

[Source](#) > Core

## Core

This is the source code to fasthtml. You won't need to read this unless you want to understand how things are built behind the scenes, or need full details of a particular API. The notebook is converted to the Python module [fasthtml/core.py](#) using [nbdev](#).

## Imports and utils

```
import time

from IPython import display
from enum import Enum
from pprint import pprint

from fastcore.test import *
from starlette.testclient import TestClient
from starlette.requests import Headers
from starlette.datastructures import UploadFile
```

We write source code *first*, and then tests come *after*. The tests serve as both a means to confirm that the code works and also serves as working examples. The first declared function, [date](#), is an example of this pattern.

### date

[source](#)

```
date (s:str)
```

Convert *s* to a datetime

```
date('2pm')
datetime.datetime(2024, 8, 22, 14, 0)
```

### snake2hyphens

[source](#)

```
snake2hyphens (s:str)
```

Convert *s* from snake case to hyphenated and capitalised

```
snake2hyphens("snake_case")
```

```
'Snake-Case'
```

[source](#)

## HtmxHeaders

```
HtmxHeaders (boosted:str|None=None, current_url:str|None=None,
             history_restore_request:str|None=None, prompt:str|None=None,
             request:str|None=None, target:str|None=None,
             trigger_name:str|None=None, trigger:str|None=None)
```

```
def test_request(url: str='/', headers: dict={}, method: str='get') -> Request:
    scope = {
        'type': 'http',
        'method': method,
        'path': url,
        'headers': Headers(headers).raw,
        'query_string': b'',
        'scheme': 'http',
        'client': ('127.0.0.1', 8000),
        'server': ('127.0.0.1', 8000),
    }
    receive = lambda: {"body": b"", "more_body": False}
    return Request(scope, receive)
```

```
h = test_request(headers=Headers({'HX-Request':'1'}))
_get_htmx(h.headers)
```

```
HtmxHeaders(boosted=None, current_url=None, history_restore_request=None, prompt=None,
request='1', target=None, trigger_name=None, trigger=None)
```

[source](#)

## str2int

```
str2int (s)
```

Convert *s* to an *int*

```
str2int('1'),str2int('none')
```

```
(1, 0)
```

## Request and response

```
test_eq(_fix_anno(Union[str,None])('a'), 'a')
test_eq(_fix_anno(float)(0.9), 0.9)
```

```
test_eq(_fix_anno(int)('1'), 1)
test_eq(_fix_anno(int)(['1','2']), 2)
test_eq(_fix_anno(list[int])(['1','2']), [1,2])
test_eq(_fix_anno(list[int])('1'), [1])
```

```
d = dict(k=int, l=List[int])
test_eq(_form_arg('k', "1", d), 1)
test_eq(_form_arg('l', "1", d), [1])
test_eq(_form_arg('l', ["1","2"], d), [1,2])
```

## HTTPHeader

[source](#)

```
HTTPHeader (k:str, v:str)
```

## form2dict

[source](#)

```
form2dict (form:starlette.datastructures.FormData)
```

*Convert starlette form data to a dict*

```
d = [('a',1),('a',2),('b',0)]
fd = FormData(d)
res = form2dict(fd)
test_eq(res['a'], [1,2])
test_eq(res['b'], 0)
```

```
async def f(req):
    def _f(p:HTTPHeader): ...
    p = first(_sig(_f).parameters.values())
    result = await _from_body(req, p)
    return JSONResponse(result.__dict__)

app = Starlette(routes=[Route('/', f, methods=['POST'])])
client = TestClient(app)

d = dict(k='value1',v=['value2','value3'])
response = client.post('/', data=d)
print(response.json())

{'k': 'value1', 'v': 'value3'}
```

```
def g(req, this:Starlette, a:str, b:HTTPHeader): ...

async def f(req):
    a = await _wrap_req(req, _sig(g).parameters)
    return Response(str(a))
```

```
app = Starlette(routes=[Route('/', f, methods=['POST'])])
client = TestClient(app)

response = client.post('/?a=1', data=d)
print(response.text)
```

```
[<starlette.requests.Request object>, <starlette.applications.Starlette object>, '1',
HttpHeader(k='value1', v='value3')]
```

## flat\_xt

[source](#)

```
flat_xt (lst)
```

*Flatten lists*

```
x = ft('a', 1)
test_eq(flat_xt([x, x, [x,x]]), [x]*4)
test_eq(flat_xt(x), [x])
```

