

---

# **argschema Documentation**

***Release 1.1.1***

**Forrest Collman, David Feng**

**Jan 31, 2018**



---

## Contents

---

<b>1</b>	<b>The User Guide</b>	<b>3</b>
1.1	User Guide . . . . .	3
1.2	Indices and tables . . . . .	11
<b>2</b>	<b>API</b>	<b>13</b>
2.1	argschema package . . . . .	13
<b>3</b>	<b>TESTS</b>	<b>33</b>
3.1	test . . . . .	33
3.2	fields package . . . . .	40
<b>4</b>	<b>Indices and tables</b>	<b>45</b>
4.1	Support/Contribute . . . . .	45
4.2	License . . . . .	45
	<b>Python Module Index</b>	<b>47</b>



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

In particular it will allow you to

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

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



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

## 1.1 User Guide

### 1.1.1 Installation

install via source code

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

or pip

```
$ pip install argschema
```

### 1.1.2 Your First Module

Listing 1.1: 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)
    print(mod.args)
```

running this code produces

```
$ python mymodule.py
{'a': 42, 'log_level': u'ERROR'}
$ python mymodule.py --a 2
{'a': 2, 'log_level': u'ERROR'}
$ python mymodule.py --a 2 --log_level WARNING
{'a': 2, 'log_level': u'WARNING'}
WARNING:argschema.argschema_parser:this program does nothing useful
$ python mymodule.py -h
usage: mymodule.py [-h] [--a A] [--output_json OUTPUT_JSON]
                  [--log_level LOG_LEVEL] [--input_json INPUT_JSON]

optional arguments:
-h, --help            show this help message and exit
--a A                my first parameter
--output_json OUTPUT_JSON
                    file path to output json file
--log_level LOG_LEVEL
                    set the logging level of the module
--input_json INPUT_JSON
                    file path of input json file
```

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 1.2: myinput.json

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

```
$ python mymodule.py --input_json myinput.json
{'a': 99, 'log_level': u'ERROR', 'input_json': u'myinput.json'}
```

or override a parameter if you want

```
$ python mymodule.py --input_json myinput.json --a 100
{'a': 100, 'log_level': u'ERROR', 'input_json': u'myinput.json'}
```

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 myinput.json --a 5!
usage: mymodule.py [-h] [--a A] [--output_json OUTPUT_JSON]
                  [--log_level LOG_LEVEL] [--input_json INPUT_JSON]
mymodule.py: error: argument --a: invalid int value: '5!'
```

or from a dictionary



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

### 1.1.3 Fields

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

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

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

The template\_module example shows how you might combine these features to define a more complex parameter structure.

Listing 1.3: template\_module.py

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

# 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']

    # 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)
```

so now if run the example commands found in run\_template.sh

Listing 1.4: 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
{u'name': u'from_command', u'inc_array': [2.0, 4.0, 7.0]}
```

```
$ python template_module.py \
    --input_json input.json \
    --output_json output_fromjson.json
{'u'name': u'from_json', u'inc_array': [4.0, 3.0, 2.0]}
$ python template_module.py
{'u'name': u'from_dictionary', u'inc_array': [5.0, 7.0, 10.0]}
```

### 1.1.4 Command-Line Specification

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

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

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 1.5: 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
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.
warnings.warn(msg, FutureWarning)
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]
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.
warnings.warn(warn_msg, FutureWarning)
{'log_level': 'ERROR', 'list_new': [6.4, 5.5, 4.6], 'list_old': [9.1, 8.2, 7.3]}

```

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

Listing 1.6: cli\_example.py

```

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

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

class MySchema(ArgSchema):
    array = NumpyArray(default=[[1, 2, 3],[4, 5, 6]], dtype="uint8",
                        description="my example array")
    string_list = List(List(Str,
                            default=["hello", "world"], ["lists!"]),
                        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])

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], 'string_list': [['foo', 'bar'], ['baz', 'buz']], 'array':
↳array([[1, 2, 3],
        [4, 5, 6]], dtype=uint8), 'log_level': 'ERROR', 'nested': {'a': 42, 'b': False}}

```

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)
    ...
marshmallow.exceptions.ValidationError: {
  "array": [
    "Command-line argument can't cast to NumpyArray"
  ]
}

```

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

### 1.1.5 Alternate Sources/Sinks

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

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

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

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

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

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

`argschema.sources.ArgSink` instance.

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

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

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

class UrlSource(ArgSource):
    ConfigSchema = UrlSourceConfig

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

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

so now a `UrlArgSchemaParser` would expect command line flags of 'input\_host' and 'input\_url', and optionally 'input\_port', 'input\_protocol' (or look for them in `input_data`) and will look to download the json from that http

location via requests. In addition, an existing `argschema.ArgSchemaParser` module could be simply passed a configured `UrlSource` via `input_source`, and it would get its parameters from there.

