
Embassy



Super lightweight async HTTP server in pure Swift.

Please read: Embedded web server for iOS UI testing.

See also: Our lightweight web framework Ambassador based on Embassy

Features

- Swift 4 & 5
- iOS / tvOS / MacOS / Linux
- Super lightweight, only 1.5 K of lines
- Zero third-party dependency
- Async event loop based HTTP server, makes long-polling, delay and bandwidth throttling all possible
- HTTP Application based on SWSGI, super flexible
- IPV6 ready, also supports IPV4 (dual stack)
- Automatic testing covered

Example

Here's a simple example shows how Embassy works.

```
1 let loop = try! SelectorEventLoop(selector: try! KqueueSelector())
2 let server = DefaultHTTPServer(eventLoop: loop, port: 8080) {
3     (
4         environ: [String: Any],
5         startResponse: ((String, [(String, String)]) -> Void),
6         sendBody: ((Data) -> Void)
7     ) in
8     // Start HTTP response
9     startResponse("200 OK", [])
10    let pathInfo = environ["PATH_INFO"]! as! String
11    sendBody(Data("the path you're visiting is \(pathInfo.
12                debugDescription)".utf8))
13    // send EOF
14    sendBody(Data())
15 }
16 // Start HTTP server to listen on the port
```

```
17 try! server.start()
18
19 // Run event loop
20 loop.runForever()
```

Then you can visit [http://\[::1\]:8080/foo-bar](http://[::1]:8080/foo-bar) in the browser and see

```
1 the path you're visiting is "/foo-bar"
```

By default, the server will be bound only to the localhost interface (::1) for security reasons. If you want to access your server over an external network, you'll need to change the bind interface to all addresses:

```
1 let server = DefaultHTTPServer(eventLoop: loop, interface: "::", port:
  8080)
```

Async Event Loop

To use the async event loop, you can get it via key `embassy.event_loop` in `environ` dictionary and cast it to `EventLoop`. For example, you can create an SWSGI app which delays `sendBody` call like this

```
1 let app = { (
2   environ: [String: Any],
3   startResponse: ((String, [(String, String)]) -> Void),
4   sendBody: @escaping ((Data) -> Void)
5 ) in
6   startResponse("200 OK", [])
7
8   let loop = environ["embassy.event_loop"] as! EventLoop
9
10  loop.call(withDelay: 1) {
11    sendBody(Data("hello ".utf8))
12  }
13  loop.call(withDelay: 2) {
14    sendBody(Data("baby ".utf8))
15  }
16  loop.call(withDelay: 3) {
17    sendBody(Data("fin".utf8))
18    sendBody(Data())
19  }
20 }
```

Please notice that functions passed into SWSGI should be only called within the same thread for running the `EventLoop`, they are all not threadsafe, therefore, **you should not use GCD for delaying any call**. Instead, there are some methods from `EventLoop` you can use, and they are all thread-

safe

call(callback: (Void) -> Void)

Call given callback as soon as possible in the event loop

call(withDelay: TimeInterval, callback: (Void) -> Void)

Schedule given callback to `withDelay` seconds then call it in the event loop.

call(atTime: Date, callback: (Void) -> Void)

Schedule given callback to be called at `atTime` in the event loop. If the given time is in the past or zero, this methods works exactly like `call` with only callback parameter.

What's SWSGI (Swift Web Server Gateway Interface)?

SWSGI is a hat tip to Python's WSGI (Web Server Gateway Interface). It's a gateway interface enables web applications to talk to HTTP clients without knowing HTTP server implementation details.

It's defined as

```
1 public typealias SWSGI = (  
2     [String: Any],  
3     @escaping ((String, [(String, String)]) -> Void),  
4     @escaping ((Data) -> Void)  
5 ) -> Void
```

environ

It's a dictionary contains all necessary information about the request. It basically follows WSGI standard, except `wsgi.*` keys will be `swsgi.*` instead. For example:

```
1 [  
2     "SERVER_NAME": "[::1]",  
3     "SERVER_PROTOCOL" : "HTTP/1.1",  
4     "SERVER_PORT" : "53479",  
5     "REQUEST_METHOD": "GET",  
6     "SCRIPT_NAME" : "",  
7     "PATH_INFO" : "/",  
8     "HTTP_HOST": "[::1]:8889",
```

```

9   "HTTP_USER_AGENT" : "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_4)
    AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.102 Safari
    /537.36",
10  "HTTP_ACCEPT_LANGUAGE" : "en-US,en;q=0.8,zh-TW;q=0.6,zh;q=0.4,zh-CN;q
    =0.2",
11  "HTTP_CONNECTION" : "keep-alive",
12  "HTTP_ACCEPT" : "text/html,application/xhtml+xml,application/xml;q
    =0.9,image/webp,*/*;q=0.8",
13  "HTTP_ACCEPT_ENCODING" : "gzip, deflate, sdch",
14  "swsgi.version" : "0.1",
15  "swsgi.input" : (Function),
16  "swsgi.error" : "",
17  "swsgi.multiprocess" : false,
18  "swsgi.multithread" : false,
19  "swsgi.url_scheme" : "http",
20  "swsgi.run_once" : false
21 ]

```

To read request from body, you can use `swsgi.input`, for example

```

1  let input = environ["swsgi.input"] as! SWSGIInput
2  input { data in
3    // handle the body data here
4  }

```

An empty Data will be passed into the input data handler when EOF reached. Also please notice that, request body won't be read if `swsgi.input` is not set or set to nil. You can use `swsgi.input` as bandwidth control, set it to nil when you don't want to receive any data from client.

Some extra Embassy server specific keys are

- `embassy.connection` - `HTTPConnection` object for the request
- `embassy.event_loop` - `EventLoop` object
- `embassy.version` - Version of embassy as a String, e.g. `3.0.0`

startResponse

Function for starting to send HTTP response header to client, the first argument is status code with message, e.g. "200 OK". The second argument is headers, as a list of key value tuple.

To response HTTP header, you can do

```

1  startResponse("200 OK", [("Set-Cookie", "foo=bar")])

```

`startResponse` can only be called once per request, extra call will be simply ignored.

sendBody

Function for sending body data to client. You need to call `startResponse` first in order to call `sendBody`. If you don't call `startResponse` first, all calls to `sendBody` will be ignored. To send data, here you can do

```
1 sendBody(Data("hello".utf8))
```

To end the response data stream simply call `sendBody` with an empty Data.

```
1 sendBody(Data())
```

Install

CocoaPods

To install with CocoaPod, add Embassy to your Podfile:

```
1 pod 'Embassy', '~> 4.1'
```

Carthage

To install with Carthage, add Embassy to your Cartfile:

```
1 github "envoy/Embassy" ~> 4.1
```

Package Manager

Add it this Embassy repo in `Package.swift`, like this

```
1 import PackageDescription
2
3 let package = Package(
4     name: "EmbassyExample",
5     dependencies: [
6         .package(url: "https://github.com/envoy/Embassy.git",
7                 from: "4.1.4"),
8     ]
9 )
```

You can read this example project [here](#).