## Beforeware

[source](#)

```
Beforeware (f, skip=None)
```

*Initialize self. See help(type(self)) for accurate signature.*

## Websockets

```
def on_receive(self, msg:str): return f"Message text was: {msg}"
c = _ws_endp(on_receive)
app = Starlette(routes=[WebSocketRoute('/', _ws_endp(on_receive))])

cli = TestClient(app)
with cli.websocket_connect('/') as ws:
    ws.send_text('{"msg":"Hi!"}')
    data = ws.receive_text()
    assert data == 'Message text was: Hi!'
```

## Routing and application

### WS\_RouteX

[source](#)

```
WS_RouteX (path:str, recv, conn:<built-infunctioncallable>=None,  
           disconn:<built-infunctioncallable>=None, name=None,  
           middleware=None, hdrs=None, before=None)
```

*Initialize self. See help(type(self)) for accurate signature.*

---

## uri

[source](#)

```
uri (_arg, **kwargs)
```

---

## decode\_uri

[source](#)

```
decode_uri (s)
```

---

## StringConvertor.to\_string

[source](#)

```
StringConvertor.to_string (value:str)
```

---

## HTTPConnection.url\_path\_for

[source](#)

```
HTTPConnection.url_path_for (name:str, **path_params)
```

---

## flat\_tuple

[source](#)

```
flat_tuple (o)
```

*Flatten lists*

---

## RouteX

[source](#)

```
RouteX (path:str, endpoint, methods=None, name=None,  
        include_in_schema=True, middleware=None, hdrs=None, ftrs=None,  
        before=None, after=None, htmlkw=None, **bodykw)
```

*Initialize self. See help(type(self)) for accurate signature.*

---

## RouterX

[source](#)

```
RouterX (routes=None, redirect_slashes=True, default=None,
         on_startup=None, on_shutdown=None, lifespan=None,
         middleware=None, hdrs=None, ftrs=None, before=None, after=None,
         htmlkw=None, **bodykw)
```

*Initialize self. See help(type(self)) for accurate signature.*

## get\_key

[source](#)

```
get_key (key=None, fname='.sesskey')
```

```
get_key()
```

```
'a604e4a2-08e8-462d-aff9-15468891fe09'
```

## FastHTML

[source](#)

```
FastHTML (debug=False, routes=None, middleware=None,
          exception_handlers=None, on_startup=None, on_shutdown=None,
          lifespan=None, hdrs=None, ftrs=None, before=None, after=None,
          ws_hdr=False, surreal=True, htmx=True, default_hdrs=True,
          sess_cls=<class
            'starlette.middleware.sessions.SessionMiddleware'>,
          secret_key=None, session_cookie='session_', max_age=31536000,
          sess_path='/', same_site='lax', sess_https_only=False,
          sess_domain=None, key_fname='.sesskey', htmlkw=None, **bodykw)
```

\*Creates an application instance.

### Parameters:

- **debug** - Boolean indicating if debug tracebacks should be returned on errors.
- **routes** - A list of routes to serve incoming HTTP and WebSocket requests.
- **middleware** - A list of middleware to run for every request. A starlette application will always automatically include two middleware classes. `ServerErrorMiddleware` is added as the very outermost middleware, to handle any uncaught errors occurring anywhere in the entire stack. `ExceptionMiddleware` is added as the very innermost middleware, to deal with handled exception cases occurring in the routing or endpoints.
- **exception\_handlers** - A mapping of either integer status codes, or exception class types onto callables which handle the exceptions. Exception handler callables should be of the form `handler(request, exc) -> response` and may be either standard functions, or async functions.
- **on\_startup** - A list of callables to run on application startup. Startup handler callables do not take any arguments, and may be either standard functions, or async functions.
- **on\_shutdown** - A list of callables to run on application shutdown. Shutdown handler callables do not take any arguments, and may be either standard functions, or async functions.

- **lifespan** - A lifespan context function, which can be used to perform startup and shutdown tasks. This is a newer style that replaces the `on_startup` and `on_shutdown` handlers. Use one or the other, not both.\*

FastHTML.route

[source](#)

```
FastHTML.route (path:str=None, methods=None, name=None,
                include_in_schema=True)
```

Add a route at `path`

serve

[source](#)

```
serve (appname=None, app='app', host='0.0.0.0', port=None, reload=True,
      reload_includes:list[str]|str|None=None,
      reload_excludes:list[str]|str|None=None)
```

Run the app in an async server, with live reload set as the default.