### 1.1.6 Sphinx Documentation

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

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

```
from argschema.autodoc import process_schemas

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

## 1.2 Indices and tables

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





This contains the complete documentation of the api

## 2.1 argschema package

### 2.1.1 Subpackages

**argschema.fields package**

**Submodules**

**argschema.fields.files module**

marshmallow fields related to validating input and output file paths

```
class argschema.fields.files.InputDir (default=<marshmallow.missing>, attribute=None,
                                         load_from=None, dump_to=None, error=None, val-
                                         idate=None, required=False, allow_none=None,
                                         load_only=False, dump_only=False,
                                         missing=<marshmallow.missing>, error_
                                         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>, attribute=None,
                                         load_from=None, dump_to=None, error=None,
                                         validate=None, required=False, allow_none=None,
                                         load_only=False, dump_only=False,
                                         missing=<marshmallow.missing>, error_
                                         ror_messages=None, **metadata)
```

Bases: `marshmallow.fields.String`

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

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

Bases: `marshmallow.fields.String`

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

#### Parameters

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

**class** `argschema.fields.files.OutputFile` (*default=<marshmallow.missing>, attribute=None, load\_from=None, dump\_to=None, error=None, validate=None, required=False, allow\_none=None, load\_only=False, dump\_only=False, missing=<marshmallow.missing>, error\_messages=None, \*\*metadata*)

Bases: `marshmallow.fields.String`

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

`argschema.fields.files.validate_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, load\_from=None, dump\_to=None, error=None, validate=None, required=False, allow\_none=None, load\_only=False, dump\_only=False, missing=<marshmallow.missing>, error\_messages=None, \*\*metadata)

Bases: `marshmallow.base.FieldABC`

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

### Parameters

- **default** – If set, this value will be used during serialization if the input value is missing. If not set, the field will be excluded from the serialized output if the input value is missing. May be a value or a callable.
- **attribute** (*str*) – The name of the attribute to get the value from. If *None*, assumes the attribute has the same name as the field.
- **load\_from** (*str*) – Additional key to look for when deserializing. Will only be checked if the field’s name is not found on the input dictionary. If checked, it will return this parameter on error.
- **dump\_to** (*str*) – Field name to use as a key when serializing.
- **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.

#### **context**

The context dictionary for the parent *Schema*.

**default\_error\_messages** = {u'null': u'Field may not be null.', u'validator\_failed': u

**deserialize** (*value*, *attr=None*, *data=None*)

Deserialize *value*.

**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** (*attr*, *obj*, *accessor=None*, *default=<marshmallow.missing>*)

Return the value for a given key from an object.

#### **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*)

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*.

**Raises *ValidationError*** – In case of formatting problem

```
class argschema.fields.Raw (default=<marshmallow.missing>, attribute=None, load_from=None,
                           dump_to=None, error=None, validate=None, required=False,
                           allow_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.
- **default** – Default value to if attribute is missing or None
- **exclude** (*tuple*) – A list or tuple of fields to exclude.
- **required** – Raise an `ValidationError` during deserialization if the field, *and* any required field values specified in the *nested* schema, are not found in the data. If not a *bool* (e.g. a *str*), the provided value will be used as the message of the `ValidationError` instead of the default message.
- **only** – A tuple or string of the field(s) to marshal. If *None*, all fields will be marshalled. If a field name (string) is given, only a single value will be returned as output instead of a dictionary. This parameter takes precedence over `exclude`.
- **many** (*bool*) – Whether the field is a collection of objects.
- **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.Dict (default=<marshmallow.missing>,          attribute=None,
                             load_from=None, dump_to=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.Field`

A dict field. Supports dicts and dict-like objects.

---

**Note:** This field is only appropriate when the structure of nested data is not known. For structured data, use *Nested*.

---

New in version 2.1.0.

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

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

```
get_value(attr, obj, accessor=None)
```

Return the value for a given key from an object.

```
class argschema.fields.String(default=<marshmallow.missing>, attribute=None,
                              load_from=None, dump_to=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 = {u'invalid_utf8': u'Not a valid utf-8 string.', u'invalid':
```

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

```
Bases: marshmallow.fields.String
```

A UUID field.

```
default_error_messages = {u'invalid_uuid': u'Not a valid UUID.', u'invalid_guid': u'!
```

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

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

Base class for number fields.

#### Parameters

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

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

**num\_type**  
alias of float

**serialize** (*attr, obj, accessor=None*)

Pulls the value for the given key from the object and returns the serialized number representation. Return a string if *self.as\_string=True*, otherwise return this field's *num\_type*. Receives the same *args* and *kwargs* as *Field*.

**class** argschema.fields.Integer (*as\_string=False, \*\*kwargs*)

Bases: `marshmallow.fields.Number`

An integer field.

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

**default\_error\_messages** = {*u'invalid': u'Not a valid integer.'*}

**num\_type**  
alias of int

**class** argschema.fields.Decimal (*places=None, rounding=None, allow\_nan=False, as\_string=False, \*\*kwargs*)

Bases: `marshmallow.fields.Number`

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

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

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

### Parameters

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

New in version 1.2.0.

**default\_error\_messages** = {*u'special': u'Special numeric values are not permitted.'*}

**num\_type**  
alias of *Decimal*

```
class argschema.fields.Boolean (default=<marshmallow.missing>,          attribute=None,
                                load_from=None, dump_to=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.Field
```

A boolean field.

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

```
default_error_messages = {u'invalid': u'Not a valid boolean.'}
falsy = set([0, u'False', u'F', u'f', u'FALSE', u'0', u'false'])
truthy = set([1, u'true', u'1', u't', u'True', u'TRUE', u'T'])
```

```
class argschema.fields.FormattedString (src_str, *args, **kwargs)
Bases: marshmallow.fields.Field
```

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

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

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

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

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

A double as IEEE-754 double precision string.

**Parameters**

- **as\_string** (*bool*) – If True, format the value as a string.
- **kwargs** – The same keyword arguments that *Number* receives.

