
argschema Documentation

Release 1.1.1

Forrest Collman, David Feng

Jul 02, 2021

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 | 44 |
| 4 | Indices and tables | 49 |
| 4.1 | Support/Contribute | 49 |
| 4.2 | License | 49 |
| | Python Module Index | 51 |
| | Index | 53 |

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 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
mymodule.py:10: DeprecationWarning: The 'warn' method is deprecated, use 'warning'
↳instead
  mod.logger.warn("this module does nothing useful")
{'a': 42, 'log_level': 'ERROR'}
```

```
$ python mymodule.py --a 2
mymodule.py:10: DeprecationWarning: The 'warn' method is deprecated, use 'warning'
↳instead
```

(continues on next page)

(continued from previous page)

```
mod.logger.warn("this module does nothing useful")
{'a': 2, 'log_level': 'ERROR'}
```

```
$ python mymodule.py --a 2 --log_level WARNING
mymodule.py:10: DeprecationWarning: The 'warn' method is deprecated, use 'warning'
↳instead
  mod.logger.warn("this module does nothing useful")
WARNING:argschema.argschema_parser:this module does nothing useful
{'a': 2, 'log_level': 'WARNING'}
```

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

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

MySchema:
  --input_json INPUT_JSON
                        file path of input json file
  --output_json OUTPUT_JSON
                        file path to output json file
  --log_level LOG_LEVEL
                        set the logging level of the module (default=ERROR)
  --a A                  my first parameter (default=42)
```

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
mymodule.py:10: DeprecationWarning: The 'warn' method is deprecated, use 'warning'
↳instead
  mod.logger.warn("this module does nothing useful")
{'log_level': 'ERROR', 'a': 99, 'input_json': 'myinput.json'}
```

or override a parameter if you want

```
$ python mymodule.py --input_json myinput.json --a 100
mymodule.py:10: DeprecationWarning: The 'warn' method is deprecated, use 'warning'
↳instead
```

(continues on next page)

(continued from previous page)

```
mod.logger.warn("this module does nothing useful")
{'input_json': 'myinput.json', 'a': 100, 'log_level': 'ERROR'}
```

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 "/home/docs/checkouts/readthedocs.org/user_builds/argschema/envs/stable/lib/
python3.7/site-packages/argschema-3.0.1-py3.7.egg/argschema/argschema_parser.py",
line 175, in __init__
  File "/home/docs/checkouts/readthedocs.org/user_builds/argschema/envs/stable/lib/
python3.7/site-packages/argschema-3.0.1-py3.7.egg/argschema/argschema_parser.py",
line 276, in load_schema_with_defaults
  File "/home/docs/checkouts/readthedocs.org/user_builds/argschema/envs/stable/lib/
python3.7/site-packages/argschema-3.0.1-py3.7.egg/argschema/argschema_utils.py",
line 418, in load
  File "/home/docs/checkouts/readthedocs.org/user_builds/argschema/envs/stable/lib/
python3.7/site-packages/marshmallow/schema.py", line 707, in load
    postprocess=True,
  File "/home/docs/checkouts/readthedocs.org/user_builds/argschema/envs/stable/lib/
python3.7/site-packages/marshmallow/schema.py", line 867, in _do_load
    raise exc
marshmallow.exceptions.ValidationError: {'a': ['Not a valid integer.']}
```

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."
  ]
}
```

1.1.2 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. *fields* contains all the built-in marshmallow fields, but also some useful custom ones, such as *InputFile*, *OutputFile*, *InputDir* that validate that the paths exist and have the proper permissions to allow files to be read or written.

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

Finally, an important Field to know is *Nested*, which allows you to define heirarchical nested structures. Note, that

if you use Nested schemas, your Nested schemas should subclass `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

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
MySchema
{'log_level': 'ERROR'}
MySchemaFill
{'log_level': 'ERROR', 'nest': {'b': 2, 'a': 1}}
```

```
$ python nested_example.py --nest.a 4
MySchema
{'nest': {'b': 2, 'a': 4}, 'log_level': 'ERROR'}
MySchemaFill
{'nest': {'b': 2, 'a': 4}, 'log_level': 'ERROR'}
```

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
template_module.py:14: DeprecationWarning: `np.float` is a deprecated alias for the
↳ builtin `float`. To silence this warning, use `float` by itself. Doing this will
↳ not modify any behavior and is safe. If you specifically wanted the numpy scalar
↳ type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/
↳ release/1.20.0-notes.html#deprecations
  array = NumpyArray(dtype=np.float, required=True,
template_module.py:29: DeprecationWarning: `np.float` is a deprecated alias for the
↳ builtin `float`. To silence this warning, use `float` by itself. Doing this will
↳ not modify any behavior and is safe. If you specifically wanted the numpy scalar
↳ type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/
↳ release/1.20.0-notes.html#deprecations
  inc_array = NumpyArray(dtype=np.float, required=True,
{'inc': {'array': array([0., 2., 5.]),
        'increment': 2,
        'name': 'from_command',
        'write_output': True},
 'log_level': 'ERROR',
 'output_json': 'output_command.json'}

```

```

$ python template_module.py
  --input_json input.json

```

(continues on next page)

(continued from previous page)

```

--output_json output_fromjson.json
template_module.py:14: DeprecationWarning: `np.float` is a deprecated alias for the
↳builtin `float`. To silence this warning, use `float` by itself. Doing this will
↳not modify any behavior and is safe. If you specifically wanted the numpy scalar
↳type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/
↳release/1.20.0-notes.html#deprecations
    array = NumpyArray(dtype=np.float, required=True,
template_module.py:29: DeprecationWarning: `np.float` is a deprecated alias for the
↳builtin `float`. To silence this warning, use `float` by itself. Doing this will
↳not modify any behavior and is safe. If you specifically wanted the numpy scalar
↳type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/
↳release/1.20.0-notes.html#deprecations
    inc_array = NumpyArray(dtype=np.float, required=True,
{'inc': {'array': array([3., 2., 1.]),
        'increment': 1,
        'name': 'from_json',
        'write_output': True},
 'input_json': 'input.json',
 'log_level': 'ERROR',
 'output_json': 'output_fromjson.json'}

```