	Type	Default	Details
appname	NoneType	None	Name of the module
app	str	app	App instance to be served
host	str	0.0.0.0	If host is 0.0.0.0 will convert to localhost
port	NoneType	None	If port is None it will default to 5001 or the PORT environment variable
reload	bool	True	Default is to reload the app upon code changes
reload_includes	list[str]   str   None	None	Additional files to watch for changes
reload_excludes	list[str]   str   None	None	Files to ignore for changes

Extras

cookie

[source](#)

```
cookie (key:str, value='', max_age=None, expires=None, path='/',
        domain=None, secure=False, httponly=False, samesite='lax')
```

Create a 'set-cookie' [HTTPHeader](#)

## reg\_re\_param

[source](#)

```
reg_re_param (m, s)
```

## MiddlewareBase

[source](#)

```
MiddlewareBase ()
```

Initialize self. See help(type(self)) for accurate signature.

## Tests

```
def get_cli(app): return app, TestClient(app), app.route
```

```
app, cli, rt = get_cli(FastHTML(secret_key='soopersecret'))
```

```
@rt("/hi")
def get(): return 'Hi there'

r = cli.get('/hi')
r.text
```

```
'Hi there'
```

```
@rt("/hi")
def post(): return 'Postal'

cli.post('/hi').text
```

```
'Postal'
```

```
@app.get("/hostie")
def show_host(req): return req.headers['host']

cli.get('/hostie').text
```

```
'testserver'
```



```
@rt
def yoyo(): return 'a yoyo'

cli.post('/yoyo').text

'a yoyo'
```

```
@app.get
def autopost(): return Html(Div('Text.', hx_post=yoyo()))
print(cli.get('/autopost').text)

<!doctype html>

<html><div hx-post="a yoyo">Text.</div>
</html>
```

```
@app.get
def autopost2(): return Html(Body(Div('Text.', cls='px-2', hx_post=show_host.rt(a='b'))))
print(cli.get('/autopost2').text)

<!doctype html>

<html><body><div class="px-2" hx-post="/hostie?a=b">Text.</div>
</body>
</html>
```

```
@app.get
def autoget2(): return Html(Div('Text.', hx_get=show_host))
print(cli.get('/autoget2').text)

<!doctype html>

<html><div hx-get="/hostie">Text.</div>
</html>
```

```
@rt('/user/{nm}', name='gday')
def get(nm:str=''): return f"Good day to you, {nm}!"
cli.get('/user/Alexis').text

'Good day to you, Alexis!'
```

```
@app.get
def autolink(): return Html(Div('Text.', link=uri('gday', nm='Alexis'))
print(cli.get('/autolink').text)

<!doctype html>

<html><div href="/user/Alexis">Text.</div>
</html>
```

```
@rt('/link')
def get(req): return f"{req.url_for('gday', nm='Alexis')}; {req.url_for('show_host')}"
```

cli.get('/link').text

'http://testserver/user/Alexis; http://testserver/hostie'

```
test_eq(app.router.url_path_for('gday', nm='Jeremy'), '/user/Jeremy')
```

```
hxhdr = {'headers':{'hx-request':"1"}}

@rt('/ft')
def get(): return Title('Foo'),H1('bar')

txt = cli.get('/ft').text
assert '<title>Foo</title>' in txt and '<h1>bar</h1>' in txt and '<html>' in txt

@rt('/xt2')
def get(): return H1('bar')

txt = cli.get('/xt2').text
assert '<title>FastHTML page</title>' in txt and '<h1>bar</h1>' in txt and '<html>' in tx

assert cli.get('/xt2', **hxhdr).text.strip() == '<h1>bar</h1>'

@rt('/xt3')
def get(): return Html(Head(Title('hi')), Body(P('there'))))

txt = cli.get('/xt3').text
assert '<title>FastHTML page</title>' not in txt and '<title>hi</title>' in txt and '<p>t
```

```
@rt('/oops')
def get(nope): return nope
test_warns(lambda: cli.get('/oops?nope=1'))
```

```
def test_r(cli, path, exp, meth='get', hx=False, **kwargs):
    if hx: kwargs['headers'] = {'hx-request':"1"}
    test_eq(getattr(cli, meth)(path, **kwargs).text, exp)

app.chk = 'foo'
ModelName = str_enum('ModelName', "alexnet", "resnet", "lenet")
fake_db = [{"name": "Foo"}, {"name": "Bar"}]
```

```
@rt('/html/{idx}')
async def get(idx:int): return Body(H4(f'Next is {idx+1}.'))

reg_re_param("imgext", "ico|gif|jpg|jpeg|webm")
```

```

@rt(r'/static/{path:path}{fn}.{ext:imgext}')
def get(fn:str, path:str, ext:str): return f"Getting {fn}.{ext} from /{path}"

@rt("/models/{nm}")
def get(nm:ModelName): return nm

@rt("/files/{path}")
async def get(path: Path): return path.with_suffix('.txt')

@rt("/items/")
def get(idx:int|None = 0): return fake_db[idx]