**num\_type**  
alias of 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.

```
DATEFORMAT_DESERIALIZATION_FUNCS = {u'rfc': <function from_rfc>, u'rfc822': <function
DATEFORMAT_SERIALIZATION_FUNCS = {u'rfc': <function rfcformat>, u'rfc822': <function
DEFAULT_FORMAT = u'iso'
default_error_messages = {u'invalid': u'Not a valid datetime.', u'format': u'"{input}
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,
load_from=None, dump_to=None, error=None, validate=None,
required=False, allow_none=None, load_only=False,
dump_only=False, missing=<marshmallow.missing>, error
ror_messages=None, **metadata)
```

Bases: `marshmallow.fields.Field`

ISO8601-formatted time string.

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

```
default_error_messages = {u'invalid': u'Not a valid time.', u'format': u'"{input}" c
```

```
class argschema.fields.Date (default=<marshmallow.missing>, attribute=None,
load_from=None, dump_to=None, error=None, validate=None,
required=False, allow_none=None, load_only=False,
dump_only=False, missing=<marshmallow.missing>, error
ror_messages=None, **metadata)
```

Bases: `marshmallow.fields.Field`

ISO8601-formatted date string.

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

```
default_error_messages = {u'invalid': u'Not a valid date.', u'format': u'"{input}" c
```

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

Bases: `marshmallow.fields.Field`

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

**Parameters**

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

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

```
DAYS = u'days'
```

```
MICROSECONDS = u'microseconds'
```

```
SECONDS = u'seconds'
```

```
default_error_messages = {u'invalid': u'Not a valid period of time.', u'format': u'{'
```

```
class argschema.fields.Url (relative=False, schemes=None, **kwargs)
```

```
Bases: marshmallow.fields.ValidatedField,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 = {u'invalid': u'Not a valid URL.'}
```

```
argschema.fields.URL
```

alias of *Url*

```
class argschema.fields.Email (*args, **kwargs)
```

```
Bases: marshmallow.fields.ValidatedField,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 = {u'invalid': u'Not a valid email address.'}
```

```
class argschema.fields.Method (serialize=None, deserialize=None, method_name=None,
                               **kwargs)
```

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

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

#### Parameters

- **method\_name** (*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.

```
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.
- **func**(*callable*) – This argument is to be deprecated. It exists for backwards compatibility. Use `serialize` instead.

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

```
argschema.fields.Str
    alias of String
```

```
argschema.fields.Bool
    alias of Boolean
```

```
argschema.fields.Int
    alias of 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,
                                  load_from=None, dump_to=None, error=None, validate=None,
                                  required=False, allow_none=None,
                                  load_only=False, dump_only=False, missing=<marshmallow.missing>,
                                  error_messages=None,
                                  **metadata)

Bases: marshmallow.fields.String
```

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

```
class argschema.fields.InputDir(default=<marshmallow.missing>,      attribute=None,
                                load_from=None, dump_to=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,
                                  load_from=None, dump_to=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`

`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,      input_source=None,
                                                    output_sink=None,      log-
                                                    ger_name='argschema.argschema_parser')
```

Bases: object

The main class you should sub-class to write your own argschema module. Takes input\_data, reference to a input\_json and the command line inputs and parses out the parameters and validates them against the schema\_type specified.

To subclass this and make a new schema be default, simply override the default\_schema and default\_output\_schema attributes of this class.

#### Parameters

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

**Raises** *marshmallow.ValidationError* – If the combination of input\_json, input\_data and command line arguments do not pass the validation of the schema

---

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

---

```
default_configurable_sinks = [<class 'argschema.sources.json_source.JsonSink'>]
default_configurable_sources = [<class 'argschema.sources.json_source.JsonSource'>]
default_output_schema = None
default_schema
    alias of ArgSchema
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*, *sink=None*, *\*\*sink\_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
- **sink** (`argschema.sources.source.ArgSink`) – `output_sink` to output to (optional default to `self.output_source`)
- **output\_path** (*str*) – path to save to output file, optional (with default to `self.mod['output_json']` location) (DEPRECATED path to save to output file, optional (with default to `self.mod['output_json']` location))
- **\*\*sink\_options** – will be passed through to `sink.put_dict`

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

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

Bases: `argschema.argschema_parser.ArgSchemaParser`

---

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

---

**default\_configurable\_sinks** = [`<class 'argschema.sources.yaml_source.YamlSink'>`]

**default\_configurable\_sources** = [`<class 'argschema.sources.yaml_source.YamlSource'>`]

`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*, *out-put\_schema\_type=None*, *args=None*, *in-put\_source=None*, *output\_sink=None*, *log-ger\_name='argschema.argschema\_parser'*)

Bases: `argschema.argschema_parser.ArgSchemaParser`

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

Bases: *argschema.schemas.ArgSchema*

deprecated name of ArgSchema

This schema is designed to be a schema\_type for an ArgSchemaParser object

Table 2.1: ModuleParameters

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

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

## 2.1.5 argschema.schemas module

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

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 2.2: ArgSchema

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

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

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

Bases: *marshmallow.schema.Schema*

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

```
make_object (in_data)
```

marshmallow.pre\_load decorated function for applying defaults on deserialiation

**Parameters in\_data** –

**Returns** a dictionary with default values applied

**Return type** dict

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

## 2.1.6 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, schemas=None)`

function to convert namespace returned by `argsparse` into a nested dictionary

#### Parameters

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

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

**Return type** dict

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

given a jsonschema, create a dictionary of `argsparse` 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 `argsparse`. ‘args’ is an `OrderedDict` of dictionaries with keys of the argument names with `kwargs` to build an `argsparse` 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.get_description_from_field(field)`

get the description for this `marshmallow` field

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

**Returns** description string (or None)

**Return type** str

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

function to get a field\_definition from a particular key, specified by it's parts list

