Declare Request Example Data

You can declare examples of the data your app can receive.

Here are several ways to do it.

Extra JSON Schema data in Pydantic models

You can declare `examples` for a Pydantic model that will be added to the generated JSON Schema.

Pydantic v2

Pydantic v1

```
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

    model_config = {
        "json_schema_extra": {
            "examples": [
                {
                    "name": "Foo",
                    "description": "A very nice Item",
                    "price": 35.4,
                    "tax": 3.2,
                }
            ]
        }
    }


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

▸ 🤓 Other versions and variants

Python 3.10+

```
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

    model_config = {
        "json_schema_extra": {
            "examples": [
                {
                    "name": "Foo",
                    "description": "A very nice Item",
                    "price": 35.4,
                    "tax": 3.2,
                }
            ]
        }
    }


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

Python 3.8+

Python 3.10+

```
from fastapi import FastAPI
from pydantic import BaseModel
```

```python
app = FastAPI()


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

    class Config:
        schema_extra = {
            "examples": [
                {
                    "name": "Foo",
                    "description": "A very nice Item",
                    "price": 35.4,
                    "tax": 3.2,
                }
            ]
        }


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

▸ 🤓 Other versions and variants

Python 3.8+

```python
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 Config:
        schema_extra = {
            "examples": [
                {
                    "name": "Foo",
                    "description": "A very nice Item",
                    "price": 35.4,
                    "tax": 3.2,
                }
            ]
        }


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

That extra info will be added as-is to the output **JSON Schema** for that model, and it will be used in the API docs.

Pydantic v2

In Pydantic version 2, you would use the attribute `model_config`, that takes a `dict` as described in Pydantic's docs: Configuration.

You can set `"json_schema_extra"` with a `dict` containing any additional data you would like to show up in the generated JSON Schema, including `examples`.

Pydantic v1

In Pydantic version 1, you would use an internal class `Config` and `schema_extra`, as described in Pydantic's docs: Schema customization.

You can set `schema_extra` with a `dict` containing any additional data you would like to show up in the generated JSON Schema, including `examples`.

Tip

You could use the same technique to extend the JSON Schema and add your own custom extra info.

For example you could use it to add metadata for a frontend user interface, etc.

Info

OpenAPI 3.1.0 (used since FastAPI 0.99.0) added support for `examples`, which is part of the **JSON Schema** standard.

Before that, it only supported the keyword `example` with a single example. That is still supported by OpenAPI 3.1.0, but is deprecated and is not part of the JSON Schema standard. So you are encouraged to migrate `example` to `examples`. 🤓

You can read more at the end of this page.

`Field` additional arguments

When using `Field()` with Pydantic models, you can also declare additional `examples` :

Python 3.10+

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

app = FastAPI()


class Item(BaseModel):
    name: str = Field(examples=["Foo"])
    description: str | None = Field(default=None, examples=["A very nice Item"])
    price: float = Field(examples=[35.4])
    tax: float | None = Field(default=None, examples=[3.2])


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

▸ 🤓 Other versions and variants

Python 3.8+

```
from typing import Union

from fastapi import FastAPI
from pydantic import BaseModel, Field

app = FastAPI()


class Item(BaseModel):
    name: str = Field(examples=["Foo"])
    description: Union[str, None] = Field(default=None, examples=["A very nice Item"])
    price: float = Field(examples=[35.4])
    tax: Union[float, None] = Field(default=None, examples=[3.2])


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

`examples` in JSON Schema - OpenAPI

When using any of:

- `Path()`
- `Query()`
- `Header()`
- `Cookie()`
- `Body()`
- `Form()`
- `File()`

you can also declare a group of `examples` with additional information that will be added to their **JSON Schemas** inside of **OpenAPI**.

`Body` with `examples`

Here we pass `examples` containing one example of the data expected in `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
```

```python
    tax: float | None = None


@app.put("/items/{item_id}")
async def update_item(
    item_id: int,
    item: Annotated[
        Item,
        Body(
            examples=[
                {
                    "name": "Foo",
                    "description": "A very nice Item",
                    "price": 35.4,
                    "tax": 3.2,
                }
            ],
        ),
    ],
):
    results = {"item_id": item_id, "item": item}
    return results
```