```

```

test_r(cli, '/html/1', '<body><h4>Next is 2.</h4>\n</body>\n', hx=True)
test_r(cli, '/static/foo/jph.ico', 'Getting jph.ico from /foo/')
test_r(cli, '/models/alexnet', 'alexnet')
test_r(cli, '/files/foo', 'foo.txt')
test_r(cli, '/items/?idx=1', '{"name":"Bar"}')
test_r(cli, '/items/', '{"name":"Foo"}')
assert cli.get('/items/?idx=g').status_code==404

```

```

@app.get("/booly/")
def _(coming:bool=True): return 'Coming' if coming else 'Not coming'

@app.get("/datie/")
def _(d:date): return d

@app.get("/ua")
async def _(user_agent:str): return user_agent

@app.get("/hxtest")
def _(htmx): return htmx.request

@app.get("/hxtest2")
def _(foo:HtmxHeaders, req): return foo.request

@app.get("/app")
def _(app): return app.chk

@app.get("/app2")
def _(foo:FastHTML): return foo.chk,HttpHeader("mykey", "myval")

```

```

test_r(cli, '/booly/?coming=true', 'Coming')
test_r(cli, '/booly/?coming=no', 'Not coming')
date_str = "17th of May, 2024, 2p"
test_r(cli, f'/datie/?d={date_str}', '2024-05-17 14:00:00')
test_r(cli, '/ua', 'FastHTML', headers={'User-Agent':'FastHTML'})
test_r(cli, '/hxtest' , '1', headers={'HX-Request':'1'})

```

```
test_r(cli, '/hxtest2', '1', headers={'HX-Request':'1'})
test_r(cli, '/app' , 'foo')
```

```
r = cli.get('/app2', **hxhdr)
test_eq(r.text, 'foo\n')
test_eq(r.headers['mykey'], 'myval')
```

```
@rt
def meta():
    return ((Title('hi'),H1('hi')),
            (Meta(property='image'), Meta(property='site_name')))
)

t = cli.post('/meta').text
assert re.search('<body>\s*<h1>hi</h1>\s*</body>', t)
assert '<meta' in t
```

```
@app.post('/profile/me')
def profile_update(username: str): return username

test_r(cli, '/profile/me', 'Alexis', 'post', data={'username' : 'Alexis'})
test_r(cli, '/profile/me', 'Missing required field: username', 'post', data={})
```

```
# Example post request with parameter that has a default value
@app.post('/pet/dog')
def pet_dog(dogname: str = None): return dogname

# Working post request with optional parameter
test_r(cli, '/pet/dog', '', 'post', data={})
```

```
@dataclass
class Bodie: a:int;b:str

@rt("/bodie/{nm}")
def post(nm:str, data:Bodie):
    res = asdict(data)
    res['nm'] = nm
    return res

@app.post("/bodied/")
def bodied(data:dict): return data

nt = namedtuple('Bodient', ['a','b'])

@app.post("/bodient/")
def bodient(data:nt): return asdict(data)

class BodieTD(TypedDict): a:int;b:str='foo'
```

```

@app.post("/bodietd/")
def bodient(data:BodieTD): return data

class Bodie2:
    a:int|None; b:str
    def __init__(self, a, b='foo'): store_attr()

@app.post("/bodie2/")
def bodie(d:Bodie2): return f"a: {d.a}; b: {d.b}"

```

```

from fasthtml.xtend import Titled

```

```

# Testing POST with Content-Type: application/json
@app.post("/")
def index(it: Bodie): return Titled("It worked!", P(f"{it.a}, {it.b}"))

s = json.dumps({"b": "Lorem", "a": 15})
response = cli.post('/', headers={"Content-Type": "application/json"}, data=s).text
assert "<title>It worked!</title>" in response and "<p>15, Lorem</p>" in response

```

```

# Testing POST with Content-Type: application/json
@app.post("/bodytext")
def index(body): return body

response = cli.post('/bodytext', headers={"Content-Type": "application/json"}, data=s).text
test_eq(response, '{"b": "Lorem", "a": 15}')