```

$ python template_module.py
template_module.py:14: DeprecationWarning: `np.float` is a deprecated alias for the
↳builtin `float`. To silence this warning, use `float` by itself. Doing this will
↳not modify any behavior and is safe. If you specifically wanted the numpy scalar
↳type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/
↳release/1.20.0-notes.html#deprecations
    array = NumpyArray(dtype=np.float, required=True,
template_module.py:29: DeprecationWarning: `np.float` is a deprecated alias for the
↳builtin `float`. To silence this warning, use `float` by itself. Doing this will
↳not modify any behavior and is safe. If you specifically wanted the numpy scalar
↳type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/
↳release/1.20.0-notes.html#deprecations
    inc_array = NumpyArray(dtype=np.float, required=True,
{'inc': {'array': array([0., 2., 5.]),
        'increment': 5,
        'name': 'from_dictionary',
        'write_output': True},
 'log_level': 'ERROR',
 'output_json': 'output_dictionary.json'}

```

1.1.3 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.

An exception to this rule is list formatting. If a schema contains a `List` and does not set the *cli_as_single_argument*

keyword argument to `True`, lists will be parsed as `--list_name <value1> <value2> ...`. In argschema 2.0 lists will be parsed in the same way as other arguments, as it allows more flexibility in list types and more clearly represents the intended data structure.

An example script showing old and new list settings:

Listing 6: deprecated_example.py

```
from argschema import ArgSchema, ArgSchemaParser
from argschema.fields import List, Float

class MySchema(ArgSchema):
    list_old = List(Float, default=[1.1, 2.2, 3.3],
                   description="float list with deprecated cli")
    list_new = List(Float, default=[4.4, 5.5, 6.6],
                   cli_as_single_argument=True,
                   description="float list with supported cli")

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

Running this code can demonstrate the differences in command-line usage:

```
$ python deprecated_example.py --help
/home/docs/checkouts/readthedocs.org/user_builds/argschema/envs/stable/lib/python3.7/
↳ site-packages/argschema-3.0.1-py3.7.egg/argschema/utils.py:346: FutureWarning: '--
↳ list_old' is using old-style command-line syntax with each element as a separate_
↳ argument. This will not be supported in argschema after 2.0. See http://argschema.
↳ readthedocs.io/en/master/user/intro.html#command-line-specification for details.
usage: deprecated_example.py [-h] [--input_json INPUT_JSON]
                             [--output_json OUTPUT_JSON]
                             [--log_level LOG_LEVEL]
                             [--list_old [LIST_OLD [LIST_OLD ...]]]
                             [--list_new LIST_NEW]

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

MySchema:
  --input_json INPUT_JSON
                        file path of input json file
  --output_json OUTPUT_JSON
                        file path to output json file
  --log_level LOG_LEVEL
                        set the logging level of the module (default=ERROR)
  --list_old [LIST_OLD [LIST_OLD ...]]
                        float list with deprecated cli (default=[1.1, 2.2,
                        3.3])
  --list_new LIST_NEW   float list with supported cli (default=[4.4, 5.5,
                        6.6])
```

```
$ python deprecated_example.py --list_old 9.1 8.2 7.3 --list_new [6.4,5.5,4.6]
/home/docs/checkouts/readthedocs.org/user_builds/argschema/envs/stable/lib/python3.7/
↳ site-packages/argschema-3.0.1-py3.7.egg/argschema/utils.py:346: FutureWarning: '--
↳ list_old' is using old-style command-line syntax with each element as a separate_
↳ argument. This will not be supported in argschema after 2.0. See http://argschema.
↳ readthedocs.io/en/master/user/intro.html#command-line-specification for details.
```

(continued from previous page)

```
{'list_old': [9.1, 8.2, 7.3], 'log_level': 'ERROR', 'list_new': [6.4, 5.5, 4.6]}
```

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

Listing 7: 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!"]],
                       cli_as_single_argument=True,
                       description="list of lists of strings")
    int_list = List(Int, default=[1, 2, 3],
                   cli_as_single_argument=True,
                   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] [--input_json INPUT_JSON]
                    [--output_json OUTPUT_JSON] [--log_level LOG_LEVEL]
                    [--array ARRAY] [--string_list STRING_LIST]
                    [--int_list INT_LIST] [--nested.a NESTED.A]
                    [--nested.b NESTED.B]

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

MySchema:
  --input_json INPUT_JSON
                        file path of input json file
  --output_json OUTPUT_JSON
                        file path to output json file
  --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]])
  --string_list STRING_LIST
                        list of lists of strings (default=[['hello', 'world'],
                        ['lists!']])
  --int_list INT_LIST   list of ints (default=[1, 2, 3])
```

(continues on next page)

(continued from previous page)

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

We can set some values and observe the output:

```
$ python cli_example.py --nested.b 0 --string_list "[['foo', 'bar'], ['baz', 'buz']]"
{'int_list': [1, 2, 3], 'log_level': 'ERROR', 'nested': {'b': False, 'a': 42},
↳ 'string_list': [['foo', 'bar'], ['baz', 'buz']], 'array': array([[1, 2, 3],
    [4, 5, 6]], dtype=uint8)}
```

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 25, in <module>
    mod = ArgSchemaParser(schema_type=MySchema)
  File "/home/docs/checkouts/readthedocs.org/user_builds/argschema/envs/stable/lib/
↳ python3.7/site-packages/argschema-3.0.1-py3.7.egg/argschema/argschema_parser.py",
↳ line 160, in __init__
  File "/home/docs/checkouts/readthedocs.org/user_builds/argschema/envs/stable/lib/
↳ python3.7/site-packages/argschema-3.0.1-py3.7.egg/argschema/utils.py", line 138, in
↳ args_to_dict
marshmallow.exceptions.ValidationError: {
  "array": [
    "Command-line argument can't cast to NumpyArray"
  ]
}
```

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

1.1.4 Sphinx Documentation

argschema comes with a autodocumentation feature for Sphinx which will help you automatically add documentation of your Schemas and 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)
```

1.1.5 Installation

install via source code

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

or pip


```
$ pip install argschema
```

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`

exception `argschema.fields.files.OutputDirModeException`

Bases: `Exception`

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)

class `argschema.fields.files.WindowsNamedTemporaryFile` (*dir=None, mode=None*)

Bases: `object`

`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`, **`meta-data`)

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 = {'null': 'Field may not be null.', 'required': 'Missing data'}

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 list or tuple of fields to marshal. If *None*, all fields are marshalled. 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 = {'type': '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 `builtins.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 `builtins.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.'}
```

```
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 = {'invalid': 'Not a valid tuple.'}
```

```
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, **metadata)
Bases: marshmallow.fields.Field
```

A string field.

