Extra Data Types

Up to now, you have been using common data types, like:

- `int`
- `float`
- `str`
- `bool`

But you can also use more complex data types.

And you will still have the same features as seen up to now:

- Great editor support.
- Data conversion from incoming requests.
- Data conversion for response data.
- Data validation.
- Automatic annotation and documentation.

Other data types

Here are some of the additional data types you can use:

- `UUID`:
    - A standard "Universally Unique Identifier", common as an ID in many databases and systems.
    - In requests and responses will be represented as a `str`.
- `datetime.datetime`:
    - A Python `datetime.datetime`.
    - In requests and responses will be represented as a `str` in ISO 8601 format, like: `2008-09-15T15:53:00+05:00`.
- `datetime.date`:
    - Python `datetime.date`.
    - In requests and responses will be represented as a `str` in ISO 8601 format, like: `2008-09-15`.
- `datetime.time`:
    - A Python `datetime.time`.
    - In requests and responses will be represented as a `str` in ISO 8601 format, like: `14:23:55.003`.
- `datetime.timedelta`:
    - A Python `datetime.timedelta`.
    - In requests and responses will be represented as a `float` of total seconds.
    - Pydantic also allows representing it as a "ISO 8601 time diff encoding", see the docs for more info.
- `frozenset`:
    - In requests and responses, treated the same as a `set`:
        - In requests, a list will be read, eliminating duplicates and converting it to a `set`.
        - In responses, the `set` will be converted to a `list`.
        - The generated schema will specify that the `set` values are unique (using JSON Schema's `uniqueItems`).
- `bytes`:

    - Standard Python `bytes`.
    - In requests and responses will be treated as `str`.
    - The generated schema will specify that it's a `str` with `binary` "format".
- `Decimal`:
    - Standard Python `Decimal`.
    - In requests and responses, handled the same as a `float`.
- You can check all the valid pydantic data types here: Pydantic data types.

Example

Here's an example *path operation* with parameters using some of the above types.

=== "Python 3.6 and above"

~~~
```Python hl_lines="1  3  12-16"
{!> ../../../docs_src/extra_data_types/tutorial001.py!}
```
~~~

=== "Python 3.10 and above"

~~~
```Python hl_lines="1  2  11-15"
{!> ../../../docs_src/extra_data_types/tutorial001_py310.py!}
```
~~~

Note that the parameters inside the function have their natural data type, and you can, for example, perform normal date manipulations, like:

=== "Python 3.6 and above"

~~~
```Python hl_lines="18-19"
{!> ../../../docs_src/extra_data_types/tutorial001.py!}
```
~~~

=== "Python 3.10 and above"

~~~
```Python hl_lines="17-18"
{!> ../../../docs_src/extra_data_types/tutorial001_py310.py!}
```
~~~