MiniScript

Dmitry Tantsur

Jan 02, 2021

CONTENTS

MiniScript is an embedded scripting language with the syntax heavily inspired by Ansible, but targeted at data processing rather than remote execution. MiniScript aims to keep the familiar look-and-feel while being trivial to embed and to extend.

Compared to real Ansible, MiniScript does NOT have:

- Roles, playbooks or any other form of reusability.
- "Batteries included" set of actions and filters.
- Any local or remote execution facility.
- Notifications, parallel execution or other advanced features.

MiniScript does offer:

- Loops, variables, conditions and blocks.
- Jinja2 templating integration.
- Lean and easily extensible feature set.
- A few filters most useful for data processing.
- An ability to return a value from a script.
- Ansible-compatible backslash handling.
- 100% unit test coverage.
- A permissive license (BSD).

Note: MiniScript does not use Ansible directly, nor does it import any Ansible code. We are also not aiming for perfect compatibility and do diverge in some aspects.

- Documentation: https://miniscript.readthedocs.io
- Source: https://github.com/dtantsur/miniscript
- Author: Dmitry Tantsur
- License: BSD (3-clause)

Contents

CHAPTER

ONE

RUNNING SCRIPTS

class miniscript.**Engine**(*tasks: Optional[Dict[str, Type[miniscript._task.Task]]] = None*, *logger: Optional[logging.Logger] = None*, *additional_filters: bool = True*)

Engine that runs scripts.

Parameters

- **tasks** – Tasks to use for this engine, see *Engine.tasks*.

  Changed in version 1.1: Tasks are now optional, only built-in tasks are used by default.

- **logger** – Logger to use for all logging. If *None*, a default one is created.
- **additional_filters** – If *True*, additional Ansible-compatible filters from *miniscript.filters* are enabled.

  New in version 1.1.

Raises ValueError on tasks conflicting with built-in parameters, see *Task*.

The development flow is:

1. define your tasks by subclassing *Task*;
2. create an *Engine* with the task definitions;
3. (optionally) create a custom *Context*;
4. run *Engine.execute()*.

Preparing an engine:

```
import miniscript

class AddTask(miniscript.Task):
    '''A task implementing addition.'''

    required_params = {'values': list}
    '''One required parameter - a list of values.'''
    singleton_param = 'values'
    '''Can be supplied without an explicit "values".'''

    def validate(self, params, context):
        '''Validate the parameters.'''
        super().validate(params, context)
        for item in params['values']:
            int(item)

    def execute(self, params, context):
        '''Execute the action, return a "sum" value.'''
```

(continues on next page)

(continued from previous page)

```
        return {"sum": sum(params['values'])}

engine = miniscript.Engine({'add': AddTask})
```

Some tasks are built into the engine:

- `block` - *tasks.Block*
- `fail` - *tasks.Fail*
- `log` - *tasks.Log*
- `return` - *tasks.Return*
- `vars` - *tasks.Vars*

An example script:

```
---
- name: only accept positive integers
  fail: "{{ item }} must be positive"
  when: item <= 0
  loop: "{{ values }}"

- name: add the provided values
  add: "{{ values }}"
  register: result

- name: log the result
  log:
    info: "The sum is {{ result.sum }}"

- name: return the result
  return: "{{ result.sum }}"
```

Executing a script (obviously, it does not have to come from YAML):

```
import yaml

with open("script.yaml") as fp:
    code = yaml.safe_load(fp)

# The context holds all variables.
context = miniscript.Context(engine, values=[23423, 43874, 22834])

# Unlike Ansible, MiniScript can return a result!
result = engine.execute(code, context)  # result == 90131
```

execute(*source: Union[List[Dict[str, Any]], Dict[str, Any]]*, *context: Optional[miniscript._context.Context] = None*) → Any

Execute a script.

Parameters

- **source** – Script source code in JSON format. An implicit *Script* object is created from it.
- **context** – A *Context* object to hold execution context.

Returns

The outcome of the script or *None*.

Note: *ExecutionFailed* is raised if a script returns an undefined value.

Raises *ExecutionFailed* on a runtime error.

Raises *InvalidScript* if the script is invalid.

Raises *InvalidTask* if a task is invalid.

environment: miniscript._context.Environment
An *Environment* object used for templating.

logger: logging.Logger
Python logger used for logging.

tasks: Dict[str, Type[miniscript._task.Task]]
Mapping of task names to their implementation classes.

The name will be used in a script. The implementation must be a *Task* subclass (not an instance).

