    method for getting the output_json pushed through validation if validation exists :param d: output dictio-
    nary to output :type d: dict
```

Returns validated and serialized version of the dictionary

Return type dict

Raises `marshmallow.ValidationError` – If any of the output dictionary doesn't meet the output schema

static initialize_logger (*name*, *log_level*)

initializes the logger to a level with a name logger = initialize_logger(name, log_level)

Parameters

- **name** (*str*) – name of the logger
- **log_level** –

Returns a logger set with the name and level specified

Return type logging.Logger

load_schema_with_defaults (*schema*, *args*)

method for deserializing the arguments dictionary (args) given the schema (schema) making sure that the default values have been filled in.

Parameters

- **args** (*dict*) – a dictionary of input arguments
- **schema** –

Returns a deserialized dictionary of the parameters converted through marshmallow

Return type dict

Raises `marshmallow.ValidationError` – If this schema contains nested schemas that don't subclass `argschema.DefaultSchema` because these won't work with loading defaults.

output (*d*, *sink=None*)

method for outputting dictionary to the output_json file path after validating it through the output_schema_type

Parameters

- **d** (*dict*) – output dictionary to output
- **sink** (*argschema.sources.source.ArgSink*) – output_sink to output to (optional default to self.output_source)

Raises `marshmallow.ValidationError` – If any of the output dictionary doesn't meet the output schema

```
class argschema.argschema_parser.ArgSchemaYamlParser (input_data=None,
                                                         schema_type=None,      out-
                                                         put_schema_type=None,
                                                         args=None,                  in-
                                                         put_source=None,           out-
                                                         put_sink=None,             log-
                                                         ger_name='argschema.argschema_parser')
```

Bases: `argschema.argschema_parser.ArgSchemaParser`

Note: This class takes a `ArgSchema` as an input to parse inputs , with a default schema of type `ArgSchema`

```
default_configurable_sinks = [<class 'argschema.sources.yaml_source.YamlSink'>]
```

```
default_configurable_sources = [<class 'argschema.sources.yaml_source.YamlSource'>]
```

2.1.4 argschema.schemas module

```
class argschema.schemas.ArgSchema (only=None, exclude=(), many=False, context=None,
                                     load_only=(), dump_only=(), partial=False, un-
                                     known=None)
```

Bases: `argschema.schemas.DefaultSchema`

The base marshmallow schema used by ArgSchemaParser to identify input_json and output_json files and the log_level

This schema is designed to be a schema_type for an ArgSchemaParser object

Table 1: ArgSchema

key	description	default	field_type	json_type
log_level	set the logging level of the module	ERROR	<code>LogLevel</code>	str

```
opts = <marshmallow.schema.SchemaOpts object>
```

```
class argschema.schemas.DefaultSchema (only=None, exclude=(), many=False, context=None,
                                         load_only=(), dump_only=(), partial=False, un-
                                         known=None)
```

Bases: `marshmallow.schema.Schema`

mm.Schema class with support for making fields default to values defined by that field's arguments.

```
make_object (in_data)
```

marshmallow.pre_load decorated function for applying defaults on deserialiation

Parameters `in_data` –

Returns a dictionary with default values applied

Return type dict

```
opts = <marshmallow.schema.SchemaOpts object>
```

2.1.5 argschema.utils module

module that contains argschema functions for converting marshmallow schemas to argparse and merging dictionaries from both systems