► 🤓 Other versions and variants

Python 3.9+

```python
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(
            examples=[
                {
                    "name": "Foo",
                    "description": "A very nice Item",
                    "price": 35.4,
                    "tax": 3.2,
                }
            ],
        ),
    ],
):
    results = {"item_id": item_id, "item": item}
    return results
```

Python 3.8+

```python
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(
            examples=[
                {
                    "name": "Foo",
                    "description": "A very nice Item",
                    "price": 35.4,
                    "tax": 3.2,
                }
            ],
        ),
    ],
):
    results = {"item_id": item_id, "item": item}
    return results
```

Python 3.10+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```python
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(
        examples=[
            {
                "name": "Foo",
                "description": "A very nice Item",
                "price": 35.4,
                "tax": 3.2,
            }
        ],
    ),
):
    results = {"item_id": item_id, "item": item}
    return results
```

Python 3.8+ - non-Annotated

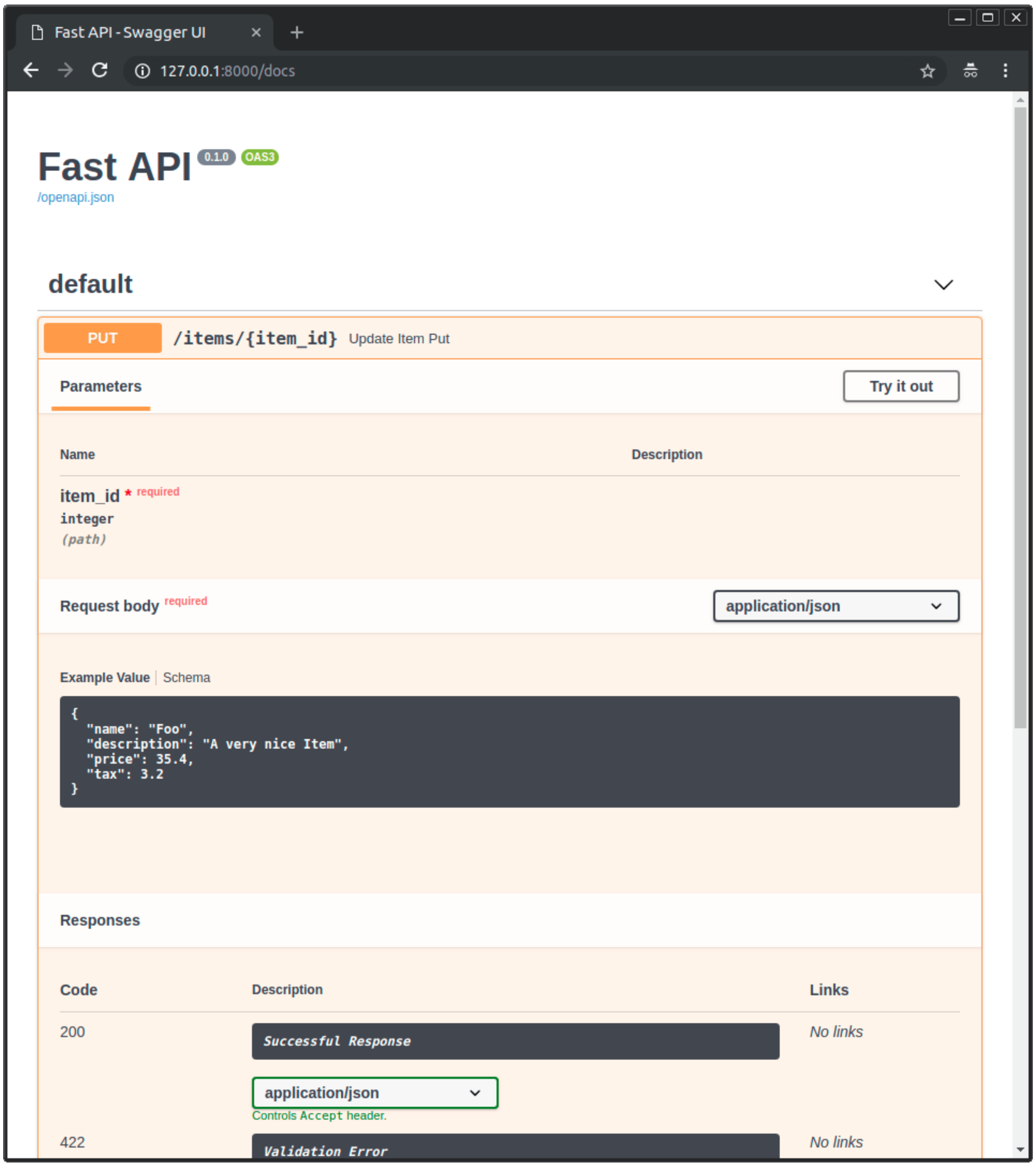
Tip

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(
        examples=[
            {
                "name": "Foo",
                "description": "A very nice Item",
                "price": 35.4,
                "tax": 3.2,
            }
        ],
    ),
):
    results = {"item_id": item_id, "item": item}
    return results
```