Includes built-in tasks.

CHAPTER

TWO

CREATING TASKS

class miniscript.**Task**(*name: str*, *definition: Dict[str, Any]*, *engine: _engine.Engine*)
An abstract base class for a task.

An implementation must override *Task.execute()* and may also override *Task.validate()*, although it is usually not necessary.

Parameters

- **name** – Name of this task as used in the script.
- **definition** – The task definition from the script.
- **engine** – An *Engine* the task is executed on.

__call__(*context: miniscript._context.Context*) → None
Check conditions and execute the task in the context.

It is not recommended to override this method, see *Task.execute()* instead.

Parameters **context** – A *Context* object to hold execution context.

abstract execute(*params: miniscript._context.Namespace*, *context: miniscript._context.Context*) → Optional[Mapping[str, Any]]
Execute the task.

Override this method to provide the task logic.

Parameters

- **params** – Validated parameters as a mapping that automatically evaluates templates.
- **context** – A *Context* object to hold execution context. It is a mutable mapping that holds variables.

Returns Values stored in a *Result* if `Task.register` is set (otherwise discarded). A mapping from names to values.

validate(*params: miniscript._context.Namespace*, *context: miniscript._context.Context*) → None
Validate the passed parameters.

The call may modify the parameters in-place, e.g. to apply type conversion. The default implementation relies on class-level *Task.required_params*, *Task.optional_params*, *Task.singleton_param*, *Task.free_form* and *Task.allow_empty* for parameter validation.

Parameters

- **params** – The current parameters as a mapping that automatically evaluates templates on access.
- **context** – A *Context* object to hold execution context.

allow_empty: bool = True
If no parameters are required, whether to allow empty input.

Makes no sense if *Task.required_params* is not empty.

engine: _engine.Engine
The *Engine* this task uses.

free_form: bool = False
Whether this task accepts any arguments.

Validation for known arguments is still run, and required parameters are still required.

ignore_errors: bool = False
Whether to ignore errors and continue execution.

Often used together with *Task.register* as

```
- name: run a task that can fail
  task_that_can_fail:
  ignore_errors: True
  register: fallible_result

- name: log if the task failed
  log:
    warning: "Task failed: {{ fallible_result.failure }}"
  when: fallible_result.failed
```

loop: Optional[Union[str, list]] = None
Value to loop over.

For each item in the resulting list, execute the task passing the item as the `item` value in the context.

```
- name: do excessive logging
  log:
    info: "I like number {{ item }}"
  loop: [1, 2, 3, 4, 5]
```

Conditions are evaluated separately for each item:

```
- name: do excessive logging
  log:
    info: "I like even numbers like {{ item }}"
  loop: [1, 2, 3, 4, 5]
  when: item % 2 == 0
```

The loop value itself may be a template yielding a list.

name: str
The description of this task in the script.

If a human-readable description is not provided, uses the task name.

optional_params: Dict[str, Optional[Type]] = {}
A mapping with optional parameters.

See *Task.required_params* for a list of supported types.

params: Mapping[str, Any]
Task parameter after passing preliminary validation.

Evaluating templated variables is not possible until execution, so this field may contain raw templates.

register: Optional[str] = None
Variable to store the result of this task as a *Result*.

required_params: Dict[str, Optional[Type]] = {}
A mapping with required parameters.

A value is either *None* or one of the supported types: *str*, *int*, *float*, *list*.

singleton_param: Optional[str] = None
A name for the parameter to store if the input is not an object.

For example (see *tasks.Fail*),

```
- fail: I have failed
```

is converted to

```
- fail:
    msg: I have failed
```

when: Optional[Callable[[miniscript._context.Context], bool]] = None
A condition of this task.

Specified via the `when` statement and supports templating, e.g.:

```
- fail: Address must be defined
  when: address is undefined
```

class miniscript.**Context**(*engine*, **args*, ***kwargs*)
A context of an execution.

copy() → miniscript._context.Context
Make a shallow copy of the context.

CHAPTER

THREE

BUILT-IN TASKS

Built-in task definitions.

class miniscript.tasks.**Assert**(*name: str*, *definition: Dict[str, Any]*, *engine: _engine.Engine*)
An assertion.