```
argschema.utils.args_to_dict (argsobj, schemas=None)
```

function to convert namespace returned by argparse into a nested dictionary

Parameters

- **argsobj** (`argparse.Namespace`) – Namespace object returned by standard argparse.parse function
- **schemas** (`list[marshmallow.Schema]`) – Optional list of schemas which will be used to cast fields via `FIELD_TYPE_MAP`

Returns dictionary of namespace values where nesting elements uses ‘.’ to denote nesting of keys

Return type dict

`argschema.utils.build_schema_arguments` (*schema*, *arguments=None*, *path=None*, *description=None*)

given a jsonschema, create a dictionary of argparse arguments, by navigating down the Nested schema tree. (recursive function)

Parameters

- **schema** (*marshmallow.Schema*) – schema with field.description filled in with help values
- **arguments** (*list or None*) – list of argument group dictionaries to add to (see Returns) (Default value = None)
- **path** (*list or None*) – list of strings denoted where you are in the tree (Default value = None)
- **description** (*str or None*) – description for the argument group at this level of the tree

Returns List of argument group dictionaries, with keys ['title','description','args'] which contain the arguments for argparse. 'args' is an OrderedDict of dictionaries with keys of the argument names with kwargs to build an argparse argument

Return type list

`argschema.utils.cli_error_dict` (*arg_path*, *field_type*, *index=0*)

Construct a nested dictionary containing a casting error message

Matches the format of errors generated by schema.dump.

Parameters

- **arg_path** (*string*) – List of nested keys
- **field_type** (*string*) – Name of the marshmallow.Field type
- **index** (*int*) – Index into arg_path for recursion

Returns Dictionary representing argument path, containing error.

Return type dict

`argschema.utils.dump` (*schema*, *d*)

function to wrap marshmallow dump to smooth differences from marshmallow 2 to 3

Parameters

- **schema** (*marshmallow.Schema*) – schema that you want to use to validate and dump
- **d** (*dict*) – dictionary to validate and dump

Returns serialized and validated dictionary

Return type dict

`argschema.utils.get_description_from_field` (*field*)

get the description for this marshmallow field

Parameters **field** (*marshmallow.fields.field*) – field to get description

Returns description string (or None)

Return type str

`argschema.utils.get_field_def_from_schema` (*parts*, *schema*)

function to get a field_definition from a particular key, specified by it's parts list

Parameters

- **parts** (*list[str]*) – the list of keys to get this schema
- **schema** (*marshmallow.Schema*) – the marshmallow schema to look up this key

Returns returns the field in the schema if it exists, otherwise returns None

Return type marshmallow.Field or None

`argschema.utils.get_type_from_field(field)`

Get type casting for command line argument from marshmallow.Field

Parameters **field** (*marshmallow.Field*) – Field class from input schema

Returns Function to call to cast argument to

Return type callable

`argschema.utils.load(schema, d)`

function to wrap marshmallow load to smooth differences from marshmallow 2 to 3

Parameters

- **schema** (*marshmallow.Schema*) – schema that you want to use to validate
- **d** (*dict*) – dictionary to validate and load

Returns deserialized and validated dictionary

Return type dict

`argschema.utils.merge_value(a, b, key, func=<built-in function add>)`

attempt to merge these dictionaries using function defined by func (default to add) raise an exception if this fails

Parameters

- **a** (*dict*) – one dictionary
- **b** (*dict*) – second dictionary
- **key** (*key*) – key to merge dictionary values on
- **func** (*a[key]*) – function that merges two values of this key Returns (Default value = add)
- **func** – merged version of values (Default value = add)

Returns

Return type value

`argschema.utils.prune_dict_with_none(d)`

function to remove all dictionaries from a nested dictionary when all the values of a particular dictionary are None

Parameters **d** (*dictionary to prune*) –

Returns pruned dictionary

Return type dict

`argschema.utils.schema_argparser(schema, additional_schemas=None)`

given a jsonschema, build an argparse.ArgumentParser

Parameters

- **schema** (`argschema.schemas.ArgSchema`) – schema to build an argparser from
- **additional_schemas** (`list[marshmallow.schema]`) – list of additional schemas to add to the command line arguments

Returns that represents the schemas

Return type `argparse.ArgumentParser`

`argschema.utils.smart_merge(a, b, path=None, merge_keys=None, overwrite_with_none=False)`
updates dictionary `a` with values in dictionary `b` being careful not to write things with `None`, and performing a merge on `merge_keys`

Parameters

- **a** (`dict`) – dictionary to perform update on
- **b** (`dict`) – dictionary to perform update with
- **path** (`list`) – list of nested keys traversed so far (used for recursion) (Default value = `None`)
- **merge_keys** (`list`) – list of keys to do merging on (default `None`)
- **overwrite_with_none** – (Default value = `False`)

Returns a dictionary that is a updated with `b`'s values

Return type `dict`

2.1.6 argschema.validate module

module for custom marshmallow validators

class `argschema.validate.Shape` (`shape=None`)

Bases: `marshmallow.validate.Validator`

Validator which succeeds if `value.shape` matches `shape`

Parameters **shape** (`tuple`) – Tuple specifying the required shape. If a value in the tuple is `None`, any length in that dimension is valid.

Raises

- **ValueError** – If the provided shape is not a valid tuple of integers and/or `None` types
- **marshmallow.ValidationError** – If the value being validated does not have a `shape` attribute

2.1.7 argschema.autodoc module

`argschema.autodoc.process_schemas` (`app, what, name, obj, options, lines`)

function designed to process a `sphinx.ext.autodoc` event as autodoc hook to alter docstring lines of argschema related classes, providing a table of parameters for schemas and links to the default schemas for `ArgSchemaParser` derived elements

use in sphinx `conf.py` as follows

```
from argschema.autodoc import process_schemas
def setup(app):
    app.connect('autodoc-process-docstring', process_schemas)
```

2.1.8 Module contents

argschema: flexible definition, validation and setting of parameters

`argschema.main()`

This contains the tests

3.1 test

3.1.1 test_first_test module

```
class test_first_test.BadExampleRecursiveSchema (extra=None, only=None, exclude=(),
                                                prefix="u", strict=None, many=False,
                                                context=None, load_only=(),
                                                dump_only=(), partial=False)
```

Bases: `argschema.schemas.ArgSchema`

This schema is designed to be a schema_type for an ArgSchemaParser object

Table 1: BadExampleRecursiveSchema

key	description	default	field_type	json_type
log_level	set the logging level of the module	ERROR	<code>LogLevel</code>	str
tree	no description	(REQUIRED)	<code>BadRecursiveSchema</code>	dict

```
opts = <marshmallow.schema.SchemaOpts object>
```

```
class test_first_test.BadRecursiveSchema (extra=None, only=None, exclude=(), pre-
                                         fix="u", strict=None, many=False, context=None,
                                         load_only=(), dump_only=(), partial=False)
```

Bases: `marshmallow.schema.Schema`

Table 2: BadRecursiveSchema

key	description	default	field_type	json_type
children	children of this node	NA	<code>BadRecursiveSchema</code>	list
name	name of this node	anonymous	<code>String</code>	str

```
opts = <marshmallow.schema.SchemaOpts object>
```

```
class test_first_test.ExampleRecursiveSchema (extra=None, only=None, exclude=(), pre-
                                             fix=u", strict=None, many=False, con-
                                             text=None, load_only=(), dump_only=(),
                                             partial=False)
```

Bases: `argschema.schemas.ArgSchema`

This schema is designed to be a schema_type for an ArgSchemaParser object

Table 3: ExampleRecursiveSchema

key	description	default	field_type	json_type
log_level	set the logging level of the module	ERROR	<code>LogLevel</code>	str
tree	no description	(REQUIRED)	<code>RecursiveSchema</code>	dict

```
opts = <marshmallow.schema.SchemaOpts object>
```

```
class test_first_test.ModelFit (extra=None, only=None, exclude=(), prefix=u", strict=None,
                                many=False, context=None, load_only=(), dump_only=(), par-
                                tial=False)
```

Bases: `argschema.schemas.DefaultSchema`

Table 4: ModelFit

key	description	default	field_type	json_type
fit_type	no description	NA	<code>String</code>	str
hof	no description	NA	<code>InputFile</code>	str
hof_fit	no description	NA	<code>InputFile</code>	str