**Parameters**

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

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

**Return type** marshmallow.Field or None

`argschema.utils.get_type_from_field(field)`

Get type casting for command line argument from marshmallow.Field

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

**Returns** Function to call to cast argument to

**Return type** callable

`argschema.utils.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)

`argschema.utils.prune_dict_with_none(d)`

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

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

**Returns** pruned dictionary

**Return type** dict

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

given a jsonschema, build an argparse.ArgumentParser

**Parameters**

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

**Returns** that represents the schemas

**Return type** argparse.ArgumentParser

`argschema.utils.smart_merge(a, b, path=None, merge_keys=None, overwrite_with_none=False)`

updates dictionary a with values in dictionary b being careful not to write things with None, and performing a merge on merge\_keys

**Parameters**

- **a** (*dict*) – dictionary to perform update on
- **b** (*dict*) – dictionary to perform update with
- **path** (*list*) – list of nested keys traversed so far (used for recursion) (Default value = None)
- **merge\_keys** (*list*) – list of keys to do merging on (default None)
- **overwrite\_with\_none** – (Default value = False)

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

**Return type** dict

## 2.1.7 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 (extra=None, only=(), exclude=(), prefix=u", strict=None, many=False, context=None, load_only=(), dump_only=(), partial=False)
```

Bases: `argschema.schemas.ArgSchema`

This schema is designed to be a schema\_type for an ArgSchemaParser object

Table 3.1: BadExampleRecursiveSchema

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

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

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

Bases: `marshmallow.schema.Schema`

Table 3.2: BadRecursiveSchema

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

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

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

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

This schema is designed to be a schema\_type for an ArgSchemaParser object

Table 3.3: ExampleRecursiveSchema

key	description	default	field_type	json_type
log_level	set the logging level of the module	ERROR	<a href="#">LogLevel</a>	unicode
tree	no description	(REQUIRED)	<a href="#">RecursiveSchema</a>	dict

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

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

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

Table 3.4: ModelFit

key	description	default	field_type	json_type
fit_type	no description	NA	<a href="#">String</a>	unicode
hof	no description	NA	<a href="#">InputFile</a>	unicode
hof_fit	no description	NA	<a href="#">InputFile</a>	unicode

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

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

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

Table 3.5: MyExtension

key	description	default	field_type	json_type
a	a string	(REQUIRED)	<a href="#">String</a>	unicode
c	an integer	10	<a href="#">Integer</a>	int
b	an integer	NA	<a href="#">Integer</a>	int
d	a list of integers	NA	<a href="#">List</a>	int

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

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

Bases: [marshmallow.schema.Schema](#)

Table 3.6: MyExtensionOld

key	description	default	field_type	json_type
a	a string	NA	<a href="#">String</a>	unicode
c	an integer	10	<a href="#">Integer</a>	int
b	an integer	NA	<a href="#">Integer</a>	int
d	a list of integers	NA	<a href="#">List</a>	int

```

opts = <marshmallow.schema.SchemaOpts object>

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

```

Bases: *argschema.schemas.ArgSchema*

This schema is designed to be a schema\_type for an ArgSchemaParser object

Table 3.7: MyShorterExtension

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

```

opts = <marshmallow.schema.SchemaOpts object>

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

```

Bases: *argschema.schemas.ArgSchema*

This schema is designed to be a schema\_type for an ArgSchemaParser object

Table 3.8: PopulationSelectionParameters

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

```

opts = <marshmallow.schema.SchemaOpts object>

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

```

Bases: *argschema.schemas.DefaultSchema*

Table 3.9: PopulationSelectionPaths

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

```

opts = <marshmallow.schema.SchemaOpts object>

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

```

Bases: *argschema.schemas.DefaultSchema*

Table 3.10: RecursiveSchema

key	description	default	field_type	json_type
children	children of this node	NA	<a href="#">RecursiveSchema</a>	list
name	name of this node	anonymous	<a href="#">String</a>	unicode

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

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

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

This schema is designed to be a schema\_type for an ArgSchemaParser object

Table 3.11: SimpleExtension

key	description	default	field_type	json_type
test	no description	(REQUIRED)	<a href="#">MyExtension</a>	dict
log_level	set the logging level of the module	ERROR	<a href="#">LogLevel</a>	unicode

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

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

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

This schema is designed to be a schema\_type for an ArgSchemaParser object

Table 3.12: SimpleExtensionOld

key	description	default	field_type	json_type
test	no description	None	<a href="#">MyExtension</a>	dict
log_level	set the logging level of the module	ERROR	<a href="#">LogLevel</a>	unicode

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