Fails if at least one of the provided statements is false:

```
- vars:
    some_value: 42

- assert:
    - some_value is defined
    - some_value == 42
```

New in version 1.1.

optional_params: Dict[str, Optional[Type]] = {'fail_msg': None}
Can accept an optional failure message.

required_params: Dict[str, Optional[Type]] = {'that': None}
Requires one or more statements as a list or a string.

singleton_param: Optional[str] = 'that'
The statement list can be provided directly to `assert`.

class miniscript.tasks.**Block**(*name: str*, *definition: Dict[str, Any]*, *engine: _engine.Engine*)
Grouping of tasks.

Blocks can be used to share top-level parameters, e.g. a condition:

```
- block:
    - task1:
    - task2:
    - task3:
  when: enable_all_three_tasks
```

required_params: Dict[str, Optional[Type]] = {'tasks': <class 'list'>}
Requires a task list.

singleton_param: Optional[str] = 'tasks'
The task list can be provided directly to `block`.

class miniscript.tasks.**Fail**(*name: str*, *definition: Dict[str, Any]*, *engine: _engine.Engine*)
Fail the execution.

Often used in combination with some condition.

```
- name: fail if the path is not defined
  fail: path must be defined
  when: path is undefined
```

required_params: Dict[str, Optional[Type]] = {'msg': <class 'str'>}
Requires a string message.

singleton_param: Optional[str] = 'msg'
The message can be provided directly to `fail`.

class miniscript.tasks.**Log**(*name: str*, *definition: Dict[str, Any]*, *engine: _engine.Engine*)
Log something.

Uses standard Python logging and understands 4 levels: `debug`, `info`, `warning` and `error`.

```
- log:
    info: "checking of something bad has happened..."
- log:
    error: "oh no, something bad has happened!"
  when: something_bad is happened
```

optional_params: Dict[str, Optional[Type]] = {'debug': <class 'str'>, 'error': <cla
Requires at least one of the levels and its message.

class miniscript.tasks.**Return**(*name: str*, *definition: Dict[str, Any]*, *engine: _engine.Engine*)
Return a value to the caller.

This is a unique feature of MiniScript not present in Ansible. The value will be returned from `Engine.execute()`.

```
- name: return the answer
  return: 42
```

optional_params: Dict[str, Optional[Type]] = {'result': None}
Optionally accepts a result (otherwise the result is `None`).

singleton_param: Optional[str] = 'result'
The result can be provided directly to `return`.

class miniscript.tasks.**Vars**(*name: str*, *definition: Dict[str, Any]*, *engine: _engine.Engine*)
Set variables.

Similar to `set_fact` in Ansible, but we don't have facts. The variables are stored in the context.

```
- vars:
    num1: 2
    num2: 2
- log:
    info: "Look ma, I can multiply: {{ num1 * num2 }}"
```

free_form: bool = True
Accepts any parameters.

CHAPTER

FOUR

BUILT-IN FILTERS

Reimplementations of common Ansible filters.

New in version 1.1.

Note: Alternatively, can also use the jinja2-ansible-filters project but it will likely require licensing your code under GPL (the license Ansible uses).

`miniscript.filters.`**`bool_`**(*value: Any*) → bool

Convert a value to a boolean according to Ansible rules.

The corresponding filter is called `bool` (without an underscore). True values are `True`, strings "Yes", "True" and "1", number 1; everything else is False.

```
- vars:
    is_true: "{{ 'YES' | bool }}"
```

New in version 1.1.

`miniscript.filters.`**`combine`**(*value: Union[Sequence[Mapping], Mapping]*, **other: Mapping*, *recursive: bool = False*, *list_merge: str = 'replace'*) → Dict

Combine several dictionaries into one.

A typical pattern of adding a value to a dictionary:

```
- vars:
    new_dict: "{{ old_dict | combine({'new key': 'new value'}) }}"
```

When a list is provided as input, all items from it are combined.

New in version 1.1.

Parameters

- **`recursive`** – Whether to merge dictionaries recursively.
- **`list_merge`** – How to merge lists, one of `replace`, `keep`, `append`, `prepend`, `append_rp`, `prepend_rp`. The `_rp` variants remove items that are present in both lists from the left-hand list.

`miniscript.filters.`**`dict2items`**(*value: Mapping*, *key_name: str = 'key'*, *value_name: str = 'value'*) → List[Dict[str, Any]]

Convert a mapping to a list of its items.

For example, converts

```
milk: 1
eggs: 10
bread: 2
```

into

```
- key: milk
  value: 1
- key: eggs
  value: 10
- key: bread
  value: 2
```

New in version 1.1.

Parameters

- **value** – Any mapping.
- **key_name** – Key name for input keys.
- **value_name** – Key name for input values.

