Body - Multiple Parameters

Now that we have seen how to use `Path` and `Query`, let's see more advanced uses of request body declarations.

Mix `Path`, `Query` and body parameters

First, of course, you can mix `Path`, `Query` and request body parameter declarations freely and **FastAPI** will know what to do.

And you can also declare body parameters as optional, by setting the default to `None`:

Python 3.10+

```
from typing import Annotated

from fastapi import FastAPI, Path
from pydantic import BaseModel

app = FastAPI()


class Item(BaseModel):
    name: str
    description: str | None = None
    price: float
    tax: float | None = None


@app.put("/items/{item_id}")
async def update_item(
    item_id: Annotated[int, Path(title="The ID of the item to get", ge=0, le=1000)],
    q: str | None = None,
    item: Item | None = None,
):
    results = {"item_id": item_id}
    if q:
        results.update({"q": q})
    if item:
        results.update({"item": item})
    return results
```

▸ 🤓 Other versions and variants

Python 3.9+

```
from typing import Annotated, Union

from fastapi import FastAPI, Path
from pydantic import BaseModel

app = FastAPI()


class Item(BaseModel):
    name: str
    description: Union[str, None] = None
    price: float
    tax: Union[float, None] = None


@app.put("/items/{item_id}")
async def update_item(
    item_id: Annotated[int, Path(title="The ID of the item to get", ge=0, le=1000)],
    q: Union[str, None] = None,
    item: Union[Item, None] = None,
):
    results = {"item_id": item_id}
    if q:
        results.update({"q": q})
    if item:
        results.update({"item": item})
    return results
```

Python 3.8+

```
from typing import Union

from fastapi import FastAPI, Path
from pydantic import BaseModel
from typing_extensions import Annotated

app = FastAPI()


class Item(BaseModel):
    name: str
    description: Union[str, None] = None
    price: float
    tax: Union[float, None] = None
```

```
@app.put("/items/{item_id}")
async def update_item(
    item_id: Annotated[int, Path(title="The ID of the item to get", ge=0, le=1000)],
    q: Union[str, None] = None,
    item: Union[Item, None] = None,
):
    results = {"item_id": item_id}
    if q:
        results.update({"q": q})
    if item:
        results.update({"item": item})
    return results
```

Python 3.10+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from fastapi import FastAPI, Path
from pydantic import BaseModel

app = FastAPI()


class Item(BaseModel):
    name: str
    description: str | None = None
    price: float
    tax: float | None = None


@app.put("/items/{item_id}")
async def update_item(
    *,
    item_id: int = Path(title="The ID of the item to get", ge=0, le=1000),
    q: str | None = None,
    item: Item | None = None,
):
    results = {"item_id": item_id}
    if q:
        results.update({"q": q})
    if item:
        results.update({"item": item})
    return results
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from typing import Union

from fastapi import FastAPI, Path
from pydantic import BaseModel

app = FastAPI()


class Item(BaseModel):
    name: str
    description: Union[str, None] = None
    price: float
    tax: Union[float, None] = None


@app.put("/items/{item_id}")
async def update_item(
    *,
    item_id: int = Path(title="The ID of the item to get", ge=0, le=1000),
    q: Union[str, None] = None,
    item: Union[Item, None] = None,
):
    results = {"item_id": item_id}
    if q:
        results.update({"q": q})
    if item:
        results.update({"item": item})
    return results
```

Multiple body parameters

Note

Notice that, in this case, the `item` that would be taken from the body is optional. As it has a `None` default value.

Multiple body parameters

In the previous example, the *path operations* would expect a JSON body with the attributes of an `Item`, like:

```
{
    "name": "Foo",
    "description": "The pretender",
    "price": 42.0,
    "tax": 3.2
}
```

But you can also declare multiple body parameters, e.g. `item` and `user`:

Python 3.10+