```
opts = <marshmallow.schema.SchemaOpts object>
```

```
class test_first_test.MyExtension (extra=None, only=None, exclude=(), prefix=u",
                                   strict=None, many=False, context=None, load_only=(),
                                   dump_only=(), partial=False)
```

Bases: `argschema.schemas.DefaultSchema`

Table 5: MyExtension

key	description	default	field_type	json_type
a	a string	(REQUIRED)	<code>String</code>	str
c	an integer	10	<code>Integer</code>	int
b	an integer	NA	<code>Integer</code>	int
d	a list of integers	NA	<code>List</code>	int

```
opts = <marshmallow.schema.SchemaOpts object>
```

```
class test_first_test.MyShorterExtension (extra=None, only=None, exclude=(), pre-
                                           fix=u", strict=None, many=False, context=None,
                                           load_only=(), dump_only=(), partial=False)
```

Bases: `argschema.schemas.ArgSchema`

This schema is designed to be a schema_type for an ArgSchemaParser object

Table 6: MyShorterExtension

key	description	default	field_type	json_type
a	a string	NA	<i>String</i>	str
c	an integer	10	<i>Integer</i>	int
b	an integer	NA	<i>Integer</i>	int
log_level	set the logging level of the module	ERROR	<i>LogLevel</i>	str
d	a list of integers	NA	<i>List</i>	int

opts = <marshmallow.schema.SchemaOpts object>

```
class test_first_test.PopulationSelectionParameters (extra=None, only=None,
                                                    exclude=(), prefix="u",
                                                    strict=None, many=False,
                                                    context=None, load_only=(),
                                                    dump_only=(), partial=False)
```

Bases: *argschema.schemas.ArgSchema*

This schema is designed to be a schema_type for an ArgSchemaParser object

Table 7: PopulationSelectionParameters

key	description	default	field_type	json_type
paths	no description	NA	<i>PopulationSelectionPaths</i>	dict
log_level	set the logging level of the module	ERROR	<i>LogLevel</i>	str

opts = <marshmallow.schema.SchemaOpts object>

```
class test_first_test.PopulationSelectionPaths (extra=None, only=None, exclude=(),
                                                prefix="u", strict=None, many=False,
                                                context=None, load_only=(),
                                                dump_only=(), partial=False)
```

Bases: *argschema.schemas.DefaultSchema*

Table 8: PopulationSelectionPaths

key	description	default	field_type	json_type
fits	no description	NA	<i>ModelFit</i>	list

opts = <marshmallow.schema.SchemaOpts object>

```
class test_first_test.RecursiveSchema (extra=None, only=None, exclude=(), pre-
                                       fix="u", strict=None, many=False, context=None,
                                       load_only=(), dump_only=(), partial=False)
```

Bases: *argschema.schemas.DefaultSchema*

Table 9: RecursiveSchema

key	description	default	field_type	json_type
children	children of this node	NA	<i>RecursiveSchema</i>	list
name	name of this node	anonymous	<i>String</i>	str

opts = <marshmallow.schema.SchemaOpts object>

```
class test_first_test.SimpleExtension (extra=None, only=None, exclude=(), pre-
                                       fix="u", strict=None, many=False, context=None,
                                       load_only=(), dump_only=(), partial=False)
```

Bases: *argschema.schemas.ArgSchema*

This schema is designed to be a schema_type for an ArgSchemaParser object

Table 10: SimpleExtension

key	description	default	field_type	json_type
test	no description	(REQUIRED)	<i>MyExtension</i>	dict
log_level	set the logging level of the module	ERROR	<i>LogLevel</i>	str

```
opts = <marshmallow.schema.SchemaOpts object>
test_first_test.bad_test_recursive_schema()
test_first_test.simple_extension_file(*args, **kwargs)
test_first_test.test_bad_input_json_argparse()
test_first_test.test_bad_path()
test_first_test.test_david_example(tmpdir_factory)
test_first_test.test_log_catch()
test_first_test.test_recursive_schema()
test_first_test.test_simple_description()
test_first_test.test_simple_example()
test_first_test.test_simple_extension_fail()
test_first_test.test_simple_extension_pass()
test_first_test.test_simple_extension_required()
test_first_test.test_simple_extension_write_debug_level(simple_extension_file)
test_first_test.test_simple_extension_write_overwrite(simple_extension_file)
test_first_test.test_simple_extension_write_overwrite_list(simple_extension_file)
test_first_test.test_simple_extension_write_pass(simple_extension_file)
```

3.1.2 test_output module

```
class test_output.MyOutputSchema (only=None, exclude=(), many=False, context=None,
                                   load_only=(), dump_only=(), partial=False, un-
                                   known=None)
Bases: argschema.schemas.DefaultSchema
```

Table 11: MyOutputSchema

key	description	default	field_type	json_type
a	a simple string	(REQUIRED)	<i>String</i>	str
b	a default integer	5	<i>Integer</i>	int
M	a numpy array of answers	(REQUIRED)	<i>NumpyArray</i>	?

```
opts = <marshmallow.schema.SchemaOpts object>
test_output.test_alt_output(tmpdir)
test_output.test_bad_output(tmpdir)
test_output.test_output(tmpdir)
```

```
test_output.test_output_unvalidated(tmpdir)
test_output.test_tmp_output_cleanput(tmpdir)
```

3.1.3 test_argschema_parser module

```
class test_argschema_parser.MyNestedSchema (extra=None, only=None, exclude=(), pre-
                                         fix=u", strict=None, many=False, con-
                                         text=None, load_only=(), dump_only=(),
                                         partial=False)
```

Bases: *argschema.schemas.DefaultSchema*

Table 12: MyNestedSchema

key	description	default	field_type	json_type
two	a nested boolean	(REQUIRED)	<i>Boolean</i>	bool
one	nested integer	(REQUIRED)	<i>Integer</i>	int

opts = <marshmallow.schema.SchemaOpts object>