Returns A list of dicts.

miniscript.filters.**difference**(*value: list*, *other: list*) → list

Difference of two lists.

```
- vars:
    new_list: "{{ [2, 4, 6, 8, 12] | difference([3, 6, 9, 12, 15]) }}"
    # -> [2, 4, 8]
```

New in version 1.1.

miniscript.filters.**flatten**(*value: list*, *levels: Optional[int] = None*) → list

Flatten a list.

```
- vars:
    new_list: "{{ [1, 2, [3, [4, 5, [6]], 7]] | flatten }}"
    # -> [1, 2, 3, 4, 5, 6, 7]
```

To flatten only the top level, use the `levels` argument:

```
- vars:
    new_list: "{{ [1, 2, [3, [4, 5, [6]], 7]] | flatten(levels=1) }}"
    # -> [1, 2, 3, [4, 5, [6]], 7]
```

New in version 1.1.

Parameters **levels** – Number of levels to flatten. If *None* - flatten everything.

miniscript.filters.**intersect**(*value: list*, *other: list*) → list

Intersection of two lists.

```
- vars:
    new_list: "{{ [2, 4, 6, 8, 12] | intersect([3, 6, 9, 12, 15]) }}"
    # -> [6, 12]
```

New in version 1.1.

miniscript.filters.**ipaddr**(*value: Union[str, int]*, *query: Optional[str] = None*) → str
Filter IP addresses and networks.

New in version 1.1.

Implements Ansible ipaddr filter.

miniscript.filters.**ipv4**(*value: Union[str, int]*, *query: Optional[str] = None*) → str
Filter IPv4 addresses and networks.

New in version 1.1.

Implements Ansible ipv4 filter.

miniscript.filters.**ipv6**(*value: Union[str, int]*, *query: Optional[str] = None*) → str
Filter IPv6 addresses and networks.

New in version 1.1.

Implements Ansible ipv6 filter.

miniscript.filters.**items2dict**(*value: List[Mapping[str, Any]]*, *key_name: str = 'key'*, *value_name: str = 'value'*) → Dict
A reverse of *dict2items()*.

For example, converts

```
- key: milk
  value: 1
- key: eggs
  value: 10
- key: bread
  value: 2
```

into

```
milk: 1
eggs: 10
bread: 2
```

New in version 1.1.

Parameters

- **value** – A list of mappings.
- **key_name** – Key name for output keys.
- **value_name** – Key name for output values.

Returns A dictionary.

miniscript.filters.**json_query**(*value: Any*, *query: str*) → Any
Run a JSON query against the data.

Requires the jmespath library. See jmespath examples.

New in version 1.1.

miniscript.filters.**regex_escape**(*value: str*) → str
Escape special regular expression characters in a string.

New in version 1.1.

miniscript.filters.**regex_findall**(*value: str*, *pattern: str*, *, *multiline: bool = False*, *ignorecase: bool = False*) → List[str]

Find all occurencies of a pattern in a string.

For example:

```
- vars:
    url: "http://www.python.org"

- return: "{{ url | regex_findall('(?<=\\W)\\w{3}(?=\\W|$)') }}"
  # returns ['www', 'org']
```

New in version 1.1.

Parameters

- **pattern** – Python regular expression.
- **multiline** – Whether ^ matches a beginning of each line, not just beginning of the string.
- **ignorecase** – Whether to ignore case when matching.

miniscript.filters.**regex_replace**(*value: str*, *pattern: str*, *replacement: str = ''*, *, *multiline: bool = False*, *ignorecase: bool = False*, *count: int = 0*) → str

Replace all occurencies of a pattern in a string.

MiniScript implements Ansible-compatible slashes handling to avoid duplication of slashes inside Jinja expressions.

```
- vars:
    url: "http://www.python.org"

- return: "{{ url | regex_replace('(?<=\\W)\\w{3}(?=\\W|$)',
                                  '\"\\1\") }}"
  # returns 'http://"www".python."org"'
```

New in version 1.1.

Parameters

- **pattern** – Python regular expression.
- **replacement** – String to replace with, an empty string by default.
- **multiline** – Whether ^ matches a beginning of each line, not just beginning of the string.
- **ignorecase** – Whether to ignore case when matching.
- **count** – How many occurencies to replace. Zero (the default) means replace all.

miniscript.filters.**regex_search**(*value: str*, *pattern: str*, *, *multiline: bool = False*, *ignorecase: bool = False*) → str

Find an occurence of a pattern in a string.

