Optimize Your Dags:

Embrace Dag Params for Efficiency and Simplicity

Sumit Maheshwari

PMC Apache Airflow, Tech Lead at Uber

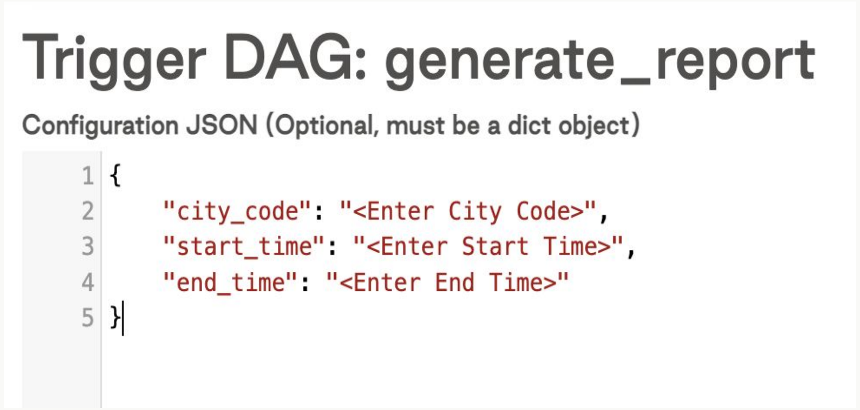
Ex - Twitter, Astronomer, Qubole

DAG Params, what?

- Part of Airflow since the beginning
- Major rework done in 2021
- Support type checking, range validations, mandate inputs, etc
- Trigger DAG UI to generate a full fledged form on the basis of params

Example DAG - Before 2.2.0

```
dag = DAG(
   dag_id='generate_report',
   start_date=datetime(2024, 1, 1),
   default_args=default_args,
   schedule_interval=None,
   params={
       "city_code": "<Enter City Code>",
       "start_time": "<Enter Start Time>",
       "end_time": "<Enter End Time>",
   }
)
```



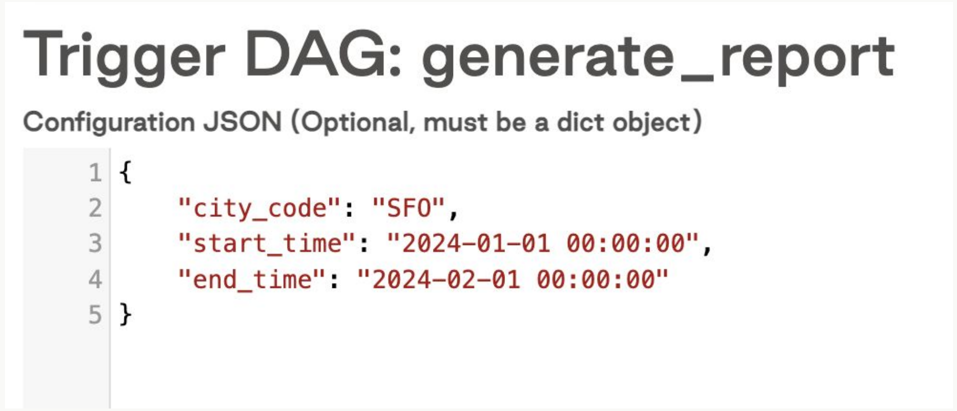
Trigger DAG: generate_report

Configuration JSON (Optional, must be a dict object)

```
{
    "city_code": "<Enter City Code>",
    "start_time": "<Enter Start Time>",
    "end_time": "<Enter End Time>"
}
```

Example DAG.. Continued

```
dag = DAG(
    dag_id='generate_report',
    start_date=datetime(2024, 1, 1),
    default_args=default_args,
    schedule_interval=None,
    params={
        "city_code": "SFO",
        "start_time": "2024-01-01 00:00:00",
        "end_time": "2024-02-01 00:00:00",
    }
)
```