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

```
default_error_messages = {'invalid': 'Not a valid string.', 'invalid_utf8': 'Not a v
```

```
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 = {'invalid_uuid': '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 = {'invalid': 'Not a valid number.', 'too_large': 'Number too
```

```
num_type
```

```
alias of builtins.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 = {'invalid': 'Not a valid integer.'}
```

```
num_type
```

```
alias of builtins.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 = {'special': 'Special numeric values (nan or infinity) are no
```

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 = {'invalid': 'Not a valid boolean.'}

falsy = {'f', 'NO', 'Off', 'No', 'false', 0, 'OFF', '0', 'N', 'False', 'FALSE', 'n', 'N'}
Default falsy values.

truthy = {1, 't', 'TRUE', 'y', '1', 'Yes', 'on', 'True', 'true', 'Y', 'On', 'yes', 'YE'}
Default truthy values.

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

num_type
alias of `builtins.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 = 'iso'

DESERIALIZATION_FUNCS = {'iso': <function from_iso_datetime>, 'iso8601': <function from_iso_datetime>}

OBJ_TYPE = 'datetime'

```
SCHEMA_OPTS_VAR_NAME = 'datetimeformat'  
SERIALIZATION_FUNCS = {'iso': <function isoformat>, 'iso8601': <function isoformat>,  
default_error_messages = {'format': '"{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 = {'format': '"{input}" cannot be formatted as a time.', 'invalid': '"{input}" is not a valid time.'}
```

```
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 = 'iso'
```

```
DESERIALIZATION_FUNCS = {'iso': <function from_iso_date>, 'iso8601': <function from_iso_date>}
```

```
OBJ_TYPE = 'date'
```

```
SCHEMA_OPTS_VAR_NAME = 'dateformat'
```

```
SERIALIZATION_FUNCS = {'iso': <function to_iso_date>, 'iso8601': <function to_iso_date>}
```

```
default_error_messages = {'format': '"{input}" cannot be formatted as a date.', 'invalid': '"{input}" is not a valid date.'}
```

```
class argschema.fields.TimeDelta (precision='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 = 'days'
```

```
HOURS = 'hours'
```

```
MICROSECONDS = 'microseconds'
```

```
MILLISECONDS = 'milliseconds'
```

```
MINUTES = 'minutes'
```

```
SECONDS = 'seconds'
```

```
WEEKS = 'weeks'
```

```
default_error_messages = {'format': '{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': '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': '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, re-  
                                   quired=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`

`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.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.OptionList (options, **kwargs)
```

Bases: `marshmallow.fields.Field`

OptionList is a **marshmallow field which enforces that this field** is one of a finite set of options. `OptionList(options,*args,**kwargs)` where options is a list of json compatible options which this option will be enforced to belong

Parameters

- **options** (*list*) – A list of python objects of which this field must be one of
- **kwargs** (*dict*) – the same as any `Field` receives

```
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 `: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

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,                   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 instead of `-input_json`
- **`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_json`, used by `self.output`
- **`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 ‘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_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 dictionary 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*, *output_path=None*, ***json_dump_options*)
 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
- **output_path** (*str*) – path to save to output file, optional (with default to `self.mod['output_json']` location)
- ****json_dump_options** – will be passed through to `json.dump`

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

`argschema.argschema_parser.contains_non_default_schemas` (*schema*, *schema_list=[]*)
 returns True if this schema contains a schema which was not an instance of `DefaultSchema`

Parameters

- **schema** (*marshmallow.Schema*) – schema to check
- **schema_list** – (Default value = [])

Returns does this schema only contain schemas which are subclassed from `schemas.DefaultSchema`

Return type `bool`

`argschema.argschema_parser.fill_defaults` (*schema*, *args*)
 DEPRECATED, function to fill in default values from schema into args bug: goes into an infinite loop when there is a recursively defined schema

Parameters

- **schema** (*marshmallow.Schema*) – schema to get defaults from
- **args** –

Returns dictionary with missing default values filled in

Return type dict

`argschema.argschema_parser.is_recursive_schema` (*schema*, *schema_list*=[])
 returns true if this schema contains recursive elements

Parameters

- **schema** (*marshmallow.Schema*) – schema to check
- **schema_list** – (Default value = [])

Returns does this schema contain any recursively defined schemas

Return type bool

2.1.4 argschema.deprecated module

class `argschema.deprecated.JsonModule` (*input_data=None*, *schema_type=None*, *output_schema_type=None*, *args=None*, *logger_name='argschema.argschema_parser'*)

Bases: `argschema.argschema_parser.ArgSchemaParser`

deprecated name of ArgSchemaParser

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

class `argschema.deprecated.ModuleParameters` (*only=None*, *exclude=()*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*, *unknown=None*)

Bases: `argschema.schemas.ArgSchema`

deprecated name of ArgSchema

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

Table 1: ModuleParameters

| key | description | default | field_type | json_type |
|--------------------------|-------------------------------------|---------|-------------------------|-----------|
| <code>input_json</code> | file path of input json file | NA | <code>InputFile</code> | str |
| <code>output_json</code> | file path to output json file | NA | <code>OutputFile</code> | str |
| <code>log_level</code> | set the logging level of the module | ERROR | <code>LogLevel</code> | str |

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

2.1.5 argschema.schemas module

class `argschema.schemas.ArgSchema` (*only=None*, *exclude=()*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*, *unknown=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 2: ArgSchema

| key | description | default | field_type | json_type |
|-------------|-------------------------------------|---------|-------------------|-----------|
| input_json | file path of input json file | NA | <i>InputFile</i> | str |
| output_json | file path to output json file | NA | <i>OutputFile</i> | str |
| log_level | set the logging level of the module | ERROR | <i>LogLevel</i> | str |

opts = <marshmallow.schema.SchemaOpts object>

class argschema.schemas.DefaultSchema (*only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=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, **kwargs*)

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>

2.1.6 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, schema=None*)

function to convert namespace returned by argparse into a nested dictionary

Parameters

- **argsobj** (*argparse.Namespace*) – Namespace object returned by standard argparse.parse function
- **schema** (*marshmallow.Schema*) – Optional schema 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

Raises `marshmallow.ValidationError` – if the dictionary does not conform to the schema

`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_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

Raises `marshmallow.ValidationError` – if the dictionary does not conform to the schema

`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*)

given a jsonschema, build an `argparse.ArgumentParser`

Parameters **schema** (`argschema.schemas.ArgSchema`) – schema to build an argparser from

Returns the represents the schema

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

2.1.7 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

2.1.8 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 sphnix conf.py as follows

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

2.1.9 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 (only=None, exclude=(), many=False,
                                                context=None, load_only=(),
                                                dump_only=(), partial=False, un-
                                                known=None)
```

