Extra Data Types

Up to now, you have been using common data types, like:

- `int`
- `float`
- `str`
- `bool`

But you can also use more complex data types.

And you will still have the same features as seen up to now:

- Great editor support.
- Data conversion from incoming requests.
- Data conversion for response data.
- Data validation.
- Automatic annotation and documentation.

Other data types

Here are some of the additional data types you can use:

- `UUID` :
  - A standard "Universally Unique Identifier", common as an ID in many databases and systems.
  - In requests and responses will be represented as a `str` .
- `datetime.datetime` :
  - A Python `datetime.datetime` .
  - In requests and responses will be represented as a `str` in ISO 8601 format, like: `2008-09-15T15:53:00+05:00` .
- `datetime.date` :
  - Python `datetime.date` .
  - In requests and responses will be represented as a `str` in ISO 8601 format, like: `2008-09-15` .
- `datetime.time` :
  - A Python `datetime.time` .
  - In requests and responses will be represented as a `str` in ISO 8601 format, like: `14:23:55.003` .
- `datetime.timedelta` :
  - A Python `datetime.timedelta` .
  - In requests and responses will be represented as a `float` of total seconds.
  - Pydantic also allows representing it as a "ISO 8601 time diff encoding", see the docs for more info.
- `frozenset` :
  - In requests and responses, treated the same as a `set` :
    - In requests, a list will be read, eliminating duplicates and converting it to a `set` .
    - In responses, the `set` will be converted to a `list` .
    - The generated schema will specify that the `set` values are unique (using JSON Schema's `uniqueItems`).
- `bytes` :
  - Standard Python `bytes` .
  - In requests and responses will be treated as `str` .
  - The generated schema will specify that it's a `str` with `binary` "format".
- `Decimal` :
  - Standard Python `Decimal` .
  - In requests and responses, handled the same as a `float` .
- You can check all the valid Pydantic data types here: Pydantic data types.

Example

Here's an example *path operation* with parameters using some of the above types.

Python 3.10+

```
from datetime import datetime, time, timedelta
from typing import Annotated
from uuid import UUID

from fastapi import Body, FastAPI

app = FastAPI()


@app.put("/items/{item_id}")
```

```
async def read_items(
    item_id: UUID,
    start_datetime: Annotated[datetime, Body()],
    end_datetime: Annotated[datetime, Body()],
    process_after: Annotated[timedelta, Body()],
    repeat_at: Annotated[time | None, Body()] = None,
):
    start_process = start_datetime + process_after
    duration = end_datetime - start_process
    return {
        "item_id": item_id,
        "start_datetime": start_datetime,
        "end_datetime": end_datetime,
        "process_after": process_after,
        "repeat_at": repeat_at,
        "start_process": start_process,
        "duration": duration,
    }
```

▸ Other versions and variants

Python 3.9+

```
from datetime import datetime, time, timedelta
from typing import Annotated, Union
from uuid import UUID

from fastapi import Body, FastAPI

app = FastAPI()


@app.put("/items/{item_id}")
async def read_items(
    item_id: UUID,
    start_datetime: Annotated[datetime, Body()],
    end_datetime: Annotated[datetime, Body()],
    process_after: Annotated[timedelta, Body()],
    repeat_at: Annotated[Union[time, None], Body()] = None,
):
    start_process = start_datetime + process_after
    duration = end_datetime - start_process
    return {
        "item_id": item_id,
        "start_datetime": start_datetime,
        "end_datetime": end_datetime,
        "process_after": process_after,
        "repeat_at": repeat_at,
        "start_process": start_process,
        "duration": duration,
    }
```

Python 3.8+

```
from datetime import datetime, time, timedelta
from typing import Union
from uuid import UUID

from fastapi import Body, FastAPI
from typing_extensions import Annotated

app = FastAPI()


@app.put("/items/{item_id}")
async def read_items(
    item_id: UUID,
    start_datetime: Annotated[datetime, Body()],
    end_datetime: Annotated[datetime, Body()],
    process_after: Annotated[timedelta, Body()],
    repeat_at: Annotated[Union[time, None], Body()] = None,
):
    start_process = start_datetime + process_after
    duration = end_datetime - start_process
    return {
        "item_id": item_id,
        "start_datetime": start_datetime,
        "end_datetime": end_datetime,
        "process_after": process_after,
        "repeat_at": repeat_at,
        "start_process": start_process,
        "duration": duration,
    }
```

Python 3.10+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```python
from datetime import datetime, time, timedelta
from uuid import UUID

from fastapi import Body, FastAPI

app = FastAPI()


@app.put("/items/{item_id}")
async def read_items(
    item_id: UUID,
    start_datetime: datetime = Body(),
    end_datetime: datetime = Body(),
    process_after: timedelta = Body(),
    repeat_at: time | None = Body(default=None),
):
    start_process = start_datetime + process_after
    duration = end_datetime - start_process
    return {
        "item_id": item_id,
        "start_datetime": start_datetime,
        "end_datetime": end_datetime,
        "process_after": process_after,
        "repeat_at": repeat_at,
        "start_process": start_process,
        "duration": duration,
    }
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```python
from datetime import datetime, time, timedelta
from typing import Union
from uuid import UUID

from fastapi import Body, FastAPI

app = FastAPI()