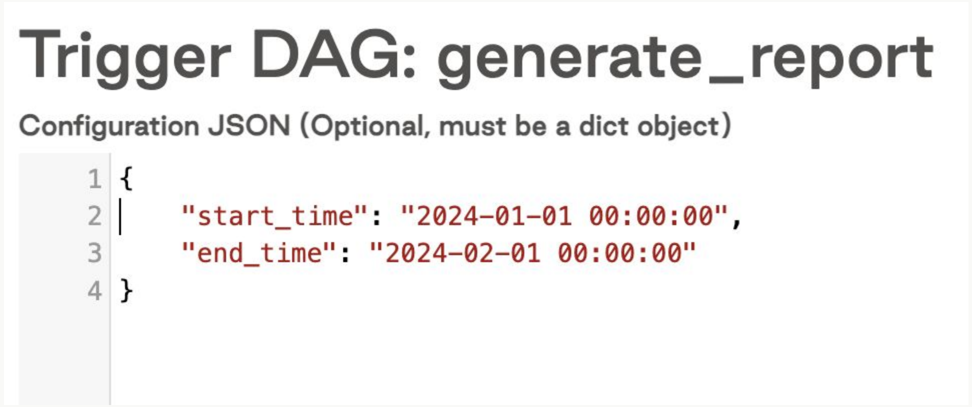
Trigger DAG: generate_report

Configuration JSON (Optional, must be a dict object)

```
{
    "city_code": "SFO",
    "start_time": "2024-01-01 00:00:00",
    "end_time": "2024-02-01 00:00:00"
}
```

Example DAG.. Continued

Can you guess, how it'll behave?



Trigger DAG: generate_report

Configuration JSON (Optional, must be a dict object)

```
{
    "start_time": "2024-01-01 00:00:00",
    "end_time": "2024-02-01 00:00:00"
}
```

Requirements

- Must
  - Ensure backward compatibility.
  - Support default values and multiple types (int, bool, str, etc.).
  - Allow validation options (min/max, length, regex).
  - Maintain consistent behavior across UI, CLI, and API.
- Good to have:
  - UI should display input controls based on param type, showing required fields and defaults.
  - For params with options, UI can display lists or live pattern matching.

Proposal

- Create a Param class for use in the params dictionary
- It should store a default value and validation rules.
- Include a method to validate and resolve the value (default or user-provided).
- Ensure easy serialization/deserialization for database use.
- It should work with both traditional and decorator-based DAG creation.