Example in the docs UI

With any of the methods above it would look like this in the `/docs`:



```
{
  "name": "Foo",
  "description": "A very nice Item",
  "price": 35.4,
  "tax": 3.2
}
```

`Body` **with multiple** `examples`

You can of course also pass multiple `examples`:

Python 3.10+

```python
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(
            examples=[
                {
                    "name": "Foo",
                    "description": "A very nice Item",
                    "price": 35.4,
                    "tax": 3.2,
                },
                {
                    "name": "Bar",
                    "price": "35.4",
                },
                {
                    "name": "Baz",
                    "price": "thirty five point four",
                },
            ],
        ),
    ],
):
    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(
            examples=[
                {
                    "name": "Foo",
                    "description": "A very nice Item",
                    "price": 35.4,
                    "tax": 3.2,
                },
                {
                    "name": "Bar",
                    "price": "35.4",
                },
                {
                    "name": "Baz",
                    "price": "thirty five point four",
                },
            ],
        ),
    ],
):
    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(
            examples=[
                {
                    "name": "Foo",
                    "description": "A very nice Item",
                    "price": 35.4,
                    "tax": 3.2,
                },
                {
                    "name": "Bar",
                    "price": "35.4",
                },
                {
                    "name": "Baz",
                    "price": "thirty five point four",
                },
            ],
        ),
    ],
):
    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(
        examples=[
            {
                "name": "Foo",
                "description": "A very nice Item",
                "price": 35.4,
                "tax": 3.2,
            },
            {
                "name": "Bar",
                "price": "35.4",
            },
            {
                "name": "Baz",
                "price": "thirty five point four",
            },
        ],
    ),
):
    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

app = FastAPI()


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


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

```python
async def update_item(
    *,
    item_id: int,
    item: Item = Body(
        examples=[
            {
                "name": "Foo",
                "description": "A very nice Item",
                "price": 35.4,
                "tax": 3.2,
            },
            {
                "name": "Bar",
                "price": "35.4",
            },
            {
                "name": "Baz",
                "price": "thirty five point four",
            },
        ],
    ),
):
    results = {"item_id": item_id, "item": item}
    return results
```

When you do this, the examples will be part of the internal **JSON Schema** for that body data.

Nevertheless, at the time of writing this, Swagger UI, the tool in charge of showing the docs UI, doesn't support showing multiple examples for the data in **JSON Schema**. But read below for a workaround.

OpenAPI-specific `examples`

Since before **JSON Schema** supported `examples` OpenAPI had support for a different field also called `examples` .

This **OpenAPI-specific** `examples` goes in another section in the OpenAPI specification. It goes in the **details for each** ***path operation***, not inside each JSON Schema.

And Swagger UI has supported this particular `examples` field for a while. So, you can use it to **show** different **examples in the docs UI**.

The shape of this OpenAPI-specific field `examples` is a `dict` with **multiple examples** (instead of a `list`), each with extra information that will be added to **OpenAPI** too.

This doesn't go inside of each JSON Schema contained in OpenAPI, this goes outside, in the *path operation* directly.

Using the `openapi_examples` Parameter

You can declare the OpenAPI-specific `examples` in FastAPI with the parameter `openapi_examples` for:

- `Path()`
- `Query()`
- `Header()`
- `Cookie()`
- `Body()`
- `Form()`
- `File()`

The keys of the `dict` identify each example, and each value is another `dict` .

Each specific example `dict` in the `examples` can contain:

- `summary` : Short description for the example.
- `description` : A long description that can contain Markdown text.
- `value` : This is the actual example shown, e.g. a `dict` .
- `externalValue` : alternative to `value` , a URL pointing to the example. Although this might not be supported by as many tools as `value` .

You can use it like this:

Python 3.10+