```
test_first_test.bad_test_recursive_schema()
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)
test_first_test.test_log_catch()
test_first_test.test_recursive_schema()
test_first_test.test_simple_description()
test_first_test.test_simple_example()
test_first_test.test_simple_extension_fail()
test_first_test.test_simple_extension_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 (extra=None, only=(), exclude=(), prefix=u", strict=None,
                                   many=False, context=None, load_only=(), dump_only=(),
                                   partial=False)
```

Bases: `argschema.schemas.DefaultSchema`

Table 3.13: MyOutputSchema

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

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

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

### 3.1.3 test\_argschema\_parser module

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

Bases: `argschema.schemas.DefaultSchema`

Table 3.14: MyNestedSchema

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

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

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

Bases: `argschema.schemas.DefaultSchema`

Table 3.15: MyNestedSchemaWithDefaults

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

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

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

Bases: `argschema.argschema_parser.ArgSchemaParser`

---

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

---

**default\_schema**

alias of `MySchema`

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

Bases: `argschema.schemas.ArgSchema`

This schema is designed to be a schema\_type for an ArgSchemaParser object

Table 3.16: MySchema

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

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

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

Bases: `argschema.schemas.ArgSchema`

This schema is designed to be a schema\_type for an ArgSchemaParser object

Table 3.17: MySchema2

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

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

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

```
test_argschema_parser.test_my_default_nested_parser ()
```

```
test_argschema_parser.test_my_parser ()
```

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

### 3.1.4 test\_utils module

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

Bases: *argschema.schemas.ArgSchema*

A description of a baseball situation

This schema is designed to be a schema\_type for an ArgSchemaParser object

Table 3.18: BaseballSituation

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

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

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

Bases: *argschema.schemas.DefaultSchema*

player information

Table 3.19: Player

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

```
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

## 3.2 fields package

### 3.2.1 Submodules

#### 3.2.2 fields.test\_deprecated module

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

Bases: *argschema.schemas.ArgSchema*

This schema is designed to be a schema\_type for an ArgSchemaParser object

Table 3.20: OptionSchema

key	description	default	field_type	json_type
a	one of 1,2,3	(REQUIRED)	OptionList	?
log_level	set the logging level of the module	ERROR	<i>LogLevel</i>	unicode

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

Bases: *argschema.schemas.ArgSchema*

This schema is designed to be a schema\_type for an ArgSchemaParser object

Table 3.21: BasicInputDir

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

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

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

Bases: *argschema.schemas.ArgSchema*

This schema is designed to be a schema\_type for an ArgSchemaParser object

Table 3.22: BasicInputFile

key	description	default	field_type	json_type
log_level	set the logging level of the module	ERROR	<a href="#">LogLevel</a>	unicode
input_file	a simple file	(REQUIRED)	<a href="#">InputFile</a>	unicode

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

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

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

This schema is designed to be a schema\_type for an ArgSchemaParser object

Table 3.23: BasicOutputDir

key	description	default	field_type	json_type
log_level	set the logging level of the module	ERROR	<a href="#">LogLevel</a>	unicode
output_dir	basic output dir	(REQUIRED)	<a href="#">OutputDir</a>	unicode

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

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

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

This schema is designed to be a schema\_type for an ArgSchemaParser object

Table 3.24: BasicOutputFile

key	description	default	field_type	json_type
output_file	a simple output file	(REQUIRED)	<a href="#">OutputFile</a>	unicode
log_level	set the logging level of the module	ERROR	<a href="#">LogLevel</a>	unicode

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

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

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

This schema is designed to be a schema\_type for an ArgSchemaParser object

Table 3.25: ModeOutputDirSchema

key	description	default	field_type	json_type
log_level	set the logging level of the module	ERROR	<a href="#">LogLevel</a>	unicode
output_dir	775 output directory	(REQUIRED)	<a href="#">OutputDir</a>	unicode

**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()
fields.test_files.test_output_dir_basic()
fields.test_files.test_output_file_relative()
fields.test_files.test_output_path(tmpdir)
fields.test_files.test_output_path_cannot_write()
fields.test_files.test_output_path_noopath()
fields.test_files.test_outputfile_no_write(tmpdir)
fields.test_files.test_outputfile_not_a_path()
fields.test_files.test_relative_file_input()
fields.test_files.test_relative_file_input_failed()
```

### 3.2.4 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(extra=None, only=(), exclude=(), pre-
                                             fix=u", strict=None, many=False, con-
                                             text=None, load_only=(), dump_only=(),
                                             partial=False)
```

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

This schema is designed to be a schema\_type for an ArgSchemaParser object

Table 3.26: NumpyFileuint16

key	description	default	field_type	json_type
a	list of lists representing a uint16 numpy array	(REQUIRED)	<a href="#">NumpyArray</a>	?
log_level	set the logging level of the module	ERROR	<a href="#">LogLevel</a>	unicode