```
from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()


class Item(BaseModel):
    name: str
    description: str | None = None
    price: float
    tax: float | None = None


class User(BaseModel):
    username: str
    full_name: str | None = None


@app.put("/items/{item_id}")
async def update_item(item_id: int, item: Item, user: User):
    results = {"item_id": item_id, "item": item, "user": user}
    return results
```

▸ 🤓 Other versions and variants

Python 3.8+

```
from typing import Union

from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()


class Item(BaseModel):
    name: str
    description: Union[str, None] = None
    price: float
    tax: Union[float, None] = None


class User(BaseModel):
    username: str
    full_name: Union[str, None] = None


@app.put("/items/{item_id}")
async def update_item(item_id: int, item: Item, user: User):
    results = {"item_id": item_id, "item": item, "user": user}
    return results
```

In this case, **FastAPI** will notice that there is more than one body parameter in the function (there are two parameters that are Pydantic models).

So, it will then use the parameter names as keys (field names) in the body, and expect a body like:

```
{
    "item": {
        "name": "Foo",
        "description": "The pretender",
        "price": 42.0,
        "tax": 3.2
    },
    "user": {
        "username": "dave",
        "full_name": "Dave Grohl"
    }
}
```

Note

Notice that even though the `item` was declared the same way as before, it is now expected to be inside of the body with a key `item`.

FastAPI will do the automatic conversion from the request, so that the parameter `item` receives its specific content and the same for `user`.

It will perform the validation of the compound data, and will document it like that for the OpenAPI schema and automatic docs.

Singular values in body

The same way there is a `Query` and `Path` to define extra data for query and path parameters, **FastAPI** provides an equivalent `Body`.

For example, extending the previous model, you could decide that you want to have another key `importance` in the same body, besides the `item` and `user`.

If you declare it as is, because it is a singular value, **FastAPI** will assume that it is a query parameter.

But you can instruct **FastAPI** to treat it as another body key using `Body`:

Python 3.10+

```
from typing import Annotated

from fastapi import Body, FastAPI
from pydantic import BaseModel

app = FastAPI()


class Item(BaseModel):
    name: str
    description: str | None = None
    price: float
    tax: float | None = None


class User(BaseModel):
    username: str
    full_name: str | None = None


@app.put("/items/{item_id}")
async def update_item(
    item_id: int, item: Item, user: User, importance: Annotated[int, Body()]
):
    results = {"item_id": item_id, "item": item, "user": user, "importance": importance}
    return results
```

▸ 🤓 Other versions and variants

Python 3.9+

```
from typing import Annotated, Union

from fastapi import Body, FastAPI
from pydantic import BaseModel

app = FastAPI()


class Item(BaseModel):
    name: str
    description: Union[str, None] = None
    price: float
    tax: Union[float, None] = None


class User(BaseModel):
    username: str
    full_name: Union[str, None] = None


@app.put("/items/{item_id}")
async def update_item(
    item_id: int, item: Item, user: User, importance: Annotated[int, Body()]
):
    results = {"item_id": item_id, "item": item, "user": user, "importance": importance}
    return results
```

Python 3.8+

```
from typing import Union

from fastapi import Body, FastAPI
from pydantic import BaseModel
from typing_extensions import Annotated

app = FastAPI()
```

```
class Item(BaseModel):
    name: str
    description: Union[str, None] = None
    price: float
    tax: Union[float, None] = None


class User(BaseModel):
    username: str
    full_name: Union[str, None] = None


@app.put("/items/{item_id}")
async def update_item(
    item_id: int, item: Item, user: User, importance: Annotated[int, Body()]
):
    results = {"item_id": item_id, "item": item, "user": user, "importance": importance}
    return results
```

Python 3.10+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from fastapi import Body, FastAPI
from pydantic import BaseModel

app = FastAPI()


class Item(BaseModel):
    name: str
    description: str | None = None
    price: float
    tax: float | None = None


class User(BaseModel):
    username: str
    full_name: str | None = None


@app.put("/items/{item_id}")
async def update_item(item_id: int, item: Item, user: User, importance: int = Body()):
    results = {"item_id": item_id, "item": item, "user": user, "importance": importance}
    return results
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from typing import Union

from fastapi import Body, FastAPI
from pydantic import BaseModel

app = FastAPI()


class Item(BaseModel):
    name: str
    description: Union[str, None] = None
    price: float
    tax: Union[float, None] = None


class User(BaseModel):
    username: str
    full_name: Union[str, None] = None


@app.put("/items/{item_id}")
async def update_item(item_id: int, item: Item, user: User, importance: int = Body()):
    results = {"item_id": item_id, "item": item, "user": user, "importance": importance}
    return results
```

