Streams

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-stream.rst, line 1)

Unknown directive type "currentmodule".

```
.. currentmodule:: asyncio
```

Source code: :source:`Lib/asyncio/streams.py`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-stream.rst, line 9); *backlink*

Unknown interpreted text role "source".

Streams are high-level async/await-ready primitives to work with network connections. Streams allow sending and receiving data without using callbacks or low-level protocols and transports.

Here is an example of a TCP echo client written using asyncio streams:

```
import asyncio

async def tcp_echo_client(message):
    reader, writer = await asyncio.open_connection(
        '127.0.0.1', 8888)

    print(f'Send: {message!r}')
    writer.write(message.encode())
    await writer.drain()

    data = await reader.read(100)
    print(f'Received: {data.decode()!r}')

    print('Close the connection')
    writer.close()
    await writer.wait_closed()

asyncio.run(tcp_echo_client('Hello World!'))
```

See also the Examples section below.

Stream Functions

The following top-level asyncio functions can be used to create and work with streams:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-stream.rst, line 51)

Unknown directive type "coroutinefunction".

```
.. coroutinefunction:: open_connection(host=None, port=None, *, \
                           limit=None, ssl=None, family=0, proto=0, \
                           flags=0, sock=None, local_addr=None, \
                           server_hostname=None, ssl_handshake_timeout=None, \
                           happy_eyeballs_delay=None, interleave=None)

   Establish a network connection and return a pair of
   ``(reader, writer)`` objects.

   The returned *reader* and *writer* objects are instances of
   :class:`StreamReader` and :class:`StreamWriter` classes.

   *limit* determines the buffer size limit used by the
   returned :class:`StreamReader` instance.  By default the *limit*
   is set to 64 KiB.

   The rest of the arguments are passed directly to
   :meth:`loop.create_connection`.

   .. versionchanged:: 3.7
      Added the *ssl_handshake_timeout* parameter.
```

```
   .. versionadded:: 3.8
      Added *happy_eyeballs_delay* and *interleave* parameters.

   .. versionchanged:: 3.10
      Removed the *loop* parameter.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-stream.rst, line 80)

Unknown directive type "coroutinefunction".

```
   .. coroutinefunction:: start_server(client_connected_cb, host=None, \
                             port=None, *, limit=None, \
                             family=socket.AF_UNSPEC, \
                             flags=socket.AI_PASSIVE, sock=None, \
                             backlog=100, ssl=None, reuse_address=None, \
                             reuse_port=None, ssl_handshake_timeout=None, \
                             start_serving=True)

      Start a socket server.

      The *client_connected_cb* callback is called whenever a new client
      connection is established.  It receives a ``(reader, writer)`` pair
      as two arguments, instances of the :class:`StreamReader` and
      :class:`StreamWriter` classes.

      *client_connected_cb* can be a plain callable or a
      :ref:`coroutine function <coroutine>`; if it is a coroutine function,
      it will be automatically scheduled as a :class:`Task`.

      *limit* determines the buffer size limit used by the
      returned :class:`StreamReader` instance.  By default the *limit*
      is set to 64 KiB.

      The rest of the arguments are passed directly to
      :meth:`loop.create_server`.

      .. versionchanged:: 3.7
         Added the *ssl_handshake_timeout* and *start_serving* parameters.

      .. versionchanged:: 3.10
         Removed the *loop* parameter.
```

Unix Sockets

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-stream.rst, line 115)

Unknown directive type "coroutinefunction".

```
   .. coroutinefunction:: open_unix_connection(path=None, *, limit=None, \
                            ssl=None, sock=None, server_hostname=None, \
                            ssl_handshake_timeout=None)

      Establish a Unix socket connection and return a pair of
      ``(reader, writer)``.

      Similar to :func:`open_connection` but operates on Unix sockets.

      See also the documentation of :meth:`loop.create_unix_connection`.

      .. availability:: Unix.

      .. versionchanged:: 3.7
         Added the *ssl_handshake_timeout* parameter.
         The *path* parameter can now be a :term:`path-like object`

      .. versionchanged:: 3.10
         Removed the *loop* parameter.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-

main\Doc\library\(cpython-main)(Doc)(library)asyncio-stream.rst, line 136)

Unknown directive type "coroutinefunction".

```
.. coroutinefunction:: start_unix_server(client_connected_cb, path=None, \
                           *, limit=None, sock=None, backlog=100, ssl=None, \
                           ssl_handshake_timeout=None, start_serving=True)

   Start a Unix socket server.

   Similar to :func:`start_server` but works with Unix sockets.

   See also the documentation of :meth:`loop.create_unix_server`.

   .. availability:: Unix.

   .. versionchanged:: 3.7
      Added the *ssl_handshake_timeout* and *start_serving* parameters.
      The *path* parameter can now be a :term:`path-like object`.

   .. versionchanged:: 3.10
      Removed the *loop* parameter.
```

StreamReader

Represents a reader object that provides APIs to read data from the IO stream.