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

Bases: *argschema.schemas.ArgSchema*

This schema is designed to be a schema\_type for an ArgSchemaParser object

Table 3.27: SliceSchema

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

```
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`, 31
- `argschema.argschema_parser`, 25
- `argschema.autodoc`, 31
- `argschema.deprecated`, 27
- `argschema.fields`, 15
  - `argschema.fields.files`, 13
  - `argschema.fields.loglevel`, 14
  - `argschema.fields.numpyarrays`, 14
  - `argschema.fields.slice`, 15
- `argschema.schemas`, 28
- `argschema.utils`, 28
- `argschema.validate`, 31

### f

- `fields`, 43
  - `fields.test_deprecated`, 40
  - `fields.test_files`, 40
  - `fields.test_loglevel`, 42
  - `fields.test_numpyarray`, 42
  - `fields.test_slice`, 43

### t

- `test_argschema_parser`, 37
- `test_first_test`, 33
- `test_output`, 37
- `test_utils`, 39



## A

args\_to\_dict() (in module argschema.utils), 28  
 ArgSchema (class in argschema.schemas), 28  
 argschema (module), 31  
 argschema.argschema\_parser (module), 25  
 argschema.autodoc (module), 31  
 argschema.deprecated (module), 27  
 argschema.fields (module), 15  
 argschema.fields.files (module), 13  
 argschema.fields.loglevel (module), 14  
 argschema.fields.numpyarrays (module), 14  
 argschema.fields.slice (module), 15  
 argschema.schemas (module), 28  
 argschema.utils (module), 28  
 argschema.validate (module), 31  
 ArgSchemaParser (class in argschema.argschema\_parser), 25  
 ArgSchemaYamlParser (class in argschema.argschema\_parser), 26

## B

bad\_test\_recursive\_schema() (in module test\_first\_test), 36  
 BadExampleRecursiveSchema (class in test\_first\_test), 33  
 BadRecursiveSchema (class in test\_first\_test), 33  
 BaseballSituation (class in test\_utils), 39  
 BasicInputDir (class in fields.test\_files), 40  
 BasicInputFile (class in fields.test\_files), 40  
 BasicOutputDir (class in fields.test\_files), 41  
 BasicOutputFile (class in fields.test\_files), 41  
 Bool (in module argschema.fields), 23  
 Boolean (class in argschema.fields), 20  
 build\_schema\_arguments() (in module argschema.utils), 29

## C

cli\_error\_dict() (in module argschema.utils), 29  
 Constant (class in argschema.fields), 23

contains\_non\_default\_schemas() (in module argschema.argschema\_parser), 27  
 context (argschema.fields.Field attribute), 16

## D

Date (class in argschema.fields), 21  
 DATEFORMAT\_DESERIALIZATION\_FUNCS (argschema.fields.DateTime attribute), 21  
 DATEFORMAT\_SERIALIZATION\_FUNCS (argschema.fields.DateTime attribute), 21  
 DateTime (class in argschema.fields), 20  
 DAYS (argschema.fields.TimeDelta attribute), 21  
 Decimal (class in argschema.fields), 19  
 default\_configurable\_sinks (argschema.argschema\_parser.ArgSchemaParser attribute), 25  
 default\_configurable\_sinks (argschema.argschema\_parser.ArgSchemaYamlParser attribute), 27  
 default\_configurable\_sources (argschema.argschema\_parser.ArgSchemaParser attribute), 25  
 default\_configurable\_sources (argschema.argschema\_parser.ArgSchemaYamlParser attribute), 27  
 default\_error\_messages (argschema.fields.Boolean attribute), 20  
 default\_error\_messages (argschema.fields.Date attribute), 21  
 default\_error\_messages (argschema.fields.DateTime attribute), 21  
 default\_error\_messages (argschema.fields.Decimal attribute), 19  
 default\_error\_messages (argschema.fields.Dict attribute), 17  
 default\_error\_messages (argschema.fields.Email attribute), 22  
 default\_error\_messages (argschema.fields.Field attribute), 16

default\_error\_messages (argschema.fields.FormattedString attribute), 20

default\_error\_messages (argschema.fields.Integer attribute), 19

default\_error\_messages (argschema.fields.List attribute), 18

default\_error\_messages (argschema.fields.Nested attribute), 17

default\_error\_messages (argschema.fields.Number attribute), 18

default\_error\_messages (argschema.fields.String attribute), 18

default\_error\_messages (argschema.fields.Time attribute), 21

default\_error\_messages (argschema.fields.TimeDelta attribute), 22

default\_error\_messages (argschema.fields.Url attribute), 22

default\_error\_messages (argschema.fields.UUID attribute), 18

DEFAULT\_FORMAT (argschema.fields.DateTime attribute), 21

default\_output\_schema (argschema.argschema\_parser.ArgSchemaParser attribute), 25

default\_schema (argschema.argschema\_parser.ArgSchemaParser attribute), 25

default\_schema (test\_argschema\_parser.MyParser attribute), 38

DefaultSchema (class in argschema.schemas), 28

deserialize() (argschema.fields.Field method), 16

Dict (class in argschema.fields), 17

## E

Email (class in argschema.fields), 22

ExampleRecursiveSchema (class in test\_first\_test), 34

## F

fail() (argschema.fields.Field method), 16

falsy (argschema.fields.Boolean attribute), 20

Field (class in argschema.fields), 15

fields (module), 43

fields.test\_deprecated (module), 40

fields.test\_files (module), 40

fields.test\_loglevel (module), 42

fields.test\_numpyarray (module), 42

fields.test\_slice (module), 43

fill\_defaults() (in module argschema.argschema\_parser), 27

Float (class in argschema.fields), 20

