Event Loop

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

Unknown directive type "currentmodule".

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

Source code: :source:`Lib/asyncio/events.py`, :source:`Lib/asyncio/base_events.py`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 8); *backlink*

Unknown interpreted text role "source".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 8); *backlink*

Unknown interpreted text role "source".

Preface

The event loop is the core of every asyncio application. Event loops run asynchronous tasks and callbacks, perform network IO operations, and run subprocesses.

Application developers should typically use the high-level asyncio functions, such as :func:`asyncio.run`, and should rarely need to reference the loop object or call its methods. This section is intended mostly for authors of lower-level code, libraries, and frameworks, who need finer control over the event loop behavior.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 19); *backlink*

Unknown interpreted text role "func".

Obtaining the Event Loop

The following low-level functions can be used to get, set, or create an event loop:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 30)

Unknown directive type "function".

```
.. function:: get_running_loop()

   Return the running event loop in the current OS thread.

   If there is no running event loop a :exc:`RuntimeError` is raised.
   This function can only be called from a coroutine or a callback.

   .. versionadded:: 3.7
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 39)

Unknown directive type "function".

```
.. function:: get_event_loop()

   Get the current event loop.

   If there is no current event loop set in the current OS thread,
   the OS thread is main, and :func:`set_event_loop` has not yet
   been called, asyncio will create a new event loop and set it as the
   current one.
```

```
Because this function has rather complex behavior (especially
when custom event loop policies are in use), using the
:func:`get_running_loop` function is preferred to :func:`get_event_loop`
in coroutines and callbacks.

Consider also using the :func:`asyncio.run` function instead of using
lower level functions to manually create and close an event loop.

.. deprecated:: 3.10
   Deprecation warning is emitted if there is no running event loop.
   In future Python releases, this function will be an alias of
   :func:`get_running_loop`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 61)

Unknown directive type "function".

```
.. function:: set_event_loop(loop)

   Set *loop* as a current event loop for the current OS thread.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 65)

Unknown directive type "function".

```
.. function:: new_event_loop()

   Create and return a new event loop object.
```

Note that the behaviour of :func:`get_event_loop`, :func:`set_event_loop`, and :func:`new_event_loop` functions can be altered by :ref:`setting a custom event loop policy <asyncio-policies>`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 69); *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-eventloop.rst, line 69); *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-eventloop.rst, line 69); *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-eventloop.rst, line 69); *backlink*

Unknown interpreted text role "ref".

Contents

This documentation page contains the following sections:

- The Event Loop Methods section is the reference documentation of the event loop APIs;
- The Callback Handles section documents the :class:`Handle` and :class:`TimerHandle` instances which are returned from scheduling methods such as :meth:`loop.call_soon` and :meth:`loop.call_later`;

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 81); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 81); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 81); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 81); *backlink*

Unknown interpreted text role "meth".

- The Server Objects section documents types returned from event loop methods like :meth:`loop.create_server`;

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 85); *backlink*

Unknown interpreted text role "meth".

- The Event Loop Implementations section documents the :class:`SelectorEventLoop` and :class:`ProactorEventLoop` classes;

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 88); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 88); *backlink*

Unknown interpreted text role "class".

- The Examples section showcases how to work with some event loop APIs.

Event Loop Methods

Event loops have **low-level** APIs for the following:

- Running and stopping the loop
- Scheduling callbacks
- Scheduling delayed callbacks
- Creating Futures and Tasks
- Opening network connections
- Creating network servers
- Transferring files
- TLS Upgrade
- Watching file descriptors
- Working with socket objects directly
- DNS
- Working with pipes
- Unix signals
- Executing code in thread or process pools
- Error Handling API
- Enabling debug mode
- Running Subprocesses

Running and stopping the loop

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 110)

Unknown directive type "method".

```
.. method:: loop.run_until_complete(future)

   Run until the *future* (an instance of :class:`Future`) has
   completed.

   If the argument is a :ref:`coroutine object <coroutine>` it
   is implicitly scheduled to run as a :class:`asyncio.Task`.

   Return the Future's result or raise its exception.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 120)

Unknown directive type "method".

```
.. method:: loop.run_forever()

   Run the event loop until :meth:`stop` is called.

   If :meth:`stop` is called before :meth:`run_forever()` is called,
   the loop will poll the I/O selector once with a timeout of zero,
   run all callbacks scheduled in response to I/O events (and
   those that were already scheduled), and then exit.

   If :meth:`stop` is called while :meth:`run_forever` is running,
   the loop will run the current batch of callbacks and then exit.
   Note that new callbacks scheduled by callbacks will not run in this
   case; instead, they will run the next time :meth:`run_forever` or
   :meth:`run_until_complete` is called.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 135)

Unknown directive type "method".