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 |
|-------------|-------------------------------------|------------|---------------------------------|-----------|
| input_json | file path of input json file | NA | <code>InputFile</code> | str |
| output_json | file path to output json file | NA | <code>OutputFile</code> | str |
| 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 (only=None, exclude=(), many=False, con-
                                                text=None, load_only=(), dump_only=(),
                                                partial=False, unknown=None)
```

Bases: `marshmallow.schema.Schema`

Table 2: BadRecursiveSchema

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

opts = <marshmallow.schema.SchemaOpts object>

```
class test_first_test.ExampleRecursiveSchema (only=None, exclude=(), many=False, context=None,
                                             load_only=(), dump_only=(), partial=False, unknown=None)
```

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 |
|-------------|-------------------------------------|------------|------------------------|-----------|
| input_json | file path of input json file | NA | <i>InputFile</i> | str |
| output_json | file path to output json file | NA | <i>OutputFile</i> | str |
| log_level | set the logging level of the module | ERROR | <i>LogLevel</i> | str |
| tree | no description | (REQUIRED) | <i>RecursiveSchema</i> | dict |

opts = <marshmallow.schema.SchemaOpts object>

```
class test_first_test.ModelFit (only=None, exclude=(), many=False, context=None,
                                load_only=(), dump_only=(), partial=False, unknown=None)
```

Bases: *argschema.schemas.DefaultSchema*

Table 4: ModelFit

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

opts = <marshmallow.schema.SchemaOpts object>

```
class test_first_test.MyExtension (only=None, exclude=(), many=False, context=None,
                                   load_only=(), dump_only=(), partial=False, unknown=None)
```

Bases: *argschema.schemas.DefaultSchema*

Table 5: MyExtension

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

opts = <marshmallow.schema.SchemaOpts object>

```
class test_first_test.MyExtensionOld (only=None, exclude=(), many=False, context=None,
                                       load_only=(), dump_only=(), partial=False, unknown=None)
```

Bases: *marshmallow.schema.Schema*

Table 6: MyExtensionOld

| key | description | default | field_type | json_type |
|-----|--------------------|---------|----------------|-----------|
| a | a string | NA | <i>String</i> | str |
| b | an integer | NA | <i>Integer</i> | int |
| c | an integer | 10 | <i>Integer</i> | int |
| d | a list of integers | NA | unknown | unknown |

opts = <marshmallow.schema.SchemaOpts object>

```
class test_first_test.MyPostLoadClass (input_data=None, schema_type=None, out-
                                     put_schema_type=None, args=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 *MySchemaPostLoad*

default_schema

alias of *MySchemaPostLoad*

run()

```
class test_first_test.MySchemaPostLoad (only=None, exclude=(), many=False, con-
                                       text=None, load_only=(), dump_only=(), par-
                                       tial=False, unknown=None)
```

Bases: *argschema.schemas.ArgSchema*

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

Table 7: MySchemaPostLoad

| key | description | default | field_type | json_type |
|-------------|-------------------------------------|------------|-------------------|-----------|
| input_json | file path of input json file | NA | <i>InputFile</i> | str |
| output_json | file path to output json file | NA | <i>OutputFile</i> | str |
| log_level | set the logging level of the module | ERROR | <i>LogLevel</i> | str |
| xid | no description | (REQUIRED) | <i>Integer</i> | int |

my_post (*data*, ***kwargs*)

opts = <marshmallow.schema.SchemaOpts object>

```
class test_first_test.MyShorterExtension (only=None, exclude=(), many=False, con-
                                       text=None, load_only=(), dump_only=(),
                                       partial=False, unknown=None)
```

Bases: *argschema.schemas.ArgSchema*

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

Table 8: MyShorterExtension

| key | description | default | field_type | json_type |
|-------------|-------------------------------------|---------|-------------------|-----------|
| input_json | file path of input json file | NA | <i>InputFile</i> | str |
| output_json | file path to output json file | NA | <i>OutputFile</i> | str |
| log_level | set the logging level of the module | ERROR | <i>LogLevel</i> | str |
| a | a string | NA | <i>String</i> | str |
| b | an integer | NA | <i>Integer</i> | int |
| c | an integer | 10 | <i>Integer</i> | int |
| d | a list of integers | NA | unknown | unknown |

opts = <marshmallow.schema.SchemaOpts object>

```
class test_first_test.PopulationSelectionParameters (only=None,          exclude=(),
                                                    many=False,       context=None,
                                                    load_only=(),    dump_only=(),
                                                    partial=False,  unknown=None)
```

Bases: *argschema.schemas.ArgSchema*

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

Table 9: PopulationSelectionParameters

| key | description | default | field_type | json_type |
|-------------|-------------------------------------|---------|---------------------------------|-----------|
| input_json | file path of input json file | NA | <i>InputFile</i> | str |
| output_json | file path to output json file | NA | <i>OutputFile</i> | str |
| log_level | set the logging level of the module | ERROR | <i>LogLevel</i> | str |
| paths | no description | NA | <i>PopulationSelectionPaths</i> | dict |

opts = <marshmallow.schema.SchemaOpts object>

```
class test_first_test.PopulationSelectionPaths (only=None,  exclude=(),  many=False,
                                                context=None, load_only=(),
                                                dump_only=(), partial=False,  un-
                                                known=None)
```

Bases: *argschema.schemas.DefaultSchema*

Table 10: 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 (only=None,  exclude=(),  many=False,  context=None,
                                       load_only=(),  dump_only=(),  partial=False,  un-
                                       known=None)
```

Bases: *argschema.schemas.DefaultSchema*

Table 11: 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` (*only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: `argschema.schemas.ArgSchema`

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

Table 12: SimpleExtension

| key | description | default | field_type | json_type |
|--------------------------|-------------------------------------|------------|--------------------------|-----------|
| <code>input_json</code> | file path of input json file | NA | <code>InputFile</code> | str |
| <code>output_json</code> | file path to output json file | NA | <code>OutputFile</code> | str |
| <code>log_level</code> | set the logging level of the module | ERROR | <code>LogLevel</code> | str |
| <code>test</code> | no description | (REQUIRED) | <code>MyExtension</code> | dict |

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

class `test_first_test.SimpleExtensionOld` (*only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: `argschema.schemas.ArgSchema`

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

Table 13: SimpleExtensionOld

| key | description | default | field_type | json_type |
|--------------------------|-------------------------------------|---------|--------------------------|-----------|
| <code>input_json</code> | file path of input json file | NA | <code>InputFile</code> | str |
| <code>output_json</code> | file path to output json file | NA | <code>OutputFile</code> | str |
| <code>log_level</code> | set the logging level of the module | ERROR | <code>LogLevel</code> | str |
| <code>test</code> | no description | None | <code>MyExtension</code> | dict |

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