In this case, **FastAPI** will expect a body like:

```
{
    "item": {
        "name": "Foo",
        "description": "The pretender",
```

```
        "price": 42.0,
        "tax": 3.2
    },
    "user": {
        "username": "dave",
        "full_name": "Dave Grohl"
    },
    "importance": 5
}
```

Again, it will convert the data types, validate, document, etc.

Multiple body params and query

Of course, you can also declare additional query parameters whenever you need, additional to any body parameters.

As, by default, singular values are interpreted as query parameters, you don't have to explicitly add a `Query`, you can just do:

```
q: Union[str, None] = None
```

Or in Python 3.10 and above:

```
q: str | None = None
```

For example:

Python 3.10+

```
from typing import Annotated

from fastapi import Body, FastAPI
from pydantic import BaseModel

app = FastAPI()


class Item(BaseModel):
    name: str
    description: str | None = None
    price: float
    tax: float | None = None


class User(BaseModel):
    username: str
    full_name: str | None = None


@app.put("/items/{item_id}")
async def update_item(
    *,
    item_id: int,
    item: Item,
    user: User,
    importance: Annotated[int, Body(gt=0)],
    q: str | None = None,
):
    results = {"item_id": item_id, "item": item, "user": user, "importance": importance}
    if q:
        results.update({"q": q})
    return results
```

▸ 🤓 Other versions and variants

Python 3.9+

```
from typing import Annotated, Union

from fastapi import Body, FastAPI
from pydantic import BaseModel

app = FastAPI()


class Item(BaseModel):
    name: str
    description: Union[str, None] = None
    price: float
    tax: Union[float, None] = None


class User(BaseModel):
    username: str
    full_name: Union[str, None] = None


@app.put("/items/{item_id}")
async def update_item(
    *,
```

```
    item_id: int,
    item: Item,
    user: User,
    importance: Annotated[int, Body(gt=0)],
    q: Union[str, None] = None,
):
    results = {"item_id": item_id, "item": item, "user": user, "importance": importance}
    if q:
        results.update({"q": q})
    return results
```

Python 3.8+

```
from typing import Union

from fastapi import Body, FastAPI
from pydantic import BaseModel
from typing_extensions import Annotated

app = FastAPI()


class Item(BaseModel):
    name: str
    description: Union[str, None] = None
    price: float
    tax: Union[float, None] = None


class User(BaseModel):
    username: str
    full_name: Union[str, None] = None


@app.put("/items/{item_id}")
async def update_item(
    *,
    item_id: int,
    item: Item,
    user: User,
    importance: Annotated[int, Body(gt=0)],
    q: Union[str, None] = None,
):
    results = {"item_id": item_id, "item": item, "user": user, "importance": importance}
    if q:
        results.update({"q": q})
    return results
```

