Path Operation Configuration

There are several parameters that you can pass to your *path operation decorator* to configure it.

!!! warning Notice that these parameters are passed directly to the *path operation decorator*, not to your *path operation function*.

Response Status Code

You can define the (HTTP) `status_code` to be used in the response of your *path operation*.

You can pass directly the `int` code, like `404`.

But if you don't remember what each number code is for, you can use the shortcut constants in `status`:

=== "Python 3.6 and above"

```Python hl_lines="3  17"
{!> ../../../docs_src/path_operation_configuration/tutorial001.py!}
```

=== "Python 3.9 and above"

```Python hl_lines="3  17"
{!> ../../../docs_src/path_operation_configuration/tutorial001_py39.py!}
```

=== "Python 3.10 and above"

```Python hl_lines="1  15"
{!> ../../../docs_src/path_operation_configuration/tutorial001_py310.py!}
```

That status code will be used in the response and will be added to the OpenAPI schema.

!!! note "Technical Details" You could also use `from starlette import status`.

```
**FastAPI** provides the same `starlette.status` as `fastapi.status` just as a convenience f
```

Tags

You can add tags to your *path operation*, pass the parameter `tags` with a `list` of `str` (commonly just one `str`):

=== "Python 3.6 and above"

```Python hl_lines="17  22  27"
{!> ../../../docs_src/path_operation_configuration/tutorial002.py!}
```

```

=== "Python 3.9 and above"

```Python hl_lines="17  22  27"
{!> ../../../docs_src/path_operation_configuration/tutorial002_py39.py!}
```

=== "Python 3.10 and above"

```Python hl_lines="15  20  25"
{!> ../../../docs_src/path_operation_configuration/tutorial002_py310.py!}
```

They will be added to the OpenAPI schema and used by the automatic documentation interfaces:

Tags with Enums

If you have a big application, you might end up accumulating **several tags**, and you would want to make sure you always use the **same tag** for related *path operations*.

In these cases, it could make sense to store the tags in an `Enum`.

FastAPI supports that the same way as with plain strings:

```
Python hl_lines="1  8-10  13  18" {!../../../docs_src/path_operation_configuration/tutorial0
```

Summary and description

You can add a `summary` and `description`:

=== "Python 3.6 and above"

```Python hl_lines="20-21"
{!> ../../../docs_src/path_operation_configuration/tutorial003.py!}
```

=== "Python 3.9 and above"

```Python hl_lines="20-21"
{!> ../../../docs_src/path_operation_configuration/tutorial003_py39.py!}
```

=== "Python 3.10 and above"

```Python hl_lines="18-19"
{!> ../../../docs_src/path_operation_configuration/tutorial003_py310.py!}
```
```

Description from docstring

As descriptions tend to be long and cover multiple lines, you can declare the *path operation* description in the function docstring and **FastAPI** will read it from there.

You can write Markdown in the docstring, it will be interpreted and displayed correctly (taking into account docstring indentation).

=== "Python 3.6 and above"

```
```Python hl_lines="19-27"
{!> ../../../docs_src/path_operation_configuration/tutorial004.py!}
```
```

=== "Python 3.9 and above"

```
```Python hl_lines="19-27"
{!> ../../../docs_src/path_operation_configuration/tutorial004_py39.py!}
```
```

=== "Python 3.10 and above"

```
```Python hl_lines="17-25"
{!> ../../../docs_src/path_operation_configuration/tutorial004_py310.py!}
```
```

It will be used in the interactive docs:

Response description

You can specify the response description with the parameter `response_description`:

=== "Python 3.6 and above"

```
```Python hl_lines="21"
{!> ../../../docs_src/path_operation_configuration/tutorial005.py!}
```
```

=== "Python 3.9 and above"

```
```Python hl_lines="21"
{!> ../../../docs_src/path_operation_configuration/tutorial005_py39.py!}
```
```

=== "Python 3.10 and above"

```
```Python hl_lines="19"
{!> ../../../docs_src/path_operation_configuration/tutorial005_py310.py!}
```
```

!!! info Notice that `response_description` refers specifically to the response, the `description` refers to the *path operation* in general.

!!! check OpenAPI specifies that each *path operation* requires a response description.

```
So, if you don't provide one, **FastAPI** will automatically generate one of "Successful res
```

Deprecate a *path operation*

If you need to mark a *path operation* as deprecated, but without removing it, pass the parameter `deprecated`:

```
Python hl_lines="16" {!../../../docs_src/path_operation_configuration/tutorial006.py!}
```

It will be clearly marked as deprecated in the interactive docs:

Check how deprecated and non-deprecated *path operations* look like:

Recap

You can configure and add metadata for your *path operations* easily by passing parameters to the *path operation decorators*.