```python
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(
            openapi_examples={
                "normal": {
                    "summary": "A normal example",
                    "description": "A **normal** item works correctly.",
                    "value": {
                        "name": "Foo",
                        "description": "A very nice Item",
                        "price": 35.4,
                        "tax": 3.2,
                    },
                },
                "converted": {
                    "summary": "An example with converted data",
                    "description": "FastAPI can convert price `strings` to actual `numbers` automatically",
                    "value": {
                        "name": "Bar",
                        "price": "35.4",
                    },
                },
                "invalid": {
                    "summary": "Invalid data is rejected with an error",
                    "value": {
                        "name": "Baz",
                        "price": "thirty five point four",
                    },
                },
            },
        ),
    ],
):
    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(
            openapi_examples={
                "normal": {
                    "summary": "A normal example",
                    "description": "A **normal** item works correctly.",
                    "value": {
                        "name": "Foo",
                        "description": "A very nice Item",
                        "price": 35.4,
                        "tax": 3.2,
                    },
                },
                "converted": {
                    "summary": "An example with converted data",
                    "description": "FastAPI can convert price `strings` to actual `numbers` automatically",
                    "value": {
                        "name": "Bar",
                        "price": "35.4",
                    },
                },
                "invalid": {
                    "summary": "Invalid data is rejected with an error",
                    "value": {
                        "name": "Baz",
                        "price": "thirty five point four",
                    },
                },
            },
        ),
    ],
):
    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(