```
class test_argschema_parser.MyNestedSchemaWithDefaults (extra=None,    only=None,
                                                         exclude=(),    pre-
                                                         fix=u",    strict=None,
                                                         many=False,    con-
                                                         text=None,    load_only=(),
                                                         dump_only=(),    par-
                                                         tial=False)
```

Bases: *argschema.schemas.DefaultSchema*

Table 13: MyNestedSchemaWithDefaults

key	description	default	field_type	json_type
two	a nested boolean	True	<i>Boolean</i>	bool
one	nested integer	1	<i>Integer</i>	int

opts = <marshmallow.schema.SchemaOpts object>

```
class test_argschema_parser.MyParser (input_data=None,    schema_type=None,    out-
                                         put_schema_type=None,    args=None,    in-
                                         put_source=None,    output_sink=None,    log-
                                         ger_name='argschema.argschema_parser')
```

Bases: *argschema.argschema_parser.ArgSchemaParser*

Note: This class takes a ArgSchema as an input to parse inputs , with a default schema of type *MySchema*

default_schema
alias of *MySchema*

```
class test_argschema_parser.MySchema (extra=None,    only=None,    exclude=(),    pre-
                                         fix=u",    strict=None,    many=False,    context=None,
                                         load_only=(),    dump_only=(),    partial=False)
```

Bases: *argschema.schemas.ArgSchema*

This schema is designed to be a schema_type for an ArgSchemaParser object

Table 14: MySchema

key	description	default	field_type	json_type
a	parameter a	(REQUIRED)	<code>Integer</code>	int
nest	a nested schema	NA	<code>MyNestedSchema</code>	dict
b	optional b string parameter	my value	<code>String</code>	str
log_level	set the logging level of the module	ERROR	<code>LogLevel</code>	str

opts = <marshmallow.schema.SchemaOpts object>

```
class test_argschema_parser.MySchema2 (extra=None, only=None, exclude=(), prefix=u", strict=None, many=False, context=None, load_only=(), dump_only=(), partial=False)
```

Bases: `argschema.schemas.ArgSchema`

This schema is designed to be a schema_type for an ArgSchemaParser object

Table 15: MySchema2

key	description	default	field_type	json_type
a	parameter a	(REQUIRED)	<code>Integer</code>	int
nest	a nested schema	NA	<code>MyNestedSchemaWithDefaults</code>	dict
b	optional b string parameter	my value	<code>String</code>	str
log_level	set the logging level of the module	ERROR	<code>LogLevel</code>	str

opts = <marshmallow.schema.SchemaOpts object>

```
test_argschema_parser.test_boolean_command_line (default, args, expected)
```

```
test_argschema_parser.test_my_default_nested_parser ()
```

```
test_argschema_parser.test_my_parser ()
```

```
test_argschema_parser.test_parser_output (tmpdir_factory)
```

3.1.4 test_utils module

```
class test_utils.BaseballSituation (extra=None, only=None, exclude=(), prefix=u", strict=None, many=False, context=None, load_only=(), dump_only=(), partial=False)
```

Bases: `argschema.schemas.ArgSchema`

A description of a baseball situation

This schema is designed to be a schema_type for an ArgSchemaParser object

Table 16: BaseballSituation

key	description	default	field_type	json_type
inning	inning (1-9)	(REQUIRED)	<i>Integer</i>	int
outs	number of outs (0-2)	(REQUIRED)	<i>Integer</i>	int
log_level	set the logging level of the module	ERROR	<i>LogLevel</i>	str
bottom	is it the bottom of the inning	(REQUIRED)	<i>Boolean</i>	bool
score_away	away team score (non-negative)	(REQUIRED)	<i>Integer</i>	int
pitcher	who is pitching	(REQUIRED)	<i>Player</i>	dict
strikes	how many strikes (0-2)	(REQUIRED)	<i>Integer</i>	int
balls	number of balls (0-4)	0	<i>Integer</i>	int
batter	who is batting	(REQUIRED)	<i>Player</i>	dict
bases_occupied	which bases are occupied	NA	<i>List</i>	int
score_home	home team score (non-negative)	(REQUIRED)	<i>Integer</i>	int

opts = <marshmallow.schema.SchemaOpts object>

class test_utils.Player (extra=None, only=None, exclude=(), prefix="u", strict=None, many=False, context=None, load_only=(), dump_only=(), partial=False)

Bases: *argschema.schemas.DefaultSchema*

player information

Table 17: Player

key	description	default	field_type	json_type
number	player's number (must be >0)	(REQUIRED)	<i>Integer</i>	int
name	players name	(REQUIRED)	<i>String</i>	str

opts = <marshmallow.schema.SchemaOpts object>

test_utils.test_merge_value_add()

test_utils.test_merge_value_fail()

test_utils.test_merge_value_subtract()

test_utils.test_schema_argparser_with_baseball()

test_utils.test_smart_merge()

test_utils.test_smart_merge_add()

test_utils.test_smart_merge_nested()

test_utils.test_smart_merge_none()

test_utils.test_smart_merge_not_none()

test_utils.test_smart_merge_same()

3.1.5 test_autodoc module

class test_autodoc.SchemaWithQuotedDescriptions (extra=None, only=None, exclude=(), prefix="u", strict=None, many=False, context=None, load_only=(), dump_only=(), partial=False)

Bases: *argschema.schemas.ArgSchema*

This schema is designed to be a schema_type for an ArgSchemaParser object

Table 18: SchemaWithQuotedDescriptions

key	description	default	field_type	json_type
a	something that is ‘quoted’ is problematic	(REQUIRED)	<i>Integer</i>	int
log_level	set the logging level of the module	ERROR	<i>LogLevel</i>	str

opts = <marshmallow.schema.SchemaOpts object>

`test_autodoc.test_autodoc()`

`test_autodoc.test_autodoc_argschemaparser()`

`test_autodoc.test_autodoc_list()`

`test_autodoc.test_autodoc_myparser()`

`test_autodoc.test_autodoc_nested()`

`test_autodoc.test_autodoc_quotes()`

`test_autodoc.test_autodoc_recursive_nested()`

`test_autodoc.test_autodoc_slice()`

`test_autodoc.validate_rst_lines(lines, level=2)`

validates a set of lines that would make up an rst file using rstcheck

Parameters

- **lines** (*list[str]*) – a list of lines that would compose some restructuredText
- **level** (*docutils.utils.Reporter.WARNING_LEVEL*) – the reporting level to hold this to

Returns

Return type None

Raises **AssertionError** – If the lines contain any errors above the level

3.2 fields package

3.2.1 Submodules

3.2.2 fields.test_files module

class `fields.test_files.BasicInputDir` (*extra=None, only=None, exclude=(), prefix=u”, strict=None, many=False, context=None, load_only=(), dump_only=(), partial=False*)

Bases: *argschema.schemas.ArgSchema*

This schema is designed to be a schema_type for an ArgSchemaParser object

Table 19: BasicInputDir

key	description	default	field_type	json_type
input_dir	a simple file	(REQUIRED)	<i>InputDir</i>	str
log_level	set the logging level of the module	ERROR	<i>LogLevel</i>	str

opts = <marshmallow.schema.SchemaOpts object>