```
.. method:: loop.stop()

   Stop the event loop.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 139)

Unknown directive type "method".

```
.. method:: loop.is_running()

   Return ``True`` if the event loop is currently running.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 143)

Unknown directive type "method".

```
.. method:: loop.is_closed()

   Return ``True`` if the event loop was closed.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 147)

Unknown directive type "method".

```
.. method:: loop.close()
```

```
Close the event loop.

The loop must not be running when this function is called.
Any pending callbacks will be discarded.

This method clears all queues and shuts down the executor, but does
not wait for the executor to finish.

This method is idempotent and irreversible.  No other methods
should be called after the event loop is closed.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 160)

Unknown directive type "coroutinemethod".

```
.. coroutinemethod:: loop.shutdown_asyncgens()

   Schedule all currently open :term:`asynchronous generator` objects to
   close with an :meth:`~agen.aclose()` call.  After calling this method,
   the event loop will issue a warning if a new asynchronous generator
   is iterated. This should be used to reliably finalize all scheduled
   asynchronous generators.

   Note that there is no need to call this function when
   :func:`asyncio.run` is used.

   Example::

    try:
        loop.run_forever()
    finally:
        loop.run_until_complete(loop.shutdown_asyncgens())
        loop.close()

   .. versionadded:: 3.6
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 181)

Unknown directive type "coroutinemethod".

```
.. coroutinemethod:: loop.shutdown_default_executor()

   Schedule the closure of the default executor and wait for it to join all of
   the threads in the :class:`ThreadPoolExecutor`. After calling this method, a
   :exc:`RuntimeError` will be raised if :meth:`loop.run_in_executor` is called
   while using the default executor.

   Note that there is no need to call this function when
   :func:`asyncio.run` is used.

   .. versionadded:: 3.9
```

Scheduling callbacks

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 197)

Unknown directive type "method".

```
.. method:: loop.call_soon(callback, *args, context=None)

   Schedule the *callback* :term:`callback` to be called with
   *args* arguments at the next iteration of the event loop.

   Callbacks are called in the order in which they are registered.
   Each callback will be called exactly once.

   An optional keyword-only *context* argument allows specifying a
   custom :class:`contextvars.Context` for the *callback* to run in.
   The current context is used when no *context* is provided.
```

```
An instance of :class:`asyncio.Handle` is returned, which can be
used later to cancel the callback.

This method is not thread-safe.
```

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

Unknown directive type "method".

```
.. method:: loop.call_soon_threadsafe(callback, *args, context=None)

   A thread-safe variant of :meth:`call_soon`.  Must be used to
   schedule callbacks *from another thread*.

   Raises :exc:`RuntimeError` if called on a loop that's been closed.
   This can happen on a secondary thread when the main application is
   shutting down.

   See the :ref:`concurrency and multithreading <asyncio-multithreading>`
   section of the documentation.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 226)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.7
   The *context* keyword-only parameter was added. See :pep:`567`
   for more details.
```

Note

Most :mod:`asyncio` scheduling functions don't allow passing keyword arguments. To do that, use :func:`functools.partial`:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 234); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 234); *backlink*

Unknown interpreted text role "func".

```
# will schedule "print("Hello", flush=True)"
loop.call_soon(
    functools.partial(print, "Hello", flush=True))
```

Using partial objects is usually more convenient than using lambdas, as asyncio can render partial objects better in debug and error messages.

Scheduling delayed callbacks

Event loop provides mechanisms to schedule callback functions to be called at some point in the future. Event loop uses monotonic clocks to track time.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 256)

Unknown directive type "method".

```
.. method:: loop.call_later(delay, callback, *args, context=None)

   Schedule *callback* to be called after the given *delay*
   number of seconds (can be either an int or a float).
```

```
An instance of :class:`asyncio.TimerHandle` is returned which can
be used to cancel the callback.

*callback* will be called exactly once.  If two callbacks are
scheduled for exactly the same time, the order in which they
are called is undefined.

The optional positional *args* will be passed to the callback when
it is called. If you want the callback to be called with keyword
arguments use :func:`functools.partial`.

An optional keyword-only *context* argument allows specifying a
custom :class:`contextvars.Context` for the *callback* to run in.
The current context is used when no *context* is provided.

.. versionchanged:: 3.7
   The *context* keyword-only parameter was added. See :pep:`567`
   for more details.

.. versionchanged:: 3.8
   In Python 3.7 and earlier with the default event loop implementation,
   the *delay* could not exceed one day.
   This has been fixed in Python 3.8.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 285)

Unknown directive type "method".

```
.. method:: loop.call_at(when, callback, *args, context=None)

   Schedule *callback* to be called at the given absolute timestamp
   *when* (an int or a float), using the same time reference as
   :meth:`loop.time`.