```
test_first_test.bad_test_recursive_schema()
test_first_test.david_data(tmpdir)
test_first_test.simple_extension_file(tmpdir_factory)
test_first_test.test_bad_input_json_argparse()
test_first_test.test_bad_path()
test_first_test.test_david_example(tmpdir_factory, david_data)
test_first_test.test_log_catch()
test_first_test.test_post_load_schema()
test_first_test.test_recursive_schema()
test_first_test.test_simple_description()
test_first_test.test_simple_example(tmpdir)
test_first_test.test_simple_extension_fail()
test_first_test.test_simple_extension_old_pass()
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 14: MyOutputSchema

| key | description | default | field_type | json_type |
|-----|--------------------------|------------|------------|-----------|
| a | a simple string | (REQUIRED) | String | str |
| b | a default integer | 5 | Integer | int |
| M | a numpy array of answers | (REQUIRED) | unknown | unknown |

```
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_cleanup (tmpdir)
```

3.1.3 test_argschema_parser module

```
class test_argschema_parser.MyNestedSchema (only=None, exclude=(), many=False, con-
                                             text=None, load_only=(), dump_only=(), par-
                                             tial=False, unknown=None)
Bases: argschema.schemas.DefaultSchema
```

Table 15: MyNestedSchema

| key | description | default | field_type | json_type |
|-----|------------------|------------|------------|-----------|
| one | nested integer | (REQUIRED) | Integer | int |
| two | a nested boolean | (REQUIRED) | Boolean | bool |

```
opts = <marshmallow.schema.SchemaOpts object>
class test_argschema_parser.MyNestedSchemaWithDefaults (only=None, exclude=(),
                                                         many=False, context=None,
                                                         load_only=(), dump_only=(),
                                                         partial=False, unknown=None)
Bases: argschema.schemas.DefaultSchema
```

Table 16: MyNestedSchemaWithDefaults

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

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

```
class test_argschema_parser.MyParser (input_data=None, schema_type=None, out-
                                     put_schema_type=None, args=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 (only=None, exclude=(), many=False, context=None,
                                     load_only=(), dump_only=(), partial=False, un-
                                     known=None)
```

Bases: `argschema.schemas.ArgSchema`

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

Table 17: MySchema

| key | description | default | field_type | json_type |
|-------------|-------------------------------------|------------|-----------------------------|-----------|
| input_json | file path of input json file | NA | <code>InputFile</code> | str |
| output_json | file path to output json file | NA | <code>OutputFile</code> | str |
| log_level | set the logging level of the module | ERROR | <code>LogLevel</code> | str |
| a | parameter a | (REQUIRED) | <code>Integer</code> | int |
| b | optional b string parameter | my value | <code>String</code> | str |
| nest | a nested schema | NA | <code>MyNestedSchema</code> | dict |

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

```
class test_argschema_parser.MySchema2 (only=None, exclude=(), many=False, context=None,
                                       load_only=(), dump_only=(), partial=False, un-
                                       known=None)
```

Bases: `argschema.schemas.ArgSchema`

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

Table 18: MySchema2

| key | description | default | field_type | json_type |
|-------------|-------------------------------------|------------|---|-----------|
| input_json | file path of input json file | NA | <code>InputFile</code> | str |
| output_json | file path to output json file | NA | <code>OutputFile</code> | str |
| log_level | set the logging level of the module | ERROR | <code>LogLevel</code> | str |
| a | parameter a | (REQUIRED) | <code>Integer</code> | int |
| b | optional b string parameter | my value | <code>String</code> | str |
| nest | a nested schema | NA | <code>MyNestedSchemaWithDefaults</code> | dict |

`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(only=None, exclude=(), many=False, context=None,
                                   load_only=(), dump_only=(), partial=False, un-
                                   known=None)
```

Bases: *argschema.schemas.ArgSchema*

A description of a baseball situation

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

Table 19: BaseballSituation

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

opts = <marshmallow.schema.SchemaOpts object>

```
class test_utils.Player(only=None, exclude=(), many=False, context=None, load_only=(),
                        dump_only=(), partial=False, unknown=None)
```

Bases: *argschema.schemas.DefaultSchema*

player information

Table 20: Player

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

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 (*only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *argschema.schemas.ArgSchema*

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

Table 21: SchemaWithQuotedDescriptions

| key | description | default | field_type | json_type |
|-------------|---|------------|-------------------|-----------|
| input_json | file path of input json file | NA | <i>InputFile</i> | str |
| output_json | file path to output json file | NA | <i>OutputFile</i> | str |
| log_level | set the logging level of the module | ERROR | <i>LogLevel</i> | str |
| a | something that is 'quoted' is problematic | (REQUIRED) | <i>Integer</i> | int |

opts = <marshmallow.schema.SchemaOpts object>

```

test_autodoc.test_autodoc()
test_autodoc.test_autodoc_argshemaparser()
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_deprecated module

class `fields.test_deprecated.OptionSchema` (*only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: `argschema.schemas.ArgSchema`

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

Table 22: OptionSchema

| key | description | default | field_type | json_type |
|--------------------------|-------------------------------------|------------|-------------------------|-----------|
| <code>input_json</code> | file path of input json file | NA | <code>InputFile</code> | str |
| <code>output_json</code> | file path to output json file | NA | <code>OutputFile</code> | str |
| <code>log_level</code> | set the logging level of the module | ERROR | <code>LogLevel</code> | str |
| <code>a</code> | one of 1,2,3 | (REQUIRED) | <code>OptionList</code> | ? |

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

`fields.test_deprecated.test_bad_option()`

`fields.test_deprecated.test_option_list()`

3.2.3 fields.test_files module

class `fields.test_files.BasicInputDir` (*only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: `argschema.schemas.ArgSchema`