```

```

d = dict(a=1, b='foo')

test_r(cli, '/bodie/me', '{"a":1,"b":"foo","nm":"me"}', 'post', data=dict(a=1, b='foo', nm='me'))
test_r(cli, '/bodied/', '{"a":1,"b":"foo"}', 'post', data=d)
test_r(cli, '/bodie2/', 'a: 1; b: foo', 'post', data={'a':1})
test_r(cli, '/bodient/', '{"a":1,"b":"foo"}', 'post', data=d)
test_r(cli, '/bodietd/', '{"a":1,"b":"foo"}', 'post', data=d)

```

```

@rt("/setcookie")
def get(req): return cookie('now', datetime.now())

@rt("/getcookie")
def get(now:date): return f'Cookie was set at time {now.time()}'

print(cli.get('/setcookie').text)
time.sleep(0.01)
cli.get('/getcookie').text

```

```
'Cookie was set at time 15:51:57.629950'
```

```
@rt("/setsess")
def get(sess, foo:str=''):
    now = datetime.now()
    sess['auth'] = str(now)
    return f'Set to {now}'

@rt("/getsess")
def get(sess): return f'Session time: {sess["auth"]}'

print(cli.get('/setsess').text)
time.sleep(0.01)

cli.get('/getsess').text
```

```
Set to 2024-08-22 15:51:57.666909
```

```
'Session time: 2024-08-22 15:51:57.666909'
```

```
@rt("/sess-first")
def post(sess, name: str):
    sess["name"] = name
    return str(sess)

cli.post('/sess-first', data={'name': 2})

@rt("/getsess-all")
def get(sess): return sess['name']

test_eq(cli.get('/getsess-all').text, '2')
```

```
@rt("/upload")
async def post(uf:UploadFile): return (await uf.read()).decode()

with open('../../CHANGELOG.md', 'rb') as f:
    print(cli.post('/upload', files={'uf':f}, data={'msg':'Hello'}).text[:15])
```

```
# Release notes
```

```
@rt("/{fname:path}.{ext:static}")
async def get(fname:str, ext:str): return FileResponse(f'{fname}.{ext}')

assert 'These are the source notebooks for FastHTML' in cli.get('/README.txt').text
```

```
@rt("/form-submit/{list_id}")
def options(list_id: str):
    headers = {
        'Access-Control-Allow-Origin': '*',
        'Access-Control-Allow-Methods': 'POST',
```

```

        'Access-Control-Allow-Headers': '*',
    }
    return Response(status_code=200, headers=headers)

```

```

h = cli.options('/form-submit/2').headers
test_eq(h['Access-Control-Allow-Methods'], 'POST')

```

```

from fasthtml.authmw import user_pwd_auth

```

```

def _not_found(req, exc): return Div('nope')

app,cli,rt = get_cli(FastHTML(exception_handlers={404:_not_found}))

txt = cli.get('/').text
assert '<div>nope</div>' in txt
assert '<!doctype html>' in txt

```

```

auth = user_pwd_auth(testuser='spycraft')
app,cli,rt = get_cli(FastHTML(middleware=[auth]))

@rt("/locked")
def get(auth): return 'Hello, ' + auth

test_eq(cli.get('/locked').text, 'not authenticated')
test_eq(cli.get('/locked', auth=("testuser","spycraft")).text, 'Hello, testuser')

```

```

auth = user_pwd_auth(testuser='spycraft')
app,cli,rt = get_cli(FastHTML(middleware=[auth]))

@rt("/locked")
def get(auth): return 'Hello, ' + auth

test_eq(cli.get('/locked').text, 'not authenticated')
test_eq(cli.get('/locked', auth=("testuser","spycraft")).text, 'Hello, testuser')

```

```

from fasthtml.live_reload import FastHTMLWithLiveReload

```

```

hdrs, routes = app.router.hdrs, app.routes
app,cli,rt = get_cli(FastHTMLWithLiveReload())


@rt("/hi")
def get(): return 'Hi there'

test_eq(cli.get('/hi').text, "Hi there")

lr_hdrs, lr_routes = app.router.hdrs, app.routes

```

```
test_eq(len(lr_hdrs), len(hdrs)+1)
assert app.LIVE_RELOAD_HEADER in lr_hdrs
test_eq(len(lr_routes), len(routes)+1)
assert app.LIVE_RELOAD_ROUTE in lr_routes
```

 [Report an issue](#)