@app.put("/items/{item_id}")
async def read_items(
    item_id: UUID,
    start_datetime: datetime = Body(),
    end_datetime: datetime = Body(),
    process_after: timedelta = Body(),
    repeat_at: Union[time, None] = Body(default=None),
):
    start_process = start_datetime + process_after
    duration = end_datetime - start_process
    return {
        "item_id": item_id,
        "start_datetime": start_datetime,
        "end_datetime": end_datetime,
        "process_after": process_after,
        "repeat_at": repeat_at,
        "start_process": start_process,
        "duration": duration,
    }
```

Note that the parameters inside the function have their natural data type, and you can, for example, perform normal date manipulations, like:

Python 3.10+

```python
from datetime import datetime, time, timedelta
from typing import Annotated
from uuid import UUID

from fastapi import Body, FastAPI

app = FastAPI()


@app.put("/items/{item_id}")
async def read_items(
    item_id: UUID,
    start_datetime: Annotated[datetime, Body()],
    end_datetime: Annotated[datetime, Body()],
    process_after: Annotated[timedelta, Body()],
    repeat_at: Annotated[time | None, Body()] = None,
):
    start_process = start_datetime + process_after
    duration = end_datetime - start_process
    return {
        "item_id": item_id,
        "start_datetime": start_datetime,
```

```
        "end_datetime": end_datetime,
        "process_after": process_after,
        "repeat_at": repeat_at,
        "start_process": start_process,
        "duration": duration,
    }
```

▸ 🤓 Other versions and variants

Python 3.9+

```
from datetime import datetime, time, timedelta
from typing import Annotated, Union
from uuid import UUID

from fastapi import Body, FastAPI

app = FastAPI()


@app.put("/items/{item_id}")
async def read_items(
    item_id: UUID,
    start_datetime: Annotated[datetime, Body()],
    end_datetime: Annotated[datetime, Body()],
    process_after: Annotated[timedelta, Body()],
    repeat_at: Annotated[Union[time, None], Body()] = None,
):
    start_process = start_datetime + process_after
    duration = end_datetime - start_process
    return {
        "item_id": item_id,
        "start_datetime": start_datetime,
        "end_datetime": end_datetime,
        "process_after": process_after,
        "repeat_at": repeat_at,
        "start_process": start_process,
        "duration": duration,
    }
```

Python 3.8+

```
from datetime import datetime, time, timedelta
from typing import Union
from uuid import UUID

from fastapi import Body, FastAPI
from typing_extensions import Annotated

app = FastAPI()


@app.put("/items/{item_id}")
async def read_items(
    item_id: UUID,
    start_datetime: Annotated[datetime, Body()],
    end_datetime: Annotated[datetime, Body()],
    process_after: Annotated[timedelta, Body()],
    repeat_at: Annotated[Union[time, None], Body()] = None,
):
    start_process = start_datetime + process_after
    duration = end_datetime - start_process
    return {
        "item_id": item_id,
        "start_datetime": start_datetime,
        "end_datetime": end_datetime,
        "process_after": process_after,
        "repeat_at": repeat_at,
        "start_process": start_process,
        "duration": duration,
    }
```

Python 3.10+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from datetime import datetime, time, timedelta
from uuid import UUID

from fastapi import Body, FastAPI

app = FastAPI()


@app.put("/items/{item_id}")
async def read_items(
```

```
    item_id: UUID,
    start_datetime: datetime = Body(),
    end_datetime: datetime = Body(),
    process_after: timedelta = Body(),
    repeat_at: time | None = Body(default=None),
):
    start_process = start_datetime + process_after
    duration = end_datetime - start_process
    return {
        "item_id": item_id,
        "start_datetime": start_datetime,
        "end_datetime": end_datetime,
        "process_after": process_after,
        "repeat_at": repeat_at,
        "start_process": start_process,
        "duration": duration,
    }
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from datetime import datetime, time, timedelta
from typing import Union
from uuid import UUID

from fastapi import Body, FastAPI

app = FastAPI()


@app.put("/items/{item_id}")
async def read_items(
    item_id: UUID,
    start_datetime: datetime = Body(),
    end_datetime: datetime = Body(),
    process_after: timedelta = Body(),
    repeat_at: Union[time, None] = Body(default=None),
):
    start_process = start_datetime + process_after
    duration = end_datetime - start_process
    return {
        "item_id": item_id,
        "start_datetime": start_datetime,
        "end_datetime": end_datetime,
        "process_after": process_after,
        "repeat_at": repeat_at,
        "start_process": start_process,
        "duration": duration,
    }
```