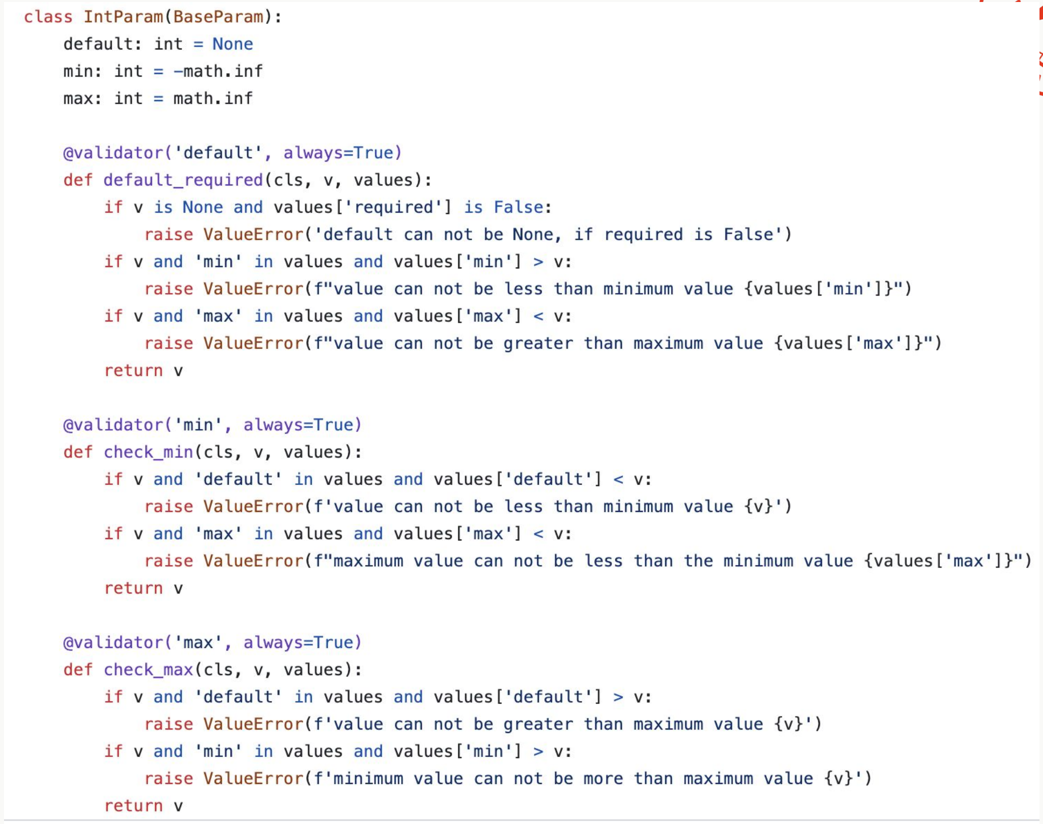
Approaches

pydantic

One of the fastest Python libraries to provide data & type validations.*

- Easy to implement
- Easy to extend

- Repeated work
- Painful modifications



```python
class IntParam(BaseParam):
    default: int = None
    min: int = -math.inf
    max: int = math.inf

    @validator('default', always=True)
    def default_required(cls, v, values):
        if v is None and values['required'] is False:
            raise ValueError('default can not be None, if required is False')
        if v and 'min' in values and values['min'] > v:
            raise ValueError(f"value can not be less than minimum value {values['min']}")
        if v and 'max' in values and values['max'] < v:
            raise ValueError(f"value can not be greater than maximum value {values['max']}")
        return v

    @validator('min', always=True)
    def check_min(cls, v, values):
        if v and 'default' in values and values['default'] < v:
            raise ValueError(f'value can not be less than minimum value {v}')
        if v and 'max' in values and values['max'] < v:
            raise ValueError(f"maximum value can not be less than the minimum value {values['max']}")
        return v

    @validator('max', always=True)
    def check_max(cls, v, values):
        if v and 'default' in values and values['default'] > v:
            raise ValueError(f'value can not be greater than maximum value {v}')
        if v and 'min' in values and values['min'] > v:
            raise ValueError(f'minimum value can not be more than maximum value {v}')
        return v
```

