Request Files

You can define files to be uploaded by the client using `File`.

!!! info To receive uploaded files, first install `python-multipart`.

```
E.g. `pip install python-multipart`.

This is because uploaded files are sent as "form data".
```

Import `File`

Import `File` and `UploadFile` from `fastapi`:

```
{!../../../docs_src/request_files/tutorial001.py!}
```

Define `File` Parameters

Create file parameters the same way you would for `Body` or `Form`:

```
{!../../../docs_src/request_files/tutorial001.py!}
```

!!! info `File` is a class that inherits directly from `Form`.

```
But remember that when you import `Query`, `Path`, `File` and others from `fastapi`,
those are actually functions that return special classes.
```

!!! tip To declare File bodies, you need to use `File`, because otherwise the parameters would be interpreted as query parameters or body (JSON) parameters.

The files will be uploaded as "form data".

If you declare the type of your *path operation function* parameter as `bytes`, **FastAPI** will read the file for you and you will receive the contents as `bytes`.

Have in mind that this means that the whole contents will be stored in memory. This will work well for small files.

But there are several cases in which you might benefit from using `UploadFile`.

File Parameters with `UploadFile`

Define a file parameter with a type of `UploadFile`:

```
{!../../../docs_src/request_files/tutorial001.py!}
```

Using `UploadFile` has several advantages over `bytes`:

- You don't have to use `File()` in the default value of the parameter.
- It uses a "spooled" file:

    - A file stored in memory up to a maximum size limit, and after passing this limit it will be stored in disk.
- This means that it will work well for large files like images, videos, large binaries, etc. without consuming all the memory.
- You can get metadata from the uploaded file.
- It has a file-like `async` interface.
- It exposes an actual Python `SpooledTemporaryFile` object that you can pass directly to other libraries that expect a file-like object.

`UploadFile`

`UploadFile` has the following attributes:

- `filename` : A `str` with the original file name that was uploaded (e.g. `myimage.jpg`).
- `content_type` : A `str` with the content type (MIME type / media type) (e.g. `image/jpeg`).
- `file` : A `SpooledTemporaryFile` (a file-like object). This is the actual Python file that you can pass directly to other functions or libraries that expect a "file-like" object.

`UploadFile` has the following `async` methods. They all call the corresponding file methods underneath (using the internal `SpooledTemporaryFile`).

- `write(data)` : Writes `data` (`str` or `bytes`) to the file.
- `read(size)` : Reads `size` (`int`) bytes/characters of the file.
- `seek(offset)` : Goes to the byte position `offset` (`int`) in the file.
    - E.g., `await myfile.seek(0)` would go to the start of the file.
    - This is especially useful if you run `await myfile.read()` once and then need to read the contents again.
- `close()` : Closes the file.

As all these methods are `async` methods, you need to "await" them.

For example, inside of an `async` *path operation function* you can get the contents with:

```
contents = await myfile.read()
```

If you are inside of a normal `def` *path operation function*, you can access the `UploadFile.file` directly, for example:

```
contents = myfile.file.read()
```

!!! note " `async` Technical Details" When you use the `async` methods, **FastAPI** runs the file methods in a threadpool and awaits for them.

!!! note "Starlette Technical Details" **FastAPI**'s `UploadFile` inherits directly from **Starlette**'s `UploadFile` , but adds some necessary parts to make it compatible with **Pydantic** and the other parts of FastAPI.

What is "Form Data"

The way HTML forms (`<form></form>`) sends the data to the server normally uses a "special" encoding for that data, it's different from JSON.

FastAPI will make sure to read that data from the right place instead of JSON.

!!! note "Technical Details" Data from forms is normally encoded using the "media type" `application/x-www-form-urlencoded` when it doesn't include files.

```
But when the form includes files, it is encoded as `multipart/form-data`. If you use `File`, **FastAPI** will know it has to get the files from the correct part of the body.

If you want to read more about these encodings and form fields, head to the <a href="https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST" class="external-link" target="_blank"><abbr title="Mozilla Developer Network">MDN</abbr> web docs for <code>POST</code></a>.
```

!!! warning You can declare multiple `File` and `Form` parameters in a *path operation*, but you can't also declare `Body` fields that you expect to receive as JSON, as the request will have the body encoded using `multipart/form-data` instead of `application/json`.

```
This is not a limitation of **FastAPI**, it's part of the HTTP protocol.
```

Optional File Upload

You can make a file optional by using standard type annotations and setting a default value of `None`:

=== "Python 3.6 and above"

```
```Python hl_lines="9  17"
{!> ../../../docs_src/request_files/tutorial001_02.py!}
```
```

=== "Python 3.9 and above"

```
```Python hl_lines="7  14"
{!> ../../../docs_src/request_files/tutorial001_02_py310.py!}
```
```

`UploadFile` with Additional Metadata

You can also use `File()` with `UploadFile`, for example, to set additional metadata:

```
{!../../../docs_src/request_files/tutorial001_03.py!}
```

Multiple File Uploads

It's possible to upload several files at the same time.

They would be associated to the same "form field" sent using "form data".

To use that, declare a list of `bytes` or `UploadFile`:

=== "Python 3.6 and above"

```Python hl_lines="10  15"
{!> ../../../docs_src/request_files/tutorial002.py!}
```

=== "Python 3.9 and above"

```Python hl_lines="8  13"
{!> ../../../docs_src/request_files/tutorial002_py39.py!}
```

You will receive, as declared, a `list` of `bytes` or `UploadFile`s.

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

```
**FastAPI** provides the same `starlette.responses` as `fastapi.responses` just as a convenience for you, the developer. But most of the available responses come directly from Starlette.
```

Multiple File Uploads with Additional Metadata

And the same way as before, you can use `File()` to set additional parameters, even for `UploadFile`:

=== "Python 3.6 and above"

```Python hl_lines="18"
{!> ../../../docs_src/request_files/tutorial003.py!}
```

=== "Python 3.9 and above"

```Python hl_lines="16"
{!> ../../../docs_src/request_files/tutorial003_py39.py!}
```

Recap

Use `File`, `bytes`, and `UploadFile` to declare files to be uploaded in the request, sent as form data.