            openapi_examples={
                "normal": {
                    "summary": "A normal example",
                    "description": "A **normal** item works correctly.",
                    "value": {
                        "name": "Foo",
                        "description": "A very nice Item",
                        "price": 35.4,
                        "tax": 3.2,
                    },
                },
                "converted": {
                    "summary": "An example with converted data",
                    "description": "FastAPI can convert price `strings` to actual `numbers` automatically",
                    "value": {
                        "name": "Bar",
                        "price": "35.4",
                    },
                },
                "invalid": {
                    "summary": "Invalid data is rejected with an error",
                    "value": {
                        "name": "Baz",
                        "price": "thirty five point four",
                    },
                },
            },
        ),
    ],
):
    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(
        openapi_examples={
            "normal": {
                "summary": "A normal example",
                "description": "A **normal** item works correctly.",
                "value": {
                    "name": "Foo",
                    "description": "A very nice Item",
                    "price": 35.4,
                    "tax": 3.2,
                },
            },
            "converted": {
                "summary": "An example with converted data",
                "description": "FastAPI can convert price `strings` to actual `numbers` automatically",
                "value": {
                    "name": "Bar",
                    "price": "35.4",
                },
            },
            "invalid": {
                "summary": "Invalid data is rejected with an error",
                "value": {
                    "name": "Baz",
                    "price": "thirty five point four",
                },
            },
        },
    ),
):
    results = {"item_id": item_id, "item": item}
    return results
```

Python 3.8+ - non-Annotated

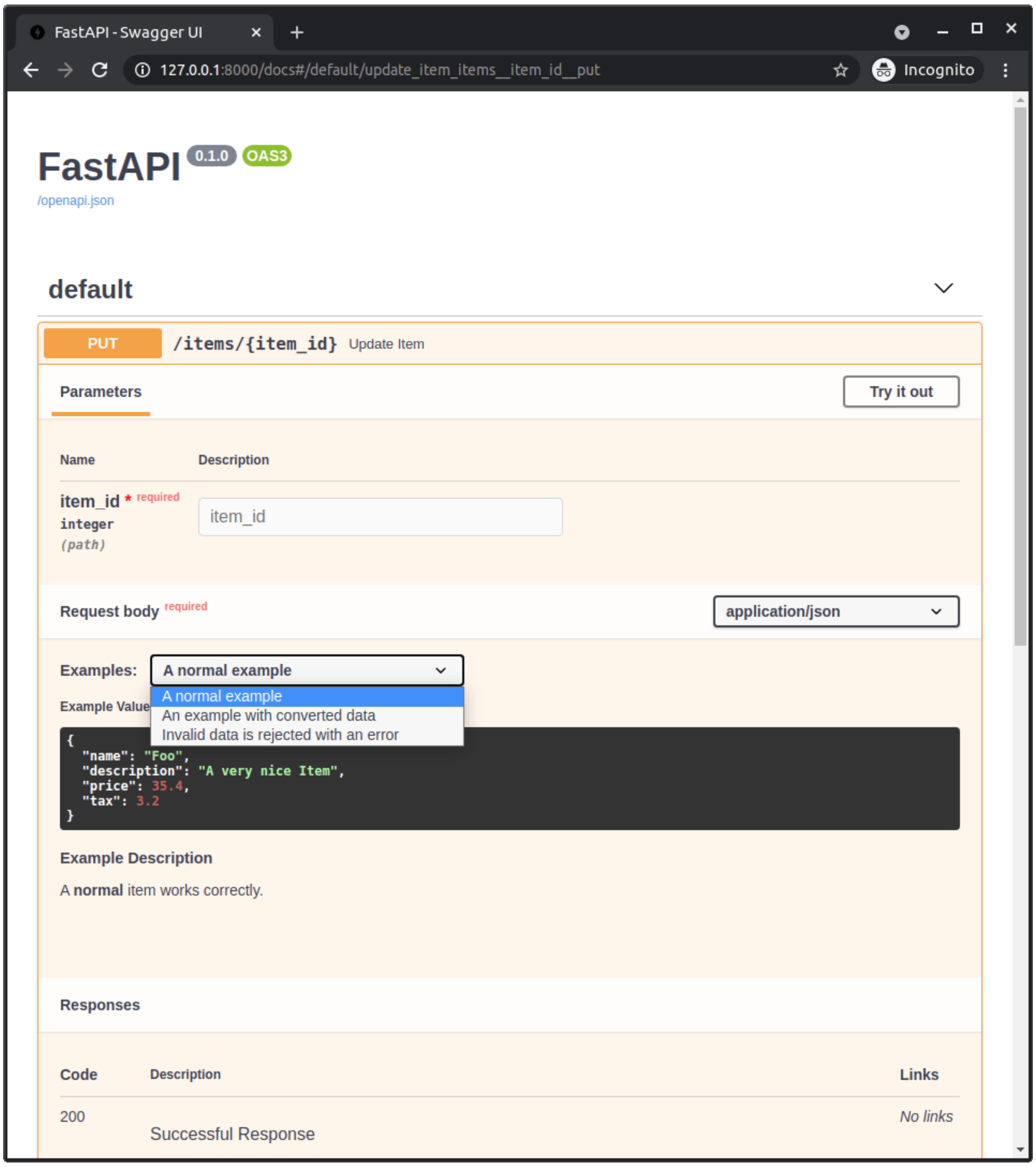
Tip

Prefer to use the `Annotated` version if possible.