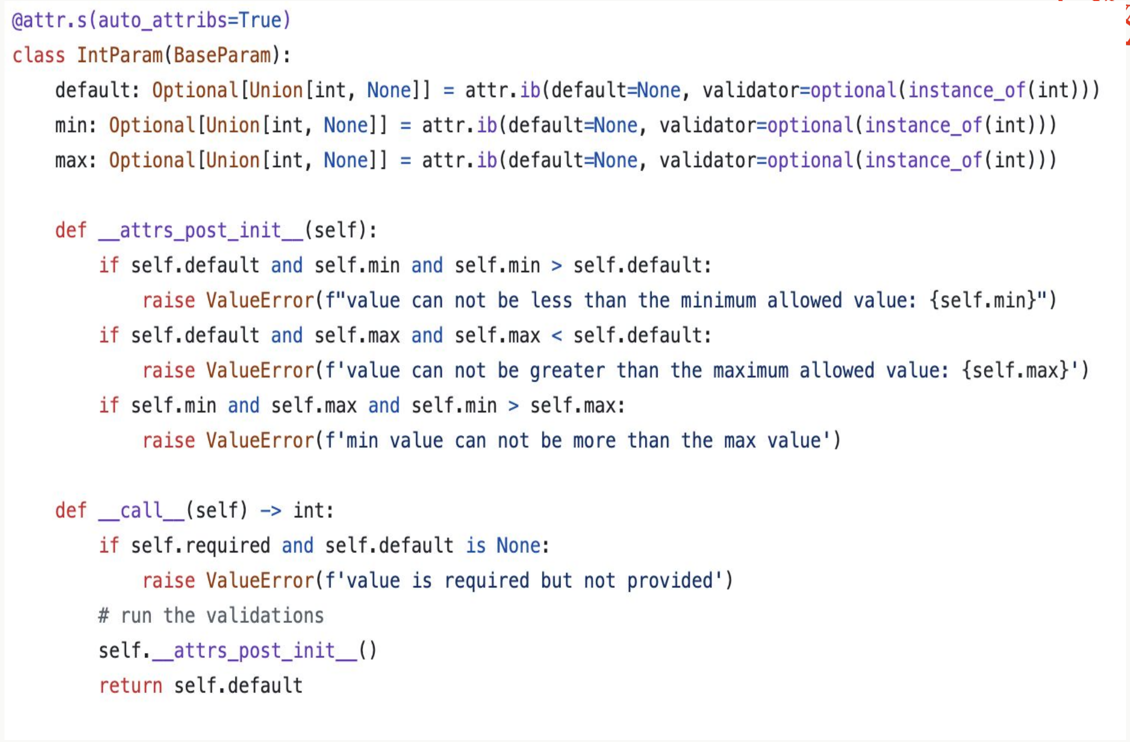
Approaches

attrs

attrs simplifies writing classes and also exposes various in-build validators & pre-post init methods.

- 🟢 Easy to implement
- 🟢 Easy to extend

- 🔴 Repeated work
- 🔴 Painful modifications



```python
@attr.s(auto_attribs=True)
class IntParam(BaseParam):
    default: Optional[Union[int, None]] = attr.ib(default=None, validator=optional(instance_of(int)))
    min: Optional[Union[int, None]] = attr.ib(default=None, validator=optional(instance_of(int)))
    max: Optional[Union[int, None]] = attr.ib(default=None, validator=optional(instance_of(int)))

    def __attrs_post_init__(self):
        if self.default and self.min and self.min > self.default:
            raise ValueError(f"value can not be less than the minimum allowed value: {self.min}")
        if self.default and self.max and self.max < self.default:
            raise ValueError(f'value can not be greater than the maximum allowed value: {self.max}')
        if self.min and self.max and self.min > self.max:
            raise ValueError(f'min value can not be more than the max value')

    def __call__(self) -> int:
        if self.required and self.default is None:
            raise ValueError(f'value is required but not provided')
        # run the validations
        self.__attrs_post_init__()
        return self.default
```

Approaches

json-schema

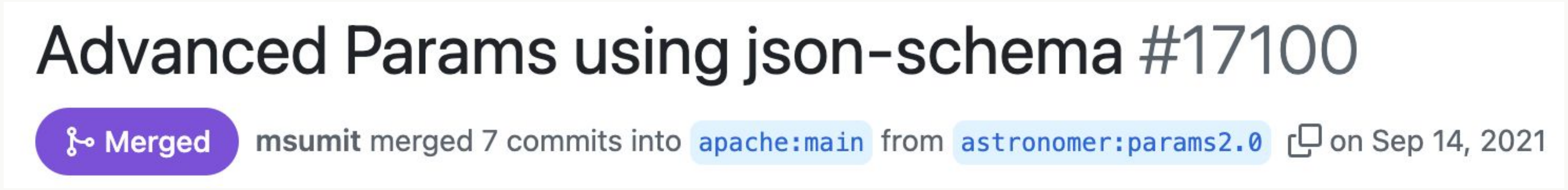
json-schema has a very powerful & extensive way to define properties (validations) on a field in a language-agnostic way. It has implementation libs in almost all major languages & provides very extensive validations.