   This method's behavior is the same as :meth:`call_later`.

   An instance of :class:`asyncio.TimerHandle` is returned which can
   be used to cancel the callback.

   .. versionchanged:: 3.7
      The *context* keyword-only parameter was added. See :pep:`567`
      for more details.

   .. versionchanged:: 3.8
      In Python 3.7 and earlier with the default event loop implementation,
      the difference between *when* and the current time could not exceed
      one day.  This has been fixed in Python 3.8.
```

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

Unknown directive type "method".

```
.. method:: loop.time()

   Return the current time, as a :class:`float` value, according to
   the event loop's internal monotonic clock.
```

Note

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 311)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.8
   In Python 3.7 and earlier timeouts (relative *delay* or absolute *when*)
   should not exceed one day.  This has been fixed in Python 3.8.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 315)

Unknown directive type "seealso".

```
.. seealso::

   The :func:`asyncio.sleep` function.
```

Creating Futures and Tasks

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

Unknown directive type "method".

```
.. method:: loop.create_future()

   Create an :class:`asyncio.Future` object attached to the event loop.

   This is the preferred way to create Futures in asyncio. This lets
   third-party event loops provide alternative implementations of
   the Future object (with better performance or instrumentation).

   .. versionadded:: 3.5.2
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 333)

Unknown directive type "method".

```
.. method:: loop.create_task(coro, *, name=None, context=None)

   Schedule the execution of a :ref:`coroutine`.
   Return a :class:`Task` object.

   Third-party event loops can use their own subclass of :class:`Task`
   for interoperability. In this case, the result type is a subclass
   of :class:`Task`.

   If the *name* argument is provided and not ``None``, it is set as
   the name of the task using :meth:`Task.set_name`.

   An optional keyword-only *context* argument allows specifying a
   custom :class:`contextvars.Context` for the *coro* to run in.
   The current context copy is created when no *context* is provided.

   .. versionchanged:: 3.8
      Added the *name* parameter.

   .. versionchanged:: 3.11
      Added the *context* parameter.
```

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

Unknown directive type "method".

```
.. method:: loop.set_task_factory(factory)

   Set a task factory that will be used by
   :meth:`loop.create_task`.

   If *factory* is ``None`` the default task factory will be set.
   Otherwise, *factory* must be a *callable* with the signature matching
   ``(loop, coro, context=None)``, where *loop* is a reference to the active
   event loop, and *coro* is a coroutine object.  The callable
   must return a :class:`asyncio.Future`-compatible object.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 366)

Unknown directive type "method".

```
.. method:: loop.get_task_factory()

   Return a task factory or ``None`` if the default one is in use.
```

Opening network connections

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 374)

Unknown directive type "coroutinemethod".

```
.. coroutinemethod:: loop.create_connection(protocol_factory, \
                        host=None, port=None, *, ssl=None, \
                        family=0, proto=0, flags=0, sock=None, \
                        local_addr=None, server_hostname=None, \
                        ssl_handshake_timeout=None, \
                        ssl_shutdown_timeout=None, \
                        happy_eyeballs_delay=None, interleave=None)

   Open a streaming transport connection to a given
   address specified by *host* and *port*.

   The socket family can be either :py:data:`~socket.AF_INET` or
   :py:data:`~socket.AF_INET6` depending on *host* (or the *family*
   argument, if provided).

   The socket type will be :py:data:`~socket.SOCK_STREAM`.

   *protocol_factory* must be a callable returning an
   :ref:`asyncio protocol <asyncio-protocol>` implementation.

   This method will try to establish the connection in the background.
   When successful, it returns a ``(transport, protocol)`` pair.

   The chronological synopsis of the underlying operation is as follows:

   #. The connection is established and a :ref:`transport <asyncio-transport>`
      is created for it.

   #. *protocol_factory* is called without arguments and is expected to
      return a :ref:`protocol <asyncio-protocol>` instance.

   #. The protocol instance is coupled with the transport by calling its
      :meth:`~BaseProtocol.connection_made` method.

   #. A ``(transport, protocol)`` tuple is returned on success.

   The created transport is an implementation-dependent bidirectional
   stream.

   Other arguments:

   * *ssl*: if given and not false, a SSL/TLS transport is created
     (by default a plain TCP transport is created).  If *ssl* is
     a :class:`ssl.SSLContext` object, this context is used to create
     the transport; if *ssl* is :const:`True`, a default context returned
     from :func:`ssl.create_default_context` is used.

     .. seealso:: :ref:`SSL/TLS security considerations <ssl-security>`

   * *server_hostname* sets or overrides the hostname that the target
     server's certificate will be matched against.  Should only be passed
     if *ssl* is not ``None``.  By default the value of the *host* argument
     is used.  If *host* is empty, there is no default and you must pass a
     value for *server_hostname*.  If *server_hostname* is an empty
     string, hostname matching is disabled (which is a serious security
     risk, allowing for potential man-in-the-middle attacks).