```
from typing import Union

from fastapi import Body, FastAPI
from pydantic import BaseModel

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(
        openapi_examples={
            "normal": {
                "summary": "A normal example",
                "description": "A **normal** item works correctly.",
                "value": {
                    "name": "Foo",
                    "description": "A very nice Item",
                    "price": 35.4,
                    "tax": 3.2,
                },
            },
            "converted": {
                "summary": "An example with converted data",
                "description": "FastAPI can convert price `strings` to actual `numbers` automatically",
                "value": {
                    "name": "Bar",
                    "price": "35.4",
                },
            },
            "invalid": {
                "summary": "Invalid data is rejected with an error",
                "value": {
                    "name": "Baz",
                    "price": "thirty five point four",
                },
            },
        },
    ),
):
    results = {"item_id": item_id, "item": item}
    return results
```

OpenAPI Examples in the Docs UI

With `openapi_examples` added to `Body()` the `/docs` would look like:



```
{
  "name": "Foo",
  "description": "A very nice Item",
  "price": 35.4,
  "tax": 3.2
}
```

Technical Details

Tip

If you are already using **FastAPI** version **0.99.0 or above**, you can probably **skip** these details.

They are more relevant for older versions, before OpenAPI 3.1.0 was available.

You can consider this a brief OpenAPI and JSON Schema **history lesson**. 🤓

Warning

These are very technical details about the standards **JSON Schema** and **OpenAPI**.

If the ideas above already work for you, that might be enough, and you probably don't need these details, feel free to skip them.

Before OpenAPI 3.1.0, OpenAPI used an older and modified version of **JSON Schema**.

JSON Schema didn't have `examples`, so OpenAPI added its own `example` field to its own modified version.

OpenAPI also added `example` and `examples` fields to other parts of the specification:

- `Parameter Object` (in the specification) that was used by FastAPI's:
  - `Path()`
  - `Query()`
  - `Header()`

    - `Cookie()`
- `Request Body Object`, in the field `content`, on the `Media Type Object` (in the specification) that was used by FastAPI's:
    - `Body()`
    - `File()`
    - `Form()`

Info

This old OpenAPI-specific `examples` parameter is now `openapi_examples` since FastAPI `0.103.0`.

JSON Schema's `examples` field

But then JSON Schema added an `examples` field to a new version of the specification.

And then the new OpenAPI 3.1.0 was based on the latest version (JSON Schema 2020-12) that included this new field `examples`.

And now this new `examples` field takes precedence over the old single (and custom) `example` field, that is now deprecated.

This new `examples` field in JSON Schema is **just a `list`** of examples, not a dict with extra metadata as in the other places in OpenAPI (described above).

Info

Even after OpenAPI 3.1.0 was released with this new simpler integration with JSON Schema, for a while, Swagger UI, the tool that provides the automatic docs, didn't support OpenAPI 3.1.0 (it does since version 5.0.0 🎉).

Because of that, versions of FastAPI previous to 0.99.0 still used versions of OpenAPI lower than 3.1.0.

Pydantic and FastAPI `examples`

When you add `examples` inside a Pydantic model, using `schema_extra` or `Field(examples=["something"])` that example is added to the **JSON Schema** for that Pydantic model.

And that **JSON Schema** of the Pydantic model is included in the **OpenAPI** of your API, and then it's used in the docs UI.

In versions of FastAPI before 0.99.0 (0.99.0 and above use the newer OpenAPI 3.1.0) when you used `example` or `examples` with any of the other utilities (`Query()`, `Body()`, etc.) those examples were not added to the JSON Schema that describes that data (not even to OpenAPI's own version of JSON Schema), they were added directly to the *path operation* declaration in OpenAPI (outside the parts of OpenAPI that use JSON Schema).

But now that FastAPI 0.99.0 and above uses OpenAPI 3.1.0, that uses JSON Schema 2020-12, and Swagger UI 5.0.0 and above, everything is more consistent and the examples are included in JSON Schema.

Swagger UI and OpenAPI-specific `examples`

Now, as Swagger UI didn't support multiple JSON Schema examples (as of 2023-08-26), users didn't have a way to show multiple examples in the docs.

To solve that, FastAPI `0.103.0` **added support** for declaring the same old **OpenAPI-specific** `examples` field with the new parameter `openapi_examples`. 🤓

Summary

I used to say I didn't like history that much... and look at me now giving "tech history" lessons. 😅

In short, **upgrade to FastAPI 0.99.0 or above**, and things are much **simpler, consistent, and intuitive**, and you don't have to know all these historic details. 😎