It is not recommended to instantiate *StreamReader* objects directly; use :func:`open_connection` and :func:`start_server` instead.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-stream.rst, line 164); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-stream.rst, line 164); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-stream.rst, line 168)

Unknown directive type "coroutinemethod".

```
.. coroutinemethod:: read(n=-1)

   Read up to *n* bytes.  If *n* is not provided, or set to ``-1``,
   read until EOF and return all read bytes.

   If EOF was received and the internal buffer is empty,
   return an empty ``bytes`` object.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-stream.rst, line 176)

Unknown directive type "coroutinemethod".

```
.. coroutinemethod:: readline()

   Read one line, where "line" is a sequence of bytes
   ending with ``\n``.

   If EOF is received and ``\n`` was not found, the method
   returns partially read data.

   If EOF is received and the internal buffer is empty,
   return an empty ``bytes`` object.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-stream.rst, line 187)

Unknown directive type "coroutinemethod".

```
.. coroutinemethod:: readexactly(n)

   Read exactly *n* bytes.

   Raise an :exc:`IncompleteReadError` if EOF is reached before *n*
   can be read.  Use the :attr:`IncompleteReadError.partial`
   attribute to get the partially read data.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-stream.rst, line 195)

Unknown directive type "coroutinemethod".

```
.. coroutinemethod:: readuntil(separator=b'\\n')

   Read data from the stream until *separator* is found.

   On success, the data and separator will be removed from the
   internal buffer (consumed). Returned data will include the
   separator at the end.

   If the amount of data read exceeds the configured stream limit, a
   :exc:`LimitOverrunError` exception is raised, and the data
   is left in the internal buffer and can be read again.

   If EOF is reached before the complete separator is found,
   an :exc:`IncompleteReadError` exception is raised, and the internal
   buffer is reset.  The :attr:`IncompleteReadError.partial` attribute
   may contain a portion of the separator.

   .. versionadded:: 3.5.2
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-stream.rst, line 214)

Unknown directive type "method".

```
.. method:: at_eof()

   Return ``True`` if the buffer is empty and :meth:`feed_eof`
   was called.
```

StreamWriter

Represents a writer object that provides APIs to write data to the IO stream.

It is not recommended to instantiate *StreamWriter* objects directly; use :func:`open_connection` and :func:`start_server` instead.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-stream.rst, line 228); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-stream.rst, line 228); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-stream.rst, line 232)

Unknown directive type "method".

```
.. method:: write(data)

   The method attempts to write the *data* to the underlying socket immediately.
   If that fails, the data is queued in an internal write buffer until it can be
   sent.

   The method should be used along with the ``drain()`` method::
```

```
        stream.write(data)
        await stream.drain()
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-stream.rst, line 243)

Unknown directive type "method".

```
.. method:: writelines(data)

   The method writes a list (or any iterable) of bytes to the underlying socket
   immediately.
   If that fails, the data is queued in an internal write buffer until it can be
   sent.

   The method should be used along with the ``drain()`` method::

      stream.writelines(lines)
      await stream.drain()
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-stream.rst, line 255)

Unknown directive type "method".

```
.. method:: close()

   The method closes the stream and the underlying socket.

   The method should be used along with the ``wait_closed()`` method::

      stream.close()
      await stream.wait_closed()
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-stream.rst, line 264)

Unknown directive type "method".

```
.. method:: can_write_eof()

   Return ``True`` if the underlying transport supports
   the :meth:`write_eof` method, ``False`` otherwise.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-stream.rst, line 269)

Unknown directive type "method".

```
.. method:: write_eof()

   Close the write end of the stream after the buffered write
   data is flushed.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-stream.rst, line 274)

Unknown directive type "attribute".

```
.. attribute:: transport

   Return the underlying asyncio transport.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-stream.rst, line 278)

Unknown directive type "method".

```
   .. method:: get_extra_info(name, default=None)

      Access optional transport information; see
      :meth:`BaseTransport.get_extra_info` for details.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-stream.rst, line 283)

Unknown directive type "coroutinemethod".

```
   .. coroutinemethod:: drain()

      Wait until it is appropriate to resume writing to the stream.
      Example::

          writer.write(data)
          await writer.drain()

      This is a flow control method that interacts with the underlying
      IO write buffer.  When the size of the buffer reaches
      the high watermark, *drain()* blocks until the size of the
      buffer is drained down to the low watermark and writing can
      be resumed.  When there is nothing to wait for, the :meth:`drain`
      returns immediately.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-stream.rst, line 298)

Unknown directive type "method".

```
   .. method:: is_closing()

      Return ``True`` if the stream is closed or in the process of
      being closed.

      .. versionadded:: 3.7
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-stream.rst, line 305)

Unknown directive type "coroutinemethod".