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

Table 23: BasicInputDir

| key | description | default | field_type | json_type |
|--------------------------|-------------------------------------|------------|-------------------------|-----------|
| <code>input_json</code> | file path of input json file | NA | <code>InputFile</code> | str |
| <code>output_json</code> | file path to output json file | NA | <code>OutputFile</code> | str |
| <code>log_level</code> | set the logging level of the module | ERROR | <code>LogLevel</code> | str |
| <code>input_dir</code> | a simple file | (REQUIRED) | <code>InputDir</code> | str |

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

class `fields.test_files.BasicInputFile` (*only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: `argschema.schemas.ArgSchema`

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

Table 24: BasicInputFile

| key | description | default | field_type | json_type |
|-------------|-------------------------------------|------------|-------------------|-----------|
| input_json | file path of input json file | NA | <i>InputFile</i> | str |
| output_json | file path to output json file | NA | <i>OutputFile</i> | str |
| 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** (*only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *argschema.schemas.ArgSchema*

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

Table 25: BasicOutputDir

| key | description | default | field_type | json_type |
|-------------|-------------------------------------|------------|-------------------|-----------|
| input_json | file path of input json file | NA | <i>InputFile</i> | str |
| output_json | file path to output json file | NA | <i>OutputFile</i> | str |
| 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** (*only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *argschema.schemas.ArgSchema*

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

Table 26: BasicOutputFile

| key | description | default | field_type | json_type |
|-------------|-------------------------------------|------------|-------------------|-----------|
| input_json | file path of input json file | NA | <i>InputFile</i> | str |
| output_json | file path to output json file | NA | <i>OutputFile</i> | str |
| log_level | set the logging level of the module | ERROR | <i>LogLevel</i> | str |
| output_file | a simple output file | (REQUIRED) | <i>OutputFile</i> | str |

opts = <marshmallow.schema.SchemaOpts object>

class fields.test_files.**ModeOutputDirSchema** (*only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *argschema.schemas.ArgSchema*

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

Table 27: ModeOutputDirSchema

| key | description | default | field_type | json_type |
|-------------|-------------------------------------|------------|-------------------|-----------|
| input_json | file path of input json file | NA | <i>InputFile</i> | str |
| output_json | file path to output json file | NA | <i>OutputFile</i> | str |
| 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_noapath()
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()
fields.test_files.test_windows_outdir_mode_fail()

```

3.2.4 fields.test_loglevel module

```

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

```

3.2.5 fields.test_numpyarray module

class fields.test_numpyarray.NumpyFileuint16 (*only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *argschema.schemas.ArgSchema*

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

Table 28: NumpyFileuint16

| key | description | default | field_type | json_type |
|-------------|---|------------|-------------------|-----------|
| input_json | file path of input json file | NA | <i>InputFile</i> | str |
| output_json | file path to output json file | NA | <i>OutputFile</i> | str |
| log_level | set the logging level of the module | ERROR | <i>LogLevel</i> | str |
| a | list of lists representing a uint16 numpy array | (REQUIRED) | unknown | unknown |

```

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()

```

3.2.6 fields.test_slice module

```

class fields.test_slice.SliceSchema(only=None, exclude=(), many=False, context=None,
                                     load_only=(), dump_only=(), partial=False, un-
                                     known=None)

```

Bases: *argschema.schemas.ArgSchema*

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

Table 29: SliceSchema

| key | description | default | field_type | json_type |
|-------------|-------------------------------------|----------------------------|-------------------|-----------|
| input_json | file path of input json file | NA | <i>InputFile</i> | str |
| output_json | file path to output json file | NA | <i>OutputFile</i> | str |
| log_level | set the logging level of the module | ERROR | <i>LogLevel</i> | str |
| a | slice the dataset | slice(None, None, None) | <i>Slice</i> | str |

