

Body - Fields

The same way you can declare additional validation and metadata in *path operation function* parameters with `Query`, `Path` and `Body`, you can declare validation and metadata inside of Pydantic models using Pydantic's `Field`.

Import Field

First, you have to import it:

Python 3.10+

```
from typing import Annotated

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

app = FastAPI()

class Item(BaseModel):
    name: str
    description: str | None = Field(
        default=None, title="The description of the item", max_length=300
    )
    price: float = Field(gt=0, description="The price must be greater than zero")
    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, Field

app = FastAPI()

class Item(BaseModel):
    name: str
    description: Union[str, None] = Field(
        default=None, title="The description of the item", max_length=300
    )
    price: float = Field(gt=0, description="The price must be greater than zero")
    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, Field
from typing_extensions import Annotated

app = FastAPI()

class Item(BaseModel):
    name: str
    description: Union[str, None] = Field(
        default=None, title="The description of the item", max_length=300
    )
    price: float = Field(gt=0, description="The price must be greater than zero")
    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, Field

app = FastAPI()

class Item(BaseModel):
    name: str
    description: str | None = Field(
        default=None, title="The description of the item", max_length=300
    )
    price: float = Field(gt=0, description="The price must be greater than zero")
    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.

```
from typing import Union

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

app = FastAPI()

class Item(BaseModel):
    name: str
    description: Union[str, None] = Field(
        default=None, title="The description of the item", max_length=300
    )
    price: float = Field(gt=0, description="The price must be greater than zero")
    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
```

Warning

Notice that `Field` is imported directly from `pydantic`, not from `fastapi` as are all the rest (`Query`, `Path`, `Body`, etc).

Declare model attributes

You can then use `Field` with model attributes:

Python 3.10+

```
from typing import Annotated

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

app = FastAPI()

class Item(BaseModel):
    name: str
    description: str | None = Field(
        default=None, title="The description of the item", max_length=300
    )
    price: float = Field(gt=0, description="The price must be greater than zero")
    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, Field

app = FastAPI()

class Item(BaseModel):
    name: str
    description: Union[str, None] = Field(
        default=None, title="The description of the item", max_length=300
    )
    price: float = Field(gt=0, description="The price must be greater than zero")
    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, Field
from typing_extensions import Annotated

app = FastAPI()

class Item(BaseModel):
    name: str
    description: Union[str, None] = Field(
        default=None, title="The description of the item", max_length=300
    )
    price: float = Field(gt=0, description="The price must be greater than zero")
    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, Field

app = FastAPI()

class Item(BaseModel):
    name: str
    description: str | None = Field(
        default=None, title="The description of the item", max_length=300
    )
    price: float = Field(gt=0, description="The price must be greater than zero")
    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.

```
from typing import Union
```

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

app = FastAPI()

class Item(BaseModel):
    name: str
    description: Union[str, None] = Field(
        default=None, title="The description of the item", max_length=300
    )
    price: float = Field(gt=0, description="The price must be greater than zero")
    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
```

Field works the same way as Query , Path and Body , it has all the same parameters, etc.

Technical Details

Actually, Query , Path and others you'll see next create objects of subclasses of a common Param class, which is itself a subclass of Pydantic's FieldInfo class.

And Pydantic's Field returns an instance of FieldInfo as well.

Body also returns objects of a subclass of FieldInfo directly. And there are others you will see later that are subclasses of the Body class.

Remember that when you import Query , Path , and others from fastapi , those are actually functions that return special classes.

Tip

Notice how each model's attribute with a type, default value and Field has the same structure as a path operation function's parameter, with Field instead of Path , Query and Body .

Add extra information

You can declare extra information in Field , Query , Body , etc. And it will be included in the generated JSON Schema.

You will learn more about adding extra information later in the docs, when learning to declare examples.

Warning

Extra keys passed to Field will also be present in the resulting OpenAPI schema for your application. As these keys may not necessarily be part of the OpenAPI specification, some OpenAPI tools, for example the OpenAPI validator, may not work with your generated schema.

Recap

You can use Pydantic's Field to declare extra validations and metadata for model attributes.

You can also use the extra keyword arguments to pass additional JSON Schema metadata.

© 2018 Sebastián Ramírez
Licensed under the MIT License.
<https://fastapi.tiangolo.com/tutorial/body-fields/>

Exported from DevDocs — <https://devdocs.io>