```
class fields.test_files.BasicInputFile (extra=None, only=None, exclude=(), prefix=u", strict=None, many=False, context=None, load_only=(), dump_only=(), partial=False)
```

Bases: *argschema.schemas.ArgSchema*

This schema is designed to be a schema_type for an ArgSchemaParser object

Table 20: BasicInputFile

key	description	default	field_type	json_type
log_level	set the logging level of the module	ERROR	<i>LogLevel</i>	str
input_file	a simple file	(REQUIRED)	<i>InputFile</i>	str

```
opts = <marshmallow.schema.SchemaOpts object>
```

```
class fields.test_files.BasicOutputDir (extra=None, only=None, exclude=(), prefix=u", strict=None, many=False, context=None, load_only=(), dump_only=(), partial=False)
```

Bases: *argschema.schemas.ArgSchema*

This schema is designed to be a schema_type for an ArgSchemaParser object

Table 21: BasicOutputDir

key	description	default	field_type	json_type
log_level	set the logging level of the module	ERROR	<i>LogLevel</i>	str
output_dir	basic output dir	(REQUIRED)	<i>OutputDir</i>	str

```
opts = <marshmallow.schema.SchemaOpts object>
```

```
class fields.test_files.BasicOutputFile (extra=None, only=None, exclude=(), prefix=u", strict=None, many=False, context=None, load_only=(), dump_only=(), partial=False)
```

Bases: *argschema.schemas.ArgSchema*

This schema is designed to be a schema_type for an ArgSchemaParser object

Table 22: BasicOutputFile

key	description	default	field_type	json_type
output_file	a simple output file	(REQUIRED)	<i>OutputFile</i>	str
log_level	set the logging level of the module	ERROR	<i>LogLevel</i>	str

```
opts = <marshmallow.schema.SchemaOpts object>
```

```
class fields.test_files.ModeOutputDirSchema (extra=None, only=None, exclude=(), prefix=u", strict=None, many=False, context=None, load_only=(), dump_only=(), partial=False)
```

Bases: *argschema.schemas.ArgSchema*

This schema is designed to be a schema_type for an ArgSchemaParser object

Table 23: ModeOutputDirSchema

key	description	default	field_type	json_type
log_level	set the logging level of the module	ERROR	<i>LogLevel</i>	str
output_dir	775 output directory	(REQUIRED)	<i>OutputDir</i>	str

```
opts = <marshmallow.schema.SchemaOpts object>
fields.test_files.test_access_inputfile_failed()
fields.test_files.test_bad_inputdir()
fields.test_files.test_basic_inputdir(tmpdir)
fields.test_files.test_enoent_outputfile_failed()
fields.test_files.test_failed_mode(tmpdir)
fields.test_files.test_inputdir_no_access(tmpdir)
fields.test_files.test_mode_output_osdir(tmpdir)
fields.test_files.test_output_dir_bad_location()
fields.test_files.test_output_dir_bad_permission(tmpdir)
fields.test_files.test_output_dir_basic(tmpdir)
fields.test_files.test_output_file_relative()
fields.test_files.test_output_path(tmpdir)
fields.test_files.test_output_path_cannot_write()
fields.test_files.test_output_path_noopath()
fields.test_files.test_outputfile_no_write(tmpdir)
fields.test_files.test_outputfile_not_a_path()
fields.test_files.test_relative_file_input()
fields.test_files.test_relative_file_input_failed()
```

3.2.3 fields.test_loglevel module

```
fields.test_loglevel.test_bad_option()
fields.test_loglevel.test_option_list()
```

3.2.4 fields.test_numpyarray module

```
class fields.test_numpyarray.NumpyFileuint16 (extra=None, only=None, exclude=(), pre-
                                             fix="u", strict=None, many=False, con-
                                             text=None, load_only=(), dump_only=(),
                                             partial=False)
```

Bases: [argschema.schemas.ArgSchema](#)

This schema is designed to be a schema_type for an ArgSchemaParser object

Table 24: NumpyFileuint16

key	description	default	field_type	json_type
a	list of lists representing a uint16 numpy array	(REQUIRED)	NumpyArray	?
log_level	set the logging level of the module	ERROR	LogLevel	str

```
opts = <marshmallow.schema.SchemaOpts object>
```

```
fields.test_numpyarray.test_bad_data()
fields.test_numpyarray.test_bad_shape()
fields.test_numpyarray.test_numpy()
fields.test_numpyarray.test_serialize()
```

3.2.5 fields.test_slice module

```
class fields.test_slice.SliceSchema(extra=None, only=None, exclude=(), prefix=u",  
                                     strict=None, many=False, context=None, load_only=(),  
                                     dump_only=(), partial=False)
```

Bases: `argschema.schemas.ArgSchema`

This schema is designed to be a schema_type for an ArgSchemaParser object

Table 25: SliceSchema

key	description	default	field_type	json_type
a	slice the dataset	slice(None, None, None)	<i>Slice</i>	str
log_level	set the logging level of the module	ERROR	<i>LogLevel</i>	str

opts = <marshmallow.schema.SchemaOpts object>