```

opts = <marshmallow.schema.SchemaOpts object>
fields.test_slice.test_bad_slice()
fields.test_slice.test_slice()

```

3.2.7 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, 34
- argschema.deprecated, 30
- argschema.fields, 17
 - argschema.fields.files, 15
 - argschema.fields.loglevel, 16
 - argschema.fields.numpyarrays, 16
 - argschema.fields.slice, 17
- argschema.schemas, 30
- argschema.utils, 31
- argschema.validate, 34

f

- fields, 47
 - fields.test_deprecated, 44
 - fields.test_files, 44
 - fields.test_loglevel, 46
 - fields.test_numpyarray, 46
 - fields.test_slice, 47

t

- test_argschema_parser, 40
- test_autodoc, 43
- test_first_test, 35
- test_output, 40
- test_utils, 42

A

args_to_dict() (in module *argschema.utils*), 31
 ArgSchema (class in *argschema.schemas*), 30
 argschema (module), 34
 argschema.argschema_parser (module), 28
 argschema.autodoc (module), 34
 argschema.deprecated (module), 30
 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), 31
 argschema.validate (module), 34
 ArgSchemaParser (class in *argschema.argschema_parser*), 28

B

bad_test_recursive_schema() (in module *test_first_test*), 39
 BadExampleRecursiveSchema (class in *test_first_test*), 35
 BadRecursiveSchema (class in *test_first_test*), 35
 BaseballSituation (class in *test_utils*), 42
 BasicInputDir (class in *fields.test_files*), 44
 BasicInputFile (class in *fields.test_files*), 44
 BasicOutputDir (class in *fields.test_files*), 45
 BasicOutputFile (class in *fields.test_files*), 45
 Bool (in module *argschema.fields*), 26
 Boolean (class in *argschema.fields*), 23
 build_schema_arguments() (in module *argschema.utils*), 31

C

cli_error_dict() (in module *argschema.utils*), 32
 Constant (class in *argschema.fields*), 26
 contains_non_default_schemas() (in module *argschema.argschema_parser*), 29

context (*argschema.fields.Field* attribute), 18

D

Date (class in *argschema.fields*), 24
 DateTime (class in *argschema.fields*), 23
 david_data() (in module *test_first_test*), 39
 DAYS (*argschema.fields.TimeDelta* attribute), 25
 Decimal (class in *argschema.fields*), 22
 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.Integer* attribute), 22
 default_error_messages (*argschema.fields.List* attribute), 21
 default_error_messages (*argschema.fields.Mapping* attribute), 20
 default_error_messages (*argschema.fields.Nested* attribute), 20
 default_error_messages (*argschema.fields.Number* attribute), 22
 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*), 23

default_output_schema (*argschema.argschema_parser.ArgSchemaParser attribute*), 28

default_schema (*argschema.argschema_parser.ArgSchemaParser attribute*), 28

default_schema (*test_argschema_parser.MyParser attribute*), 41

default_schema (*test_first_test.MyPostLoadClass attribute*), 37

DefaultSchema (*class in argschema.schemas*), 31

DESERIALIZATION_FUNCS (*argschema.fields.Date attribute*), 24

DESERIALIZATION_FUNCS (*argschema.fields.DateTime attribute*), 23

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

Dict (*class in argschema.fields*), 20

dump() (*in module argschema.utils*), 32

E

Email (*class in argschema.fields*), 35

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*), 47

fields.test_deprecated (*module*), 44

fields.test_files (*module*), 44

fields.test_loglevel (*module*), 46

fields.test_numpyarray (*module*), 46

fields.test_slice (*module*), 47

fill_defaults() (*in module argschema.argschema_parser*), 29

Float (*class in argschema.fields*), 23

Function (*class in argschema.fields*), 26

G

get_description_from_field() (*in module argschema.utils*), 32

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

H

HOURS (*argschema.fields.TimeDelta attribute*), 25

I

initialize_logger() (*argschema.argschema_parser.ArgSchemaParser static method*), 28

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

is_recursive_schema() (*in module argschema.argschema_parser*), 30

J

JsonModule (*class in argschema.deprecated*), 30

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*), 31

Mapping (*class in argschema.fields*), 20

mapping_type (*argschema.fields.Dict attribute*), 20

mapping_type (*argschema.fields.Mapping attribute*), 20

merge_value() (*in module argschema.utils*), 33

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*), 45

ModuleParameters (*class in argschema.deprecated*), 30

- `my_post()` (*test_first_test.MySchemaPostLoad method*), 37
- `MyExtension` (*class in test_first_test*), 36
- `MyExtensionOld` (*class in test_first_test*), 36
- `MyNestedSchema` (*class in test_argschema_parser*), 40
- `MyNestedSchemaWithDefaults` (*class in test_argschema_parser*), 40
- `MyOutputSchema` (*class in test_output*), 40
- `MyParser` (*class in test_argschema_parser*), 41
- `MyPostLoadClass` (*class in test_first_test*), 37
- `MySchema` (*class in test_argschema_parser*), 41
- `MySchema2` (*class in test_argschema_parser*), 41
- `MySchemaPostLoad` (*class in test_first_test*), 37
- `MyShorterExtension` (*class in test_first_test*), 37
- ## 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*), 22
- `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*), 46
- ## O
- `OBJ_TYPE` (*argschema.fields.Date attribute*), 24
- `OBJ_TYPE` (*argschema.fields.DateTime attribute*), 23
- `OptionList` (*class in argschema.fields*), 27
- `options` (*argschema.fields.LogLevel attribute*), 27
- `options` (*argschema.fields.loglevel.LogLevel attribute*), 16
- `OptionSchema` (*class in fields.test_deprecated*), 44
- `opts` (*argschema.deprecated.ModuleParameters attribute*), 30
- `opts` (*argschema.schemas.ArgSchema attribute*), 31
- `opts` (*argschema.schemas.DefaultSchema attribute*), 31
- `opts` (*fields.test_deprecated.OptionSchema attribute*), 44
- `opts` (*fields.test_files.BasicInputDir attribute*), 44
- `opts` (*fields.test_files.BasicInputFile attribute*), 45
- `opts` (*fields.test_files.BasicOutputDir attribute*), 45
- `opts` (*fields.test_files.BasicOutputFile attribute*), 45
- `opts` (*fields.test_files.ModeOutputDirSchema attribute*), 45
- `opts` (*fields.test_numpyarray.NumpyFileuint16 attribute*), 46
- `opts` (*fields.test_slice.SliceSchema attribute*), 47
- `opts` (*test_argschema_parser.MyNestedSchema attribute*), 40
- `opts` (*test_argschema_parser.MyNestedSchemaWithDefaults attribute*), 41
- `opts` (*test_argschema_parser.MySchema attribute*), 41
- `opts` (*test_argschema_parser.MySchema2 attribute*), 41
- `opts` (*test_autodoc.SchemaWithQuotedDescriptions attribute*), 43
- `opts` (*test_first_test.BadExampleRecursiveSchema attribute*), 35
- `opts` (*test_first_test.BadRecursiveSchema attribute*), 36
- `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.MyExtensionOld attribute*), 37
- `opts` (*test_first_test.MySchemaPostLoad attribute*), 37
- `opts` (*test_first_test.MyShorterExtension attribute*), 38
- `opts` (*test_first_test.PopulationSelectionParameters attribute*), 38
- `opts` (*test_first_test.PopulationSelectionPaths attribute*), 38
- `opts` (*test_first_test.RecursiveSchema attribute*), 38
- `opts` (*test_first_test.SimpleExtension attribute*), 39