FormattedString (class in argschema.fields), 20

Function (class in argschema.fields), 22

## G

get\_description\_from\_field() (in module argschema.utils), 29

get\_field\_def\_from\_schema() (in module argschema.utils), 29

get\_output\_json() (argschema.argschema\_parser.ArgSchemaParser method), 25

get\_type\_from\_field() (in module argschema.utils), 30

get\_value() (argschema.fields.Field method), 16

get\_value() (argschema.fields.List method), 18

## I

initialize\_logger() (argschema.argschema\_parser.ArgSchemaParser static method), 26

InputDir (class in argschema.fields), 23

InputDir (class in argschema.fields.files), 13

InputFile (class in argschema.fields), 23

InputFile (class in argschema.fields.files), 13

Int (in module argschema.fields), 23

Integer (class in argschema.fields), 19

is\_recursive\_schema() (in module argschema.argschema\_parser), 27

## J

JsonModule (class in argschema.deprecated), 27

## L

List (class in argschema.fields), 18

load\_schema\_with\_defaults() (argschema.argschema\_parser.ArgSchemaParser method), 26

LocalDateTime (class in argschema.fields), 21

localtime (argschema.fields.DateTime attribute), 21

localtime (argschema.fields.LocalDateTime attribute), 21

LogLevel (class in argschema.fields), 24

LogLevel (class in argschema.fields.loglevel), 14

## M

main() (in module argschema), 31

make\_object() (argschema.schemas.DefaultSchema method), 28

merge\_value() (in module argschema.utils), 30

Method (class in argschema.fields), 22

MICROSECONDS (argschema.fields.TimeDelta attribute), 22

ModelFit (class in test\_first\_test), 34

ModeOutputDirSchema (class in fields.test\_files), 41

ModuleParameters (class in argschema.deprecated), 27

MyExtension (class in test\_first\_test), 34

MyExtensionOld (class in test\_first\_test), 34

MyNestedSchema (class in test\_argschema\_parser), 37

MyNestedSchemaWithDefaults (class in test\_argschema\_parser), 37

MyOutputSchema (class in test\_output), 37

MyParser (class in test\_argschema\_parser), 38

MySchema (class in test\_argschema\_parser), 38  
 MySchema2 (class in test\_argschema\_parser), 38  
 MyShorterExtension (class in test\_first\_test), 35

## N

Nested (class in argschema.fields), 16  
 num\_type (argschema.fields.Decimal attribute), 19  
 num\_type (argschema.fields.Float attribute), 20  
 num\_type (argschema.fields.Integer attribute), 19  
 num\_type (argschema.fields.Number attribute), 18  
 Number (class in argschema.fields), 18  
 NumpyArray (class in argschema.fields), 24  
 NumpyArray (class in argschema.fields.numpyarrays), 14  
 NumpyFileuint16 (class in fields.test\_numpyarray), 42

## O

OptionList (class in argschema.fields), 24  
 options (argschema.fields.LogLevel attribute), 24  
 options (argschema.fields.loglevel.LogLevel attribute), 14  
 OptionSchema (class in fields.test\_deprecated), 40  
 opts (argschema.deprecated.ModuleParameters attribute), 28  
 opts (argschema.schemas.ArgSchema attribute), 28  
 opts (argschema.schemas.DefaultSchema attribute), 28  
 opts (fields.test\_deprecated.OptionSchema attribute), 40  
 opts (fields.test\_files.BasicInputDir attribute), 40  
 opts (fields.test\_files.BasicInputFile attribute), 41  
 opts (fields.test\_files.BasicOutputDir attribute), 41  
 opts (fields.test\_files.BasicOutputFile attribute), 41  
 opts (fields.test\_files.ModeOutputDirSchema attribute), 41  
 opts (fields.test\_numpyarray.NumpyFileuint16 attribute), 42  
 opts (fields.test\_slice.SliceSchema attribute), 43  
 opts (test\_argschema\_parser.MyNestedSchema attribute), 37  
 opts (test\_argschema\_parser.MyNestedSchemaWithDefaults attribute), 38  
 opts (test\_argschema\_parser.MySchema attribute), 38  
 opts (test\_argschema\_parser.MySchema2 attribute), 38  
 opts (test\_first\_test.BadExampleRecursiveSchema attribute), 33  
 opts (test\_first\_test.BadRecursiveSchema attribute), 33  
 opts (test\_first\_test.ExampleRecursiveSchema attribute), 34  
 opts (test\_first\_test.ModelFit attribute), 34  
 opts (test\_first\_test.MyExtension attribute), 34  
 opts (test\_first\_test.MyExtensionOld attribute), 35  
 opts (test\_first\_test.MyShorterExtension attribute), 35  
 opts (test\_first\_test.PopulationSelectionParameters attribute), 35  
 opts (test\_first\_test.PopulationSelectionPaths attribute), 35  
 opts (test\_first\_test.RecursiveSchema attribute), 36

opts (test\_first\_test.SimpleExtension attribute), 36  
 opts (test\_first\_test.SimpleExtensionOld attribute), 36  
 opts (test\_output.MyOutputSchema attribute), 37  
 opts (test\_utils.BaseballSituation attribute), 39  
 opts (test\_utils.Player attribute), 39  
 output() (argschema.argschema\_parser.ArgSchemaParser method), 26  
 OutputDir (class in argschema.fields), 24  
 OutputDir (class in argschema.fields.files), 14  
 OutputFile (class in argschema.fields), 23  
 OutputFile (class in argschema.fields.files), 14

## P

Player (class in test\_utils), 39  
 PopulationSelectionParameters (class in test\_first\_test), 35  
 PopulationSelectionPaths (class in test\_first\_test), 35  
 process\_schemas() (in module argschema.autodoc), 31  
 prune\_dict\_with\_none() (in module argschema.utils), 30

## R

Raw (class in argschema.fields), 16  