Python 3.10+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from fastapi import Body, FastAPI
from pydantic import BaseModel

app = FastAPI()


class Item(BaseModel):
    name: str
    description: str | None = None
    price: float
    tax: float | None = None


class User(BaseModel):
    username: str
    full_name: str | None = None


@app.put("/items/{item_id}")
async def update_item(
    *,
    item_id: int,
    item: Item,
    user: User,
    importance: int = Body(gt=0),
    q: str | None = None,
):
    results = {"item_id": item_id, "item": item, "user": user, "importance": importance}
    if q:
        results.update({"q": q})
    return results
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from typing import Union

from fastapi import Body, FastAPI
from pydantic import BaseModel

app = FastAPI()


class Item(BaseModel):
    name: str
    description: Union[str, None] = None
    price: float
    tax: Union[float, None] = None


class User(BaseModel):
    username: str
    full_name: Union[str, None] = None


@app.put("/items/{item_id}")
async def update_item(
    *,
    item_id: int,
    item: Item,
    user: User,
    importance: int = Body(gt=0),
    q: Union[str, None] = None,
):
    results = {"item_id": item_id, "item": item, "user": user, "importance": importance}
    if q:
        results.update({"q": q})
    return results
```

Info

`Body` also has all the same extra validation and metadata parameters as `Query`, `Path` and others you will see later.

Embed a single body parameter

Let's say you only have a single `item` body parameter from a Pydantic model `Item`.

By default, **FastAPI** will then expect its body directly.

But if you want it to expect a JSON with a key `item` and inside of it the model contents, as it does when you declare extra body parameters, you can use the special `Body` parameter `embed`:

```
item: Item = Body(embed=True)
```

as in:

Python 3.10+

```
from typing import Annotated

from fastapi import Body, FastAPI
from pydantic import BaseModel

app = FastAPI()


class Item(BaseModel):
    name: str
    description: str | None = None
    price: float
    tax: float | None = None


@app.put("/items/{item_id}")
async def update_item(item_id: int, item: Annotated[Item, Body(embed=True)]):
    results = {"item_id": item_id, "item": item}
    return results
```

▸ 🤓 Other versions and variants

Python 3.9+

```
from typing import Annotated, Union

from fastapi import Body, FastAPI
from pydantic import BaseModel

app = FastAPI()
```

```
class Item(BaseModel):
    name: str
    description: Union[str, None] = None
    price: float
    tax: Union[float, None] = None


@app.put("/items/{item_id}")
async def update_item(item_id: int, item: Annotated[Item, Body(embed=True)]):
    results = {"item_id": item_id, "item": item}
    return results
```

Python 3.8+

```
from typing import Union

from fastapi import Body, FastAPI
from pydantic import BaseModel
from typing_extensions import Annotated

app = FastAPI()


class Item(BaseModel):
    name: str
    description: Union[str, None] = None
    price: float
    tax: Union[float, None] = None


@app.put("/items/{item_id}")
async def update_item(item_id: int, item: Annotated[Item, Body(embed=True)]):
    results = {"item_id": item_id, "item": item}
    return results
```

Python 3.10+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from fastapi import Body, FastAPI
from pydantic import BaseModel

app = FastAPI()


class Item(BaseModel):
    name: str
    description: str | None = None
    price: float
    tax: float | None = None


@app.put("/items/{item_id}")
async def update_item(item_id: int, item: Item = Body(embed=True)):
    results = {"item_id": item_id, "item": item}
    return results
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```python
from typing import Union

from fastapi import Body, FastAPI
from pydantic import BaseModel

app = FastAPI()


class Item(BaseModel):
    name: str
    description: Union[str, None] = None
    price: float
    tax: Union[float, None] = None


@app.put("/items/{item_id}")
async def update_item(item_id: int, item: Item = Body(embed=True)):
    results = {"item_id": item_id, "item": item}
    return results
```

In this case **FastAPI** will expect a body like:

```
{
    "item": {
        "name": "Foo",
        "description": "The pretender",
        "price": 42.0,
        "tax": 3.2
    }
}
```

instead of:

```
{
    "name": "Foo",
    "description": "The pretender",
    "price": 42.0,
    "tax": 3.2
}
```

Recap

You can add multiple body parameters to your *path operation function*, even though a request can only have a single body.

But **FastAPI** will handle it, give you the correct data in your function, and validate and document the correct schema in the *path operation*.

You can also declare singular values to be received as part of the body.

And you can instruct **FastAPI** to embed the body in a key even when there is only a single parameter declared.