New in version 1.1.

Parameters

- **pattern** – Python regular expression.
- **multiline** – Whether ^ matches a beginning of each line, not just beginning of the string.
- **ignorecase** – Whether to ignore case when matching.

miniscript.filters.**symmetric_difference**(*value: list*, *other: list*) → list

Symmetric difference (exclusive OR) of two lists.

```
- vars:
    new_list: "{{ [2, 4, 6, 8, 12]
                  | symmetric_difference([3, 6, 9, 12, 15]) }}"
    # -> [2, 3, 4, 8, 9, 15]
```

New in version 1.1.

miniscript.filters.**to_datetime**(*value: str*, *format: str = '%Y-%m-%d %H:%M:%S'*) → datetime.datetime

Parse a date/time according to the format.

The default format is `%Y-%m-%d %H:%M:%S`.

New in version 1.1.

miniscript.filters.**union**(*value: list*, *other: list*) → list

Union of two lists.

```
- vars:
    new_list: "{{ [2, 4, 6, 8, 12] | union([3, 6, 9, 12, 15]) }}"
    # -> [2, 3, 4, 6, 8, 9, 12, 15]
```

New in version 1.1.

miniscript.filters.**urlsplit**(*value: str*, *query: Optional[str] = None*) → Union[Dict, str]

Split a URL into components.

Known components are fragment, hostname, netloc, password, path, port, query, scheme, username.

New in version 1.1.

Parameters **query** – Requested component. If *None*, all components are returned in a dictionary.

miniscript.filters.**zip_**(*first: Sequence*, **other: Sequence*) → Iterator

Zip two sequences together.

The corresponding filter is called `zip` (without an underscore).

New in version 1.1.

miniscript.filters.**zip_longest**(*first: Sequence*, **other: Sequence*, *fillvalue: Optional[Any] = None*) → Iterator

Zip sequences together, always exhausing all of them.

New in version 1.1.

Parameters **fillvalue** – Value to fill shorter sequences with.

CHAPTER

FIVE

ERRORS

class miniscript.**Error**
Base class for all errors.

class miniscript.**InvalidScript**
The script definition is invalid.

class miniscript.**InvalidTask**
The task definition is invalid.

class miniscript.**UnknownTask**
An task is not known.

class miniscript.**ExecutionFailed**
Execution of a task failed.

CHAPTER

SIX

ADVANCED

class miniscript.**Environment**
A templating environment.

class miniscript.**Namespace**(*environment: miniscript._context.Environment*, *context: miniscript._context.Context*, **args*, ***kwargs*)
A namespace with value rendering.

Works like a dictionary, but evaluates values on access using the provided templating environment.

New in version 1.1.

Parameters

- **environment** – Templating environment to use.
- **context** – A *Context* object to hold execution context.
- **args** – Passed to `dict` unchanged.
- **kwargs** – Passed to `dict` unchanged.

copy() → miniscript._context.Namespace
Make a shallow copy of the namespace.

get_raw(*key*, *default=None*) → Any
Get a value without evaluating it.

materialize() → dict
Recursively evaluate values, returning a normal dict.

class miniscript.**Result**(*results: Mapping[str, Any]*, *failure: Optional[str] = None*, *skipped: bool = False*)
A result of a task.

Any resulting values are stored directly on the object.

failed: bool
Whether the task failed (the opposite of *Result.succeeded*).

failure: Optional[str] = None
Failure message if the task failed.

skipped: bool = False
Whether the task was skipped via a *when* statement.

succeeded: bool
Whether the task succeeded (the opposite of *Result.failed*).

class miniscript.**Script**(*engine: miniscript._engine.Engine*, *source: Union[List[Dict[str, Any]], Dict[str, Any]]*)
A script.

Parameters

- **engine** – An *Engine*.
- **source** – A source definition or a list of tasks to execute.

__call__(*context: Optional[miniscript._context.Context] = None*) → Any
Execute the script.

Parameters **context** – A *Context* object to hold execution context.

Returns The outcome of the script or *None*

Raises *ExecutionFailed* on a runtime error.

Raises *InvalidScript* if the script is invalid.

Raises *InvalidTask* if a task is invalid.

engine: miniscript._engine.Engine
The *Engine* of this script.

tasks: List[miniscript._task.Task]
A list of `Tasks` in the order of execution.

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

INDEX

Symbols

A

B

C

D

E

F

G

I

J

L

M

N

O