Body - Updates

Update replacing with `PUT`

To update an item you can use the HTTP `PUT` operation.

You can use the `jsonable_encoder` to convert the input data to data that can be stored as JSON (e.g. with a NoSQL database). For example, converting `datetime` to `str`.

=== "Python 3.6 and above"

~~~
```Python hl_lines="30-35"
{!> ../../../docs_src/body_updates/tutorial001.py!}
```
~~~

=== "Python 3.9 and above"

~~~
```Python hl_lines="30-35"
{!> ../../../docs_src/body_updates/tutorial001_py39.py!}
```
~~~

=== "Python 3.10 and above"

~~~
```Python hl_lines="28-33"
{!> ../../../docs_src/body_updates/tutorial001_py310.py!}
```
~~~

`PUT` is used to receive data that should replace the existing data.

Warning about replacing

That means that if you want to update the item `bar` using `PUT` with a body containing:

```
{
    "name": "Barz",
    "price": 3,
    "description": None,
}
```

because it doesn't include the already stored attribute `"tax": 20.2`, the input model would take the default value of `"tax": 10.5`.

And the data would be saved with that "new" `tax` of `10.5`.

Partial updates with `PATCH`

You can also use the HTTP `PATCH` operation to *partially* update data.

This means that you can send only the data that you want to update, leaving the rest intact.

!!! Note `PATCH` is less commonly used and known than `PUT`.

```
And many teams use only `PUT`, even for partial updates.

You are **free** to use them however you want, **FastAPI** doesn't impose any
restrictions.

But this guide shows you, more or less, how they are intended to be used.
```

Using Pydantic's `exclude_unset` parameter

If you want to receive partial updates, it's very useful to use the parameter `exclude_unset` in Pydantic's model's `.dict()` .

Like `item.dict(exclude_unset=True)` .

That would generate a `dict` with only the data that was set when creating the `item` model, excluding default values.

Then you can use this to generate a `dict` with only the data that was set (sent in the request), omitting default values:

=== "Python 3.6 and above"

~~~
```Python hl_lines="34"
{!> ../../../docs_src/body_updates/tutorial002.py!}
```
~~~

=== "Python 3.9 and above"

~~~
```Python hl_lines="34"
{!> ../../../docs_src/body_updates/tutorial002_py39.py!}
```
~~~

=== "Python 3.10 and above"

~~~
```Python hl_lines="32"
{!> ../../../docs_src/body_updates/tutorial002_py310.py!}
```
~~~

Using Pydantic's `update` parameter

Now, you can create a copy of the existing model using `.copy()` , and pass the `update` parameter with a `dict` containing the data to update.

Like `stored_item_model.copy(update=update_data)` :

=== "Python 3.6 and above"

~~~
```Python hl_lines="35"
{!> ../../../docs_src/body_updates/tutorial002.py!}
```
~~~

=== "Python 3.9 and above"

~~~
```Python hl_lines="35"
{!> ../../../docs_src/body_updates/tutorial002_py39.py!}
```
~~~

=== "Python 3.10 and above"

~~~
```Python hl_lines="33"
{!> ../../../docs_src/body_updates/tutorial002_py310.py!}
```
~~~

Partial updates recap

In summary, to apply partial updates you would:

- (Optionally) use `PATCH` instead of `PUT`.
- Retrieve the stored data.
- Put that data in a Pydantic model.
- Generate a `dict` without default values from the input model (using `exclude_unset`).
    - This way you can update only the values actually set by the user, instead of overriding values already stored with default values in your model.
- Create a copy of the stored model, updating it's attributes with the received partial updates (using the `update` parameter).
- Convert the copied model to something that can be stored in your DB (for example, using the `jsonable_encoder`).
    - This is comparable to using the model's `.dict()` method again, but it makes sure (and converts) the values to data types that can be converted to JSON, for example, `datetime` to `str`.
- Save the data to your DB.
- Return the updated model.

=== "Python 3.6 and above"

~~~
```Python hl_lines="30-37"
{!> ../../../docs_src/body_updates/tutorial002.py!}
```
~~~

=== "Python 3.9 and above"

~~~
```Python hl_lines="30-37"
{!> ../../../docs_src/body_updates/tutorial002_py39.py!}
```
~~~

=== "Python 3.10 and above"

~~~
```Python hl_lines="28-35"
{!> ../../../docs_src/body_updates/tutorial002_py310.py!}
```
~~~

!!! tip You can actually use this same technique with an HTTP `PUT` operation.

```
But the example here uses `PATCH` because it was created for these use cases.
```

!!! note Notice that the input model is still validated.

```
So, if you want to receive partial updates that can omit all the attributes, you need
to have a model with all the attributes marked as optional (with default values or
`None`).

To distinguish from the models with all optional values for **updates** and models
with required values for **creation**, you can use the ideas described in [Extra
Models](extra-models.md){.internal-link target=_blank}.
```