- json-schema is being used in DAG serialization already
- Plenty of OOB rules/validations to suffice major use-cases
- Can use it's JS framework to validate data on UI

- Complex rules can overwhelm users

Airflow 2.2.0 - Welcome DAG Params

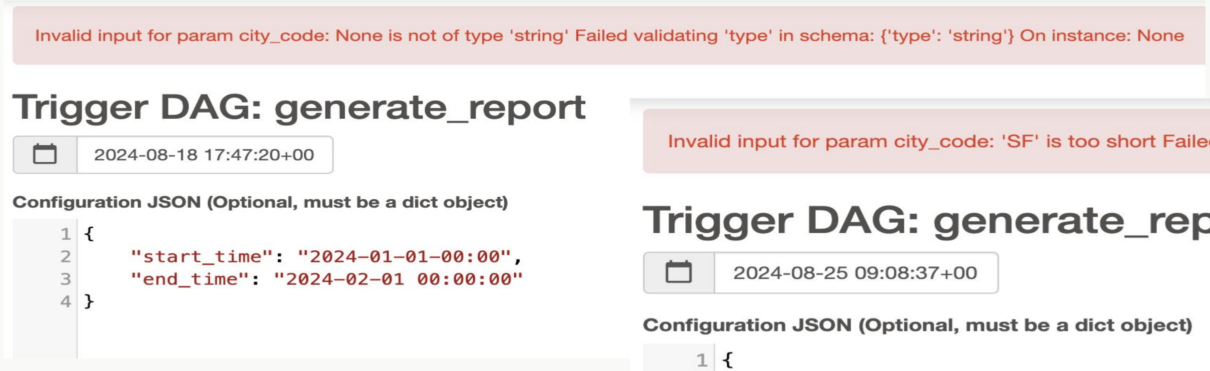
- Based on json-schema
- In-drop replacement of existing params dictionary
- Fully backward compatible
- Supports multiple types like string, int, bool, list, and many more
- Supports regex, making it useful for variety of use-cases
- Supports pre-defined validation formats like uri, date-time, email, hostname, ipv4/6, etc


Advanced Params using json-schema #17100

Merged msumit merged 7 commits into `apache:main` from `astronomer:params2.0` on Sep 14, 2021

Airflow 2.2.0 - Example Dag

```
DAG(
    dag_id='generate_report',
    start_date=datetime(2024, 1, 1),
    default_args=default_args,
    schedule_interval=None,
    params={
        "city_code": Param(type="string", minLength=3, maxLength=3),
        "start_time": Param("2024-01-01 00:00:00", type="string", format="date-time"),
        "end_time": Param("2024-02-01 00:00:00", type="string", format="date-time"),
    }
)
```



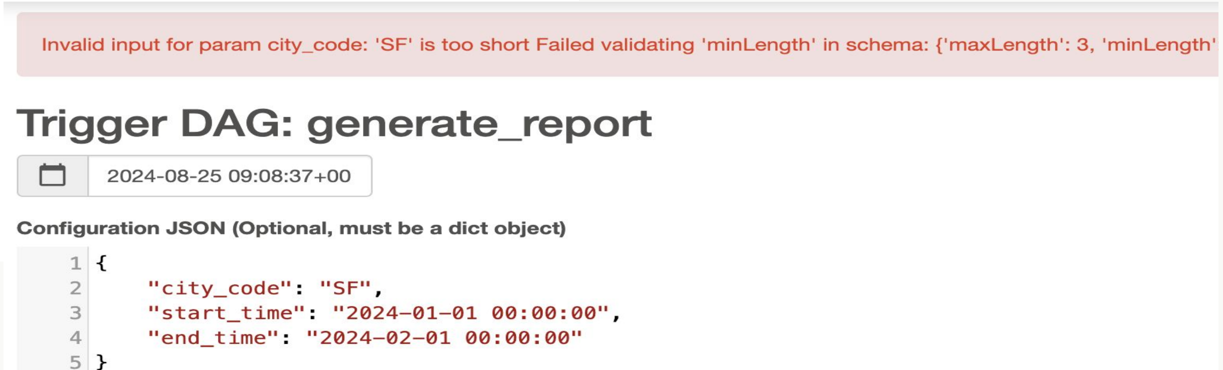
Invalid input for param city_code: None is not of type 'string' Failed validating 'type' in schema: {'type': 'string'} On instance: None