```
fields.test_slice.test_bad_slice()
fields.test_slice.test_slice()
```

3.2.6 Module contents

- `genindex`
- `modindex`
- `search`

4.1 Support/Contribute

We are planning on occasional updating this tool with no fixed schedule. Community involvement is encouraged through both issues and pull requests. Please make pull requests against the dev branch, as we will test changes there before merging into master.

- Issue Tracker: <https://github.com/AllenInstitute/argschema/issues>
- Source Code: <https://github.com/AllenInstitute/argschema>

4.2 License

The project is licensed under the BSD Clause 2 license with a non-commercial use clause.

a

- `argschema`, 34
- `argschema.argschema_parser`, 28
- `argschema.autodoc`, 33
- `argschema.fields`, 17
- `argschema.fields.files`, 15
- `argschema.fields.loglevel`, 16
- `argschema.fields.numpyarrays`, 16
- `argschema.fields.slice`, 17
- `argschema.schemas`, 30
- `argschema.utils`, 30
- `argschema.validate`, 33

f

- `fields`, 45
- `fields.test_files`, 42
- `fields.test_loglevel`, 44
- `fields.test_numpyarray`, 44
- `fields.test_slice`, 45

t

- `test_argschema_parser`, 39
- `test_autodoc`, 41
- `test_first_test`, 35
- `test_output`, 38
- `test_utils`, 40

A

args_to_dict() (in module argschema.utils), 30
 ArgSchema (class in argschema.schemas), 30
 argschema (module), 34
 argschema.argschema_parser (module), 28
 argschema.autodoc (module), 33
 argschema.fields (module), 17
 argschema.fields.files (module), 15
 argschema.fields.loglevel (module), 16
 argschema.fields.numpyarrays (module), 16
 argschema.fields.slice (module), 17
 argschema.schemas (module), 30
 argschema.utils (module), 30
 argschema.validate (module), 33
 ArgSchemaParser (class in argschema.argschema_parser), 28
 ArgSchemaYamlParser (class in argschema.argschema_parser), 29

B

bad_test_recursive_schema() (in module test_first_test), 38
 BadExampleRecursiveSchema (class in test_first_test), 35
 BadRecursiveSchema (class in test_first_test), 35
 BaseballSituation (class in test_utils), 40
 BasicInputDir (class in fields.test_files), 42
 BasicInputFile (class in fields.test_files), 42
 BasicOutputDir (class in fields.test_files), 43
 BasicOutputFile (class in fields.test_files), 43
 Bool (in module argschema.fields), 26
 Boolean (class in argschema.fields), 22
 build_schema_arguments() (in module argschema.utils), 30

C

cli_error_dict() (in module argschema.utils), 31
 Constant (class in argschema.fields), 26
 context (argschema.fields.Field attribute), 18

D

Date (class in argschema.fields), 24
 DateTime (class in argschema.fields), 23
 DAYS (argschema.fields.TimeDelta attribute), 25
 Decimal (class in argschema.fields), 22
 default_configurable_sinks (argschema.argschema_parser.ArgSchemaParser attribute), 28
 default_configurable_sinks (argschema.argschema_parser.ArgSchemaYamlParser attribute), 29
 default_configurable_sources (argschema.argschema_parser.ArgSchemaParser attribute), 28
 in default_configurable_sources (argschema.argschema_parser.ArgSchemaYamlParser attribute), 29
 in default_error_messages (argschema.fields.Boolean attribute), 23
 default_error_messages (argschema.fields.Date attribute), 24
 default_error_messages (argschema.fields.DateTime attribute), 24
 default_error_messages (argschema.fields.Decimal attribute), 22
 default_error_messages (argschema.fields.Email attribute), 25
 default_error_messages (argschema.fields.Field attribute), 18
 default_error_messages (argschema.fields.Float attribute), 23
 default_error_messages (argschema.fields.FormattedString attribute), 23
 default_error_messages (argschema.fields.Integer attribute), 22
 default_error_messages (argschema.fields.List attribute), 20
 default_error_messages (argschema.fields.Mapping attribute), 20

default_error_messages (argschema.fields.Nested attribute), 19

default_error_messages (argschema.fields.Number attribute), 21

default_error_messages (argschema.fields.String attribute), 21

default_error_messages (argschema.fields.Time attribute), 24

default_error_messages (argschema.fields.TimeDelta attribute), 25

default_error_messages (argschema.fields.Tuple attribute), 21

default_error_messages (argschema.fields.Url attribute), 25

default_error_messages (argschema.fields.UUID attribute), 21

DEFAULT_FORMAT (argschema.fields.Date attribute), 24

DEFAULT_FORMAT (argschema.fields.DateTime attribute), 24

default_output_schema (argschema.argschema_parser.ArgSchemaParser attribute), 28

default_schema (argschema.argschema_parser.ArgSchemaParser attribute), 28

default_schema (test_argschema_parser.MyParser attribute), 39

DefaultSchema (class in argschema.schemas), 30

DESERIALIZATION_FUNCS (argschema.fields.Date attribute), 24

DESERIALIZATION_FUNCS (argschema.fields.DateTime attribute), 24

deserialize() (argschema.fields.Field method), 18

Dict (class in argschema.fields), 20

dump() (in module argschema.utils), 31

E

Email (class in argschema.fields), 25

ExampleRecursiveSchema (class in test_first_test), 36

F

fail() (argschema.fields.Field method), 18

falsy (argschema.fields.Boolean attribute), 23

Field (class in argschema.fields), 17

fields (module), 45

fields.test_files (module), 42

fields.test_loglevel (module), 44

fields.test_numpyarray (module), 44

fields.test_slice (module), 45

Float (class in argschema.fields), 23

FormattedString (class in argschema.fields), 23

Function (class in argschema.fields), 26

G

get_description_from_field() (in module argschema.utils), 31