- `opts` (*test_first_test.SimpleExtensionOld attribute*), 39
- `opts` (*test_output.MyOutputSchema attribute*), 40
- `opts` (*test_utils.BaseballSituation attribute*), 42
- `opts` (*test_utils.Player attribute*), 42
- `output()` (*argschema.argschema_parser.ArgSchemaParser method*), 29
- `OutputDir` (*class in argschema.fields*), 27
- `OutputDir` (*class in argschema.fields.files*), 16
- `OutputDirModeException`, 16
- `OutputFile` (*class in argschema.fields*), 26
- `OutputFile` (*class in argschema.fields.files*), 16
- ## P
- `Player` (*class in test_utils*), 42
- `PopulationSelectionParameters` (*class in test_first_test*), 38
- `PopulationSelectionPaths` (*class in test_first_test*), 38
- `process_schemas()` (*in module argschema.autodoc*), 34
- `prune_dict_with_none()` (*in module argschema.utils*), 33
- ## R
- `Raw` (*class in argschema.fields*), 19
- `RecursiveSchema` (*class in test_first_test*), 38
- `root` (*argschema.fields.Field attribute*), 18
- `run()` (*test_first_test.MyPostLoadClass method*), 37
- ## S
- `schema` (*argschema.fields.Nested attribute*), 20

- [schema_argparser\(\) \(in module argschema.utils\), 33](#)
[SCHEMA_OPTS_VAR_NAME \(argschema.fields.Date attribute\), 24](#)
[SCHEMA_OPTS_VAR_NAME \(argschema.fields.DateTime attribute\), 23](#)
[SchemaWithQuotedDescriptions \(class in test_autodoc\), 43](#)
[SECONDS \(argschema.fields.TimeDelta attribute\), 25](#)
[SERIALIZATION_FUNCS \(argschema.fields.Date attribute\), 24](#)
[SERIALIZATION_FUNCS \(argschema.fields.DateTime attribute\), 24](#)
[serialize\(\) \(argschema.fields.Field method\), 19](#)
[Shape \(class in argschema.validate\), 34](#)
[simple_extension_file\(\) \(in module test_first_test\), 39](#)
[SimpleExtension \(class in test_first_test\), 38](#)
[SimpleExtensionOld \(class in test_first_test\), 39](#)
[Slice \(class in argschema.fields\), 27](#)
[Slice \(class in argschema.fields.slice\), 17](#)
[SliceSchema \(class in fields.test_slice\), 47](#)
[smart_merge\(\) \(in module argschema.utils\), 33](#)
[Str \(in module argschema.fields\), 26](#)
[String \(class in argschema.fields\), 21](#)
- ## T
- [test_access_inputfile_failed\(\) \(in module fields.test_files\), 46](#)
[test_alt_output\(\) \(in module test_output\), 40](#)
[test_argschema_parser \(module\), 40](#)
[test_autodoc \(module\), 43](#)
[test_autodoc\(\) \(in module test_autodoc\), 43](#)
[test_autodoc_argschemaparser\(\) \(in module test_autodoc\), 43](#)
[test_autodoc_list\(\) \(in module test_autodoc\), 43](#)
[test_autodoc_myparser\(\) \(in module test_autodoc\), 43](#)
[test_autodoc_nested\(\) \(in module test_autodoc\), 43](#)
[test_autodoc_quotes\(\) \(in module test_autodoc\), 43](#)
[test_autodoc_recursive_nested\(\) \(in module test_autodoc\), 43](#)
[test_autodoc_slice\(\) \(in module test_autodoc\), 43](#)
[test_bad_data\(\) \(in module fields.test_numpyarray\), 47](#)
[test_bad_input_json_argparse\(\) \(in module test_first_test\), 39](#)
[test_bad_inputdir\(\) \(in module fields.test_files\), 46](#)
[test_bad_option\(\) \(in module fields.test_deprecated\), 44](#)
[test_bad_option\(\) \(in module fields.test_loglevel\), 46](#)
[test_bad_output\(\) \(in module test_output\), 40](#)
[test_bad_path\(\) \(in module test_first_test\), 39](#)
[test_bad_shape\(\) \(in module fields.test_numpyarray\), 47](#)
[test_bad_slice\(\) \(in module fields.test_slice\), 47](#)
[test_basic_inputdir\(\) \(in module fields.test_files\), 46](#)
[test_boolean_command_line\(\) \(in module test_argschema_parser\), 41](#)
[test_david_example\(\) \(in module test_first_test\), 39](#)
[test_enoent_outputfile_failed\(\) \(in module fields.test_files\), 46](#)
[test_failed_mode\(\) \(in module fields.test_files\), 46](#)
[test_first_test \(module\), 35](#)
[test_inputdir_no_access\(\) \(in module fields.test_files\), 46](#)
[test_log_catch\(\) \(in module test_first_test\), 39](#)
[test_merge_value_add\(\) \(in module test_utils\), 42](#)
[test_merge_value_fail\(\) \(in module test_utils\), 42](#)
[test_merge_value_subtract\(\) \(in module test_utils\), 42](#)
[test_mode_output_osdir\(\) \(in module fields.test_files\), 46](#)
[test_my_default_nested_parser\(\) \(in module test_argschema_parser\), 42](#)
[test_my_parser\(\) \(in module test_argschema_parser\), 42](#)
[test_numpy\(\) \(in module fields.test_numpyarray\), 47](#)
[test_option_list\(\) \(in module fields.test_deprecated\), 44](#)
[test_option_list\(\) \(in module fields.test_loglevel\), 46](#)
[test_output \(module\), 40](#)
[test_output\(\) \(in module test_output\), 40](#)
[test_output_dir_bad_location\(\) \(in module fields.test_files\), 46](#)
[test_output_dir_bad_permission\(\) \(in module fields.test_files\), 46](#)
[test_output_dir_basic\(\) \(in module fields.test_files\), 46](#)
[test_output_file_relative\(\) \(in module fields.test_files\), 46](#)
[test_output_path\(\) \(in module fields.test_files\), 46](#)
[test_output_path_cannot_write\(\) \(in module fields.test_files\), 46](#)
[test_output_path_noapath\(\) \(in module fields.test_files\), 46](#)

test_output_unvalidated() (in module *test_output*), 40
 test_outputfile_no_write() (in module *fields.test_files*), 46
 test_outputfile_not_a_path() (in module *fields.test_files*), 46
 test_parser_output() (in module *test_argschema_parser*), 42
 test_post_load_schema() (in module *test_first_test*), 39
 test_recursive_schema() (in module *test_first_test*), 39
 test_relative_file_input() (in module *fields.test_files*), 46
 test_relative_file_input_failed() (in module *fields.test_files*), 46
 test_schema_argparser_with_baseball() (in module *test_utils*), 42
 test_serialize() (in module *fields.test_numpyarray*), 47
 test_simple_description() (in module *test_first_test*), 39
 test_simple_example() (in module *test_first_test*), 39
 test_simple_extension_fail() (in module *test_first_test*), 39
 test_simple_extension_old_pass() (in module *test_first_test*), 39
 test_simple_extension_pass() (in module *test_first_test*), 39
 test_simple_extension_required() (in module *test_first_test*), 39
 test_simple_extension_write_debug_level() (in module *test_first_test*), 39
 test_simple_extension_write_overwrite() (in module *test_first_test*), 40
 test_simple_extension_write_overwrite_list() (in module *test_first_test*), 40
 test_simple_extension_write_pass() (in module *test_first_test*), 40
 test_slice() (in module *fields.test_slice*), 47
 test_smart_merge() (in module *test_utils*), 42
 test_smart_merge_add() (in module *test_utils*), 42
 test_smart_merge_nested() (in module *test_utils*), 43
 test_smart_merge_none() (in module *test_utils*), 43
 test_smart_merge_not_none() (in module *test_utils*), 43
 test_smart_merge_same() (in module *test_utils*), 43
 test_tmp_output_cleanput() (in module *test_output*), 40
 test_utils (module), 42
 test_windows_outdir_mode_fail() (in module *fields.test_files*), 46
 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*), 43

W

WEEKS (*argschema.fields.TimeDelta* attribute), 25
 WindowsNamedTemporaryFile (class in *argschema.fields.files*), 16