Trigger DAG: generate_report

2024-08-18 17:47:20+00

Configuration JSON (Optional, must be a dict object)

```
{
    "start_time": "2024-01-01-00:00",
    "end_time": "2024-02-01 00:00:00"
}
```



Invalid input for param city_code: 'SF' is too short Failed validating 'minLength' in schema: {'maxLength': 3, 'minLength'

Trigger DAG: generate_report

2024-08-25 09:08:37+00

Configuration JSON (Optional, must be a dict object)

```
{
    "city_code": "SF",
    "start_time": "2024-01-01 00:00:00",
    "end_time": "2024-02-01 00:00:00"
}
```

Trigger UI Revamp Journey

2.6.0
Initial version of new DAG trigger UI

2.6.3
Fix rendering empty list, decimal vs integer

2.7.0
Skip trigger button, Multi-Select, Labels on drop-downs, Non string arrays, Fix JSON propagation

2.7.2
Fix render “0” default, None values

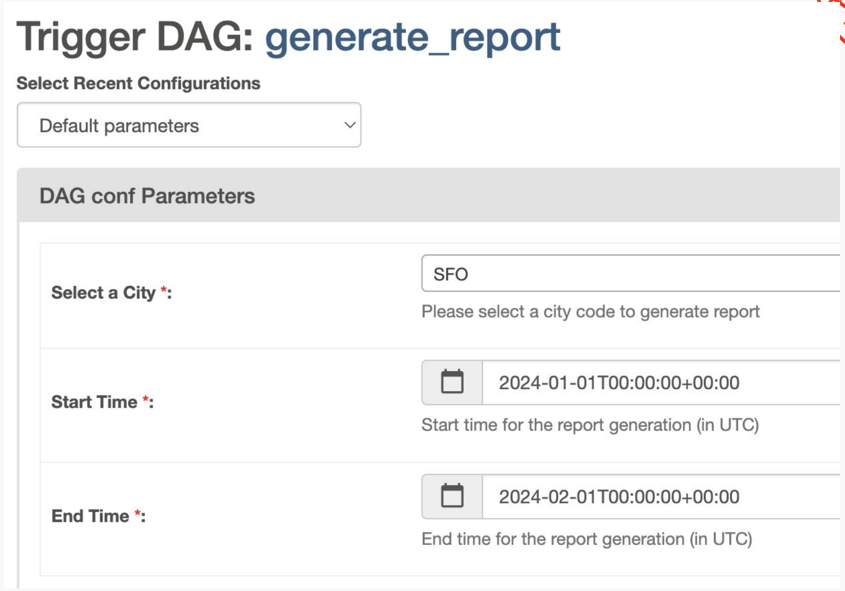
2.8.2
pre-population of trigger form values via URL parameters

Special thanks to:

- jscheffl
- techolga
- herlambang
- SamWheating
- MatthieuBlais
- bbovenzi
- ryanahamilton
- hussein-awala
- jedcunningham

Airflow 2.8+ - Example Dag

```
DAG(
   dag_id='generate_report',
   start_date=datetime(2024, 1, 1),
   default_args=default_args,
   schedule_interval=None,
   params={
       "city_code": Param(
           type="string",
           enum=["SFO", "NYC", "WDC", "CHI", "BLR", "MUM"],
           title="Select a City",
           description="Please select a city code to generate report",
       ),
       "start_time": Param("2024-01-01 00:00:00",
           type="string",
           format="date-time",
           title="Start Time",
           description="Start time for the report generation (in UTC)"
       ),
       "end_time": Param("2024-02-01 00:00:00",
           type="string",
           format="date-time",
           title="End Time",
           description="End time for the report generation (in UTC)"
       ),
   }
)
```