get_field_def_from_schema() (in module argschema.utils), 31

get_output_json() (argschema.argschema_parser.ArgSchemaParser method), 28

get_type_from_field() (in module argschema.utils), 32

get_value() (argschema.fields.Field method), 18

get_value() (argschema.fields.List method), 20

get_value() (argschema.fields.Tuple method), 21

H

HOURS (argschema.fields.TimeDelta attribute), 25

I

initialize_logger() (argschema.argschema_parser.ArgSchemaParser static method), 29

InputDir (class in argschema.fields), 26

InputDir (class in argschema.fields.files), 15

InputFile (class in argschema.fields), 27

InputFile (class in argschema.fields.files), 15

Int (in module argschema.fields), 26

Integer (class in argschema.fields), 22

L

List (class in argschema.fields), 20

load() (in module argschema.utils), 32

load_schema_with_defaults() (argschema.argschema_parser.ArgSchemaParser method), 29

LocalDateTime (class in argschema.fields), 24

localtime (argschema.fields.DateTime attribute), 24

localtime (argschema.fields.LocalDateTime attribute), 24

LogLevel (class in argschema.fields), 27

LogLevel (class in argschema.fields.loglevel), 16

M

main() (in module argschema), 34

make_object() (argschema.schemas.DefaultSchema method), 30

Mapping (class in argschema.fields), 19

mapping_type (argschema.fields.Dict attribute), 20

mapping_type (argschema.fields.Mapping attribute), 20

merge_value() (in module argschema.utils), 32

Method (class in argschema.fields), 25

MICROSECONDS (argschema.fields.TimeDelta attribute), 25

MILLISECONDS (argschema.fields.TimeDelta attribute), 25

MINUTES (argschema.fields.TimeDelta attribute), 25

ModelFit (class in test_first_test), 36

ModeOutputDirSchema (class in fields.test_files), 43

MyExtension (class in test_first_test), 36

MyNestedSchema (class in test_argschema_parser), 39

get_description_from_field()

(in module

- MyNestedSchemaWithDefaults (class in test_argschema_parser), 39
- MyOutputSchema (class in test_output), 38
- MyParser (class in test_argschema_parser), 39
- MySchema (class in test_argschema_parser), 39
- MySchema2 (class in test_argschema_parser), 40
- MyShorterExtension (class in test_first_test), 36
- ## N
- Nested (class in argschema.fields), 19
- num_type (argschema.fields.Decimal attribute), 22
- num_type (argschema.fields.Float attribute), 23
- num_type (argschema.fields.Integer attribute), 22
- num_type (argschema.fields.Number attribute), 21
- Number (class in argschema.fields), 21
- NumpyArray (class in argschema.fields), 27
- NumpyArray (class in argschema.fields.numpyarrays), 16
- NumpyFileuint16 (class in fields.test_numpyarray), 44
- ## O
- OBJ_TYPE (argschema.fields.Date attribute), 24
- OBJ_TYPE (argschema.fields.DateTime attribute), 24
- options (argschema.fields.LogLevel attribute), 27
- options (argschema.fields.loglevel.LogLevel attribute), 16
- opts (argschema.schemas.ArgSchema attribute), 30
- opts (argschema.schemas.DefaultSchema attribute), 30
- opts (fields.test_files.BasicInputDir attribute), 42
- opts (fields.test_files.BasicInputFile attribute), 43
- opts (fields.test_files.BasicOutputDir attribute), 43
- opts (fields.test_files.BasicOutputFile attribute), 43
- opts (fields.test_files.ModeOutputDirSchema attribute), 43
- opts (fields.test_numpyarray.NumpyFileuint16 attribute), 44
- opts (fields.test_slice.SliceSchema attribute), 45
- opts (test_argschema_parser.MyNestedSchema attribute), 39
- opts (test_argschema_parser.MyNestedSchemaWithDefaults attribute), 39
- opts (test_argschema_parser.MySchema attribute), 40
- opts (test_argschema_parser.MySchema2 attribute), 40
- opts (test_autodoc.SchemaWithQuotedDescriptions attribute), 42
- opts (test_first_test.BadExampleRecursiveSchema attribute), 35
- opts (test_first_test.BadRecursiveSchema attribute), 35
- opts (test_first_test.ExampleRecursiveSchema attribute), 36
- opts (test_first_test.ModelFit attribute), 36
- opts (test_first_test.MyExtension attribute), 36
- opts (test_first_test.MyShorterExtension attribute), 37
- opts (test_first_test.PopulationSelectionParameters attribute), 37
- in opts (test_first_test.PopulationSelectionPaths attribute), 37
- opts (test_first_test.RecursiveSchema attribute), 37
- opts (test_first_test.SimpleExtension attribute), 38
- opts (test_output.MyOutputSchema attribute), 38
- opts (test_utils.BaseballSituation attribute), 41
- opts (test_utils.Player attribute), 41
- output() (argschema.argschema_parser.ArgSchemaParser method), 29
- OutputDir (class in argschema.fields), 27
- OutputDir (class in argschema.fields.files), 16
- OutputFile (class in argschema.fields), 26
- OutputFile (class in argschema.fields.files), 16
- ## P
- Player (class in test_utils), 41
- PopulationSelectionParameters (class in test_first_test), 37
- PopulationSelectionPaths (class in test_first_test), 37
- process_schemas() (in module argschema.autodoc), 33
- prune_dict_with_none() (in module argschema.utils), 32
- ## R
- Raw (class in argschema.fields), 19
- RecursiveSchema (class in test_first_test), 37
- root (argschema.fields.Field attribute), 18
- ## S
- schema (argschema.fields.Nested attribute), 19
- schema_argparser() (in module argschema.utils), 32
- SCHEMA_OPTS_VAR_NAME (argschema.fields.Date attribute), 24
- SCHEMA_OPTS_VAR_NAME (argschema.fields.DateTime attribute), 24
- SchemaWithQuotedDescriptions (class in test_autodoc), 41
- SECONDS (argschema.fields.TimeDelta attribute), 25