```
   .. coroutinemethod:: wait_closed()

      Wait until the stream is closed.

      Should be called after :meth:`close` to wait until the underlying
      connection is closed.

      .. versionadded:: 3.7
```

Examples

TCP echo client using streams

TCP echo client using the :func:`asyncio.open_connection` function:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-stream.rst, line 323); *backlink*

Unknown interpreted text role "func".

```
import asyncio

async def tcp_echo_client(message):
    reader, writer = await asyncio.open_connection(
        '127.0.0.1', 8888)

    print(f'Send: {message!r}')
    writer.write(message.encode())
    await writer.drain()
```

```
    data = await reader.read(100)
    print(f'Received: {data.decode()!r}')

    print('Close the connection')
    writer.close()

asyncio.run(tcp_echo_client('Hello World!'))
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-stream.rst, line 344)

Unknown directive type "seealso".

```
.. seealso::

   The :ref:`TCP echo client protocol <asyncio_example_tcp_echo_client_protocol>`
   example uses the low-level :meth:`loop.create_connection` method.
```

TCP echo server using streams

TCP echo server using the :func:`asyncio.start_server` function:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-stream.rst, line 355); *backlink*

Unknown interpreted text role "func".

```
import asyncio

async def handle_echo(reader, writer):
    data = await reader.read(100)
    message = data.decode()
    addr = writer.get_extra_info('peername')

    print(f"Received {message!r} from {addr!r}")

    print(f"Send: {message!r}")
    writer.write(data)
    await writer.drain()

    print("Close the connection")
    writer.close()

async def main():
    server = await asyncio.start_server(
        handle_echo, '127.0.0.1', 8888)

    addrs = ', '.join(str(sock.getsockname()) for sock in server.sockets)
    print(f'Serving on {addrs}')

    async with server:
        await server.serve_forever()

asyncio.run(main())
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-stream.rst, line 386)

Unknown directive type "seealso".

```
.. seealso::

   The :ref:`TCP echo server protocol <asyncio_example_tcp_echo_server_protocol>`
   example uses the :meth:`loop.create_server` method.
```

Get HTTP headers

Simple example querying HTTP headers of the URL passed on the command line:

```
import asyncio
import urllib.parse
import sys
```

```
async def print_http_headers(url):
    url = urllib.parse.urlsplit(url)
    if url.scheme == 'https':
        reader, writer = await asyncio.open_connection(
            url.hostname, 443, ssl=True)
    else:
        reader, writer = await asyncio.open_connection(
            url.hostname, 80)

    query = (
        f"HEAD {url.path or '/'} HTTP/1.0\r\n"
        f"Host: {url.hostname}\r\n"
        f"\r\n"
    )

    writer.write(query.encode('latin-1'))
    while True:
        line = await reader.readline()
        if not line:
            break

        line = line.decode('latin1').rstrip()
        if line:
            print(f'HTTP header> {line}')

    # Ignore the body, close the socket
    writer.close()

url = sys.argv[1]
asyncio.run(print_http_headers(url))
```

Usage:

```
python example.py http://example.com/path/page.html
```

or with HTTPS:

```
python example.py https://example.com/path/page.html
```

Register an open socket to wait for data using streams

Coroutine waiting until a socket receives data using the :func:`open_connection` function:

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-stream.rst, line 447); *backlink***
>
> Unknown interpreted text role "func".

```
import asyncio
import socket

async def wait_for_data():
    # Get a reference to the current event loop because
    # we want to access low-level APIs.
    loop = asyncio.get_running_loop()

    # Create a pair of connected sockets.
    rsock, wsock = socket.socketpair()

    # Register the open socket to wait for data.
    reader, writer = await asyncio.open_connection(sock=rsock)

    # Simulate the reception of data from the network
    loop.call_soon(wsock.send, 'abc'.encode())

    # Wait for data
    data = await reader.read(100)

    # Got data, we are done: close the socket
    print("Received:", data.decode())
    writer.close()

    # Close the second socket
    wsock.close()

asyncio.run(wait_for_data())
```

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-**

main\Doc\library\(cpython-main)(Doc)(library)asyncio-stream.rst, line 479)

Unknown directive type "seealso".

```
.. seealso::

   The :ref:`register an open socket to wait for data using a protocol
   <asyncio_example_create_connection>` example uses a low-level protocol and
   the :meth:`loop.create_connection` method.

   The :ref:`watch a file descriptor for read events
   <asyncio_example_watch_fd>` example uses the low-level
   :meth:`loop.add_reader` method to watch a file descriptor.
```