Trigger DAG: generate_report

Select Recent Configurations

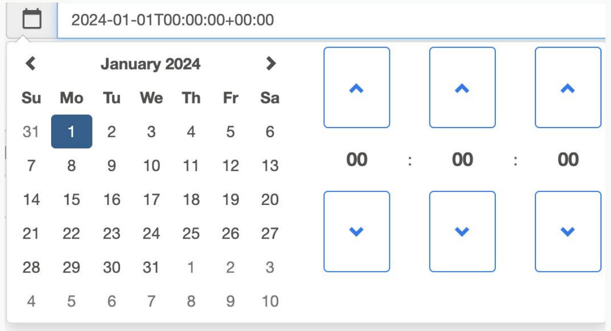
Default parameters

DAG conf Parameters

Select a City *:	SFO Please select a city code to generate report
Start Time *:	2024-01-01T00:00:00+00:00 Start time for the report generation (in UTC)
End Time *:	2024-02-01T00:00:00+00:00 End time for the report generation (in UTC)

Latest Airflow

- Mandatory vs non-mandatory fields
- Various types, int, decimal, string, bool, list, dict
- Length checks, value checks
- Date-time picker
- Type ahead suggestions
- Json forms
- Multi-selects
- Selection box with option labels
- Quick select prev run conf

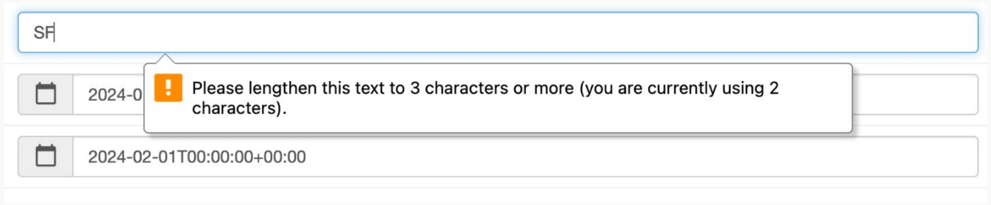


2024-01-01T00:00:00+00:00

January 2024

Su	Mo	Tu	We	Th	Fr	Sa
31	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3
4	5	6	7	8	9	10

00 : 00 : 00

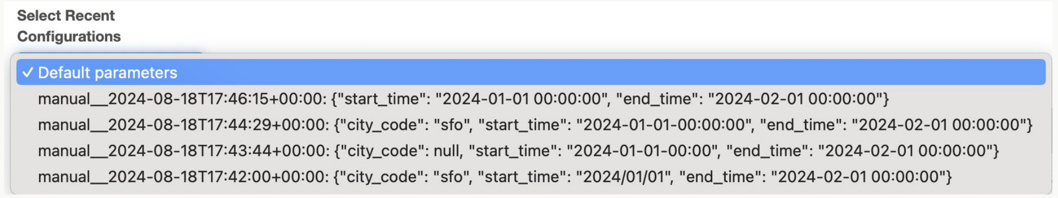


SF

2024-0

Please lengthen this text to 3 characters or more (you are currently using 2 characters).

2024-02-01T00:00:00+00:00



Select Recent Configurations

✓ Default parameters
manual__2024-08-18T17:46:15+00:00: {"start_time": "2024-01-01 00:00:00", "end_time": "2024-02-01 00:00:00"}
manual__2024-08-18T17:44:29+00:00: {"city_code": "sfo", "start_time": "2024-01-01-00:00:00", "end_time": "2024-02-01 00:00:00"}
manual__2024-08-18T17:43:44+00:00: {"city_code": null, "start_time": "2024-01-01-00:00", "end_time": "2024-02-01 00:00:00"}
manual__2024-08-18T17:42:00+00:00: {"city_code": "sfo", "start_time": "2024/01/01", "end_time": "2024-02-01 00:00:00"}

Future

- Possibility to extend Params class into custom params classes

```python
class MyCustomParam(Param):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        # Custom initialization logic

    def resolve(self, value: Any = NOTSET, suppress_exception: bool = False) -> Any:
        # Custom logic to resolve the value
```

Questions?

maheshwarisumit

@sumitmaheshwari