- SERIALIZATION_FUNCS (argschema.fields.Date attribute), 24
- SERIALIZATION_FUNCS (argschema.fields.DateTime attribute), 24
- serialize() (argschema.fields.Field method), 18
- Shape (class in argschema.validate), 33
- simple_extension_file() (in module test_first_test), 38
- SimpleExtension (class in test_first_test), 37
- Slice (class in argschema.fields), 27
- Slice (class in argschema.fields.slice), 17
- SliceSchema (class in fields.test_slice), 45
- smart_merge() (in module argschema.utils), 33
- Str (in module argschema.fields), 26
- String (class in argschema.fields), 21

T

`test_access_inputfile_failed()` (in module `fields.test_files`), 44

`test_alt_output()` (in module `test_output`), 38

`test_argschema_parser` (module), 39

`test_autodoc` (module), 41

`test_autodoc()` (in module `test_autodoc`), 42

`test_autodoc_argschemaparser()` (in module `test_autodoc`), 42

`test_autodoc_list()` (in module `test_autodoc`), 42

`test_autodoc_myparser()` (in module `test_autodoc`), 42

`test_autodoc_nested()` (in module `test_autodoc`), 42

`test_autodoc_quotes()` (in module `test_autodoc`), 42

`test_autodoc_recursive_nested()` (in module `test_autodoc`), 42

`test_autodoc_slice()` (in module `test_autodoc`), 42

`test_bad_data()` (in module `fields.test_numpyarray`), 44

`test_bad_input_json_argparse()` (in module `test_first_test`), 38

`test_bad_inputdir()` (in module `fields.test_files`), 44

`test_bad_option()` (in module `fields.test_loglevel`), 44

`test_bad_output()` (in module `test_output`), 38

`test_bad_path()` (in module `test_first_test`), 38

`test_bad_shape()` (in module `fields.test_numpyarray`), 45

`test_bad_slice()` (in module `fields.test_slice`), 45

`test_basic_inputdir()` (in module `fields.test_files`), 44

`test_boolean_command_line()` (in module `test_argschema_parser`), 40

`test_david_example()` (in module `test_first_test`), 38

`test_enoent_outputfile_failed()` (in module `fields.test_files`), 44

`test_failed_mode()` (in module `fields.test_files`), 44

`test_first_test` (module), 35

`test_inputdir_no_access()` (in module `fields.test_files`), 44

`test_log_catch()` (in module `test_first_test`), 38

`test_merge_value_add()` (in module `test_utils`), 41

`test_merge_value_fail()` (in module `test_utils`), 41

`test_merge_value_subtract()` (in module `test_utils`), 41

`test_mode_output_osdir()` (in module `fields.test_files`), 44

`test_my_default_nested_parser()` (in module `test_argschema_parser`), 40

`test_my_parser()` (in module `test_argschema_parser`), 40

`test_numpy()` (in module `fields.test_numpyarray`), 45

`test_option_list()` (in module `fields.test_loglevel`), 44

`test_output` (module), 38

`test_output()` (in module `test_output`), 38

`test_output_dir_bad_location()` (in module `fields.test_files`), 44

`test_output_dir_bad_permission()` (in module `fields.test_files`), 44

`test_output_dir_basic()` (in module `fields.test_files`), 44

`test_output_file_relative()` (in module `fields.test_files`), 44

`test_output_path()` (in module `fields.test_files`), 44

`test_output_path_cannot_write()` (in module `fields.test_files`), 44

`test_output_path_noopath()` (in module `fields.test_files`), 44

`test_output_unvalidated()` (in module `test_output`), 38

`test_outputfile_no_write()` (in module `fields.test_files`), 44

`test_outputfile_not_a_path()` (in module `fields.test_files`), 44

`test_parser_output()` (in module `test_argschema_parser`), 40

`test_recursive_schema()` (in module `test_first_test`), 38

`test_relative_file_input()` (in module `fields.test_files`), 44

`test_relative_file_input_failed()` (in module `fields.test_files`), 44

`test_schema_argparser_with_baseball()` (in module `test_utils`), 41

`test_serialize()` (in module `fields.test_numpyarray`), 45

`test_simple_description()` (in module `test_first_test`), 38

`test_simple_example()` (in module `test_first_test`), 38

`test_simple_extension_fail()` (in module `test_first_test`), 38

`test_simple_extension_pass()` (in module `test_first_test`), 38

`test_simple_extension_required()` (in module `test_first_test`), 38

`test_simple_extension_write_debug_level()` (in module `test_first_test`), 38

`test_simple_extension_write_overwrite()` (in module `test_first_test`), 38

`test_simple_extension_write_overwrite_list()` (in module `test_first_test`), 38

`test_simple_extension_write_pass()` (in module `test_first_test`), 38

`test_slice()` (in module `fields.test_slice`), 45

`test_smart_merge()` (in module `test_utils`), 41

`test_smart_merge_add()` (in module `test_utils`), 41

`test_smart_merge_nested()` (in module `test_utils`), 41

`test_smart_merge_none()` (in module `test_utils`), 41

`test_smart_merge_not_none()` (in module `test_utils`), 41

`test_smart_merge_same()` (in module `test_utils`), 41

`test_tmp_output_cleanput()` (in module `test_output`), 39

`test_utils` (module), 40

`Time` (class in `argschema.fields`), 24

`TimeDelta` (class in `argschema.fields`), 24

`truthy` (`argschema.fields.Boolean` attribute), 23

`Tuple` (class in `argschema.fields`), 21

U

`Url` (class in `argschema.fields`), 25

`URL` (in module `argschema.fields`), 25

`UUID` (class in `argschema.fields`), 21

V

`validate_input_path()` (in module `argschema.fields.files`),

[16](#)

`validate_outpath()` (in module `argschema.fields.files`), [16](#)

`validate_rst_lines()` (in module `test_autodoc`), [42](#)

W

`WEEKS` (`argschema.fields.TimeDelta` attribute), [25](#)