 RecursiveSchema (class in test\_first\_test), 35  
 root (argschema.fields.Field attribute), 16

## S

schema (argschema.fields.Nested attribute), 17  
 schema\_argparser() (in module argschema.utils), 30  
 SECONDS (argschema.fields.TimeDelta attribute), 22  
 serialize() (argschema.fields.Field method), 16  
 serialize() (argschema.fields.Number method), 19  
 Shape (class in argschema.validate), 31  
 simple\_extension\_file() (in module test\_first\_test), 36  
 SimpleExtension (class in test\_first\_test), 36  
 SimpleExtensionOld (class in test\_first\_test), 36  
 Slice (class in argschema.fields), 24  
 Slice (class in argschema.fields.slice), 15  
 SliceSchema (class in fields.test\_slice), 43  
 smart\_merge() (in module argschema.utils), 30  
 Str (in module argschema.fields), 23  
 String (class in argschema.fields), 18

## T

test\_access\_inputfile\_failed() (in module fields.test\_files), 41  
 test\_alt\_output() (in module test\_output), 37  
 test\_argschema\_parser (module), 37  
 test\_bad\_data() (in module fields.test\_numpyarray), 42  
 test\_bad\_input\_json\_argparse() (in module test\_first\_test), 36  
 test\_bad\_inputdir() (in module fields.test\_files), 41  
 test\_bad\_option() (in module fields.test\_deprecated), 40  
 test\_bad\_option() (in module fields.test\_loglevel), 42

test\_bad\_output() (in module test\_output), 37  
test\_bad\_path() (in module test\_first\_test), 36  
test\_bad\_shape() (in module fields.test\_numpyarray), 42  
test\_bad\_slice() (in module fields.test\_slice), 43  
test\_basic\_inputdir() (in module fields.test\_files), 41  
test\_boolean\_command\_line() (in module test\_argschema\_parser), 38  
test\_david\_example() (in module test\_first\_test), 36  
test\_enoent\_outputfile\_failed() (in module fields.test\_files), 41  
test\_failed\_mode() (in module fields.test\_files), 42  
test\_first\_test (module), 33  
test\_inputdir\_no\_access() (in module fields.test\_files), 42  
test\_log\_catch() (in module test\_first\_test), 36  
test\_merge\_value\_add() (in module test\_utils), 39  
test\_merge\_value\_fail() (in module test\_utils), 39  
test\_merge\_value\_subtract() (in module test\_utils), 39  
test\_mode\_output\_osdir() (in module fields.test\_files), 42  
test\_my\_default\_nested\_parser() (in module test\_argschema\_parser), 38  
test\_my\_parser() (in module test\_argschema\_parser), 38  
test\_numpy() (in module fields.test\_numpyarray), 42  
test\_option\_list() (in module fields.test\_deprecated), 40  
test\_option\_list() (in module fields.test\_loglevel), 42  
test\_output (module), 37  
test\_output() (in module test\_output), 37  
test\_output\_dir\_bad\_location() (in module fields.test\_files), 42  
test\_output\_dir\_bad\_permission() (in module fields.test\_files), 42  
test\_output\_dir\_basic() (in module fields.test\_files), 42  
test\_output\_file\_relative() (in module fields.test\_files), 42  
test\_output\_path() (in module fields.test\_files), 42  
test\_output\_path\_cannot\_write() (in module fields.test\_files), 42  
test\_output\_path\_noapath() (in module fields.test\_files), 42  
test\_output\_unvalidated() (in module test\_output), 37  
test\_outputfile\_no\_write() (in module fields.test\_files), 42  
test\_outputfile\_not\_a\_path() (in module fields.test\_files), 42  
test\_parser\_output() (in module test\_argschema\_parser), 39  
test\_recursive\_schema() (in module test\_first\_test), 36  
test\_relative\_file\_input() (in module fields.test\_files), 42  
test\_relative\_file\_input\_failed() (in module fields.test\_files), 42  
test\_schema\_argparser\_with\_baseball() (in module test\_utils), 39  
test\_serialize() (in module fields.test\_numpyarray), 42  
test\_simple\_description() (in module test\_first\_test), 36  
test\_simple\_example() (in module test\_first\_test), 36  
test\_simple\_extension\_fail() (in module test\_first\_test), 36  
test\_simple\_extension\_old\_pass() (in module test\_first\_test), 36  
test\_simple\_extension\_pass() (in module test\_first\_test), 36  
test\_simple\_extension\_required() (in module test\_first\_test), 36  
test\_simple\_extension\_write\_debug\_level() (in module test\_first\_test), 36  
test\_simple\_extension\_write\_overwrite() (in module test\_first\_test), 37  
test\_simple\_extension\_write\_overwrite\_list() (in module test\_first\_test), 37  
test\_simple\_extension\_write\_pass() (in module test\_first\_test), 37  
test\_slice() (in module fields.test\_slice), 43  
test\_smart\_merge() (in module test\_utils), 39  
test\_smart\_merge\_add() (in module test\_utils), 39  
test\_smart\_merge\_nested() (in module test\_utils), 39  
test\_smart\_merge\_none() (in module test\_utils), 39  
test\_smart\_merge\_not\_none() (in module test\_utils), 39  
test\_smart\_merge\_same() (in module test\_utils), 40  
test\_tmp\_output\_cleanput() (in module test\_output), 37  
test\_utils (module), 39  
Time (class in argschema.fields), 21  
TimeDelta (class in argschema.fields), 21  
truthy (argschema.fields.Boolean attribute), 20

## U

Url (class in argschema.fields), 22  
URL (in module argschema.fields), 22  
UUID (class in argschema.fields), 18

## V

validate\_outpath() (in module argschema.fields.files), 14