   * *family*, *proto*, *flags* are the optional address family, protocol
     and flags to be passed through to getaddrinfo() for *host* resolution.
     If given, these should all be integers from the corresponding
     :mod:`socket` module constants.

   * *happy_eyeballs_delay*, if given, enables Happy Eyeballs for this
     connection. It should
     be a floating-point number representing the amount of time in seconds
```

```
      to wait for a connection attempt to complete, before starting the next
      attempt in parallel. This is the "Connection Attempt Delay" as defined
      in :rfc:`8305`. A sensible default value recommended by the RFC is ``0.25``
      (250 milliseconds).

   * *interleave* controls address reordering when a host name resolves to
      multiple IP addresses.
      If ``0`` or unspecified, no reordering is done, and addresses are
      tried in the order returned by :meth:`getaddrinfo`. If a positive integer
      is specified, the addresses are interleaved by address family, and the
      given integer is interpreted as "First Address Family Count" as defined
      in :rfc:`8305`. The default is ``0`` if *happy_eyeballs_delay* is not
      specified, and ``1`` if it is.

   * *sock*, if given, should be an existing, already connected
      :class:`socket.socket` object to be used by the transport.
      If *sock* is given, none of *host*, *port*, *family*, *proto*, *flags*,
      *happy_eyeballs_delay*, *interleave*
      and *local_addr* should be specified.

   * *local_addr*, if given, is a ``(local_host, local_port)`` tuple used
      to bind the socket locally.  The *local_host* and *local_port*
      are looked up using ``getaddrinfo()``, similarly to *host* and *port*.

   * *ssl_handshake_timeout* is (for a TLS connection) the time in seconds
      to wait for the TLS handshake to complete before aborting the connection.
      ``60.0`` seconds if ``None`` (default).

   * *ssl_shutdown_timeout* is the time in seconds to wait for the SSL shutdown
      to complete before aborting the connection. ``30.0`` seconds if ``None``
      (default).

   .. versionchanged:: 3.5

      Added support for SSL/TLS in :class:`ProactorEventLoop`.

   .. versionchanged:: 3.6

      The socket option :py:data:`~socket.TCP_NODELAY` is set by default
      for all TCP connections.

   .. versionchanged:: 3.7

      Added the *ssl_handshake_timeout* parameter.

   .. versionchanged:: 3.8

      Added the *happy_eyeballs_delay* and *interleave* parameters.

      Happy Eyeballs Algorithm: Success with Dual-Stack Hosts.
      When a server's IPv4 path and protocol are working, but the server's
      IPv6 path and protocol are not working, a dual-stack client
      application experiences significant connection delay compared to an
      IPv4-only client.  This is undesirable because it causes the dual-
      stack client to have a worse user experience.  This document
      specifies requirements for algorithms that reduce this user-visible
      delay and provides an algorithm.

      For more information: https://tools.ietf.org/html/rfc6555

   .. versionchanged:: 3.11

      Added the *ssl_shutdown_timeout* parameter.

   .. seealso::

      The :func:`open_connection` function is a high-level alternative
      API.  It returns a pair of (:class:`StreamReader`, :class:`StreamWriter`)
      that can be used directly in async/await code.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 509)

Unknown directive type "coroutinemethod".

```
.. coroutinemethod:: loop.create_datagram_endpoint(protocol_factory, \
                       local_addr=None, remote_addr=None, *, \
                       family=0, proto=0, flags=0, \
                       reuse_port=None, \
```

```
                        allow_broadcast=None, sock=None)

    Create a datagram connection.

    The socket family can be either :py:data:`~socket.AF_INET`,
    :py:data:`~socket.AF_INET6`, or :py:data:`~socket.AF_UNIX`,
    depending on *host* (or the *family* argument, if provided).

    The socket type will be :py:data:`~socket.SOCK_DGRAM`.

    *protocol_factory* must be a callable returning a
    :ref:`protocol <asyncio-protocol>` implementation.

    A tuple of ``(transport, protocol)`` is returned on success.

    Other arguments:

    * *local_addr*, if given, is a ``(local_host, local_port)`` tuple used
      to bind the socket locally.  The *local_host* and *local_port*
      are looked up using :meth:`getaddrinfo`.

    * *remote_addr*, if given, is a ``(remote_host, remote_port)`` tuple used
      to connect the socket to a remote address.  The *remote_host* and
      *remote_port* are looked up using :meth:`getaddrinfo`.

    * *family*, *proto*, *flags* are the optional address family, protocol
      and flags to be passed through to :meth:`getaddrinfo` for *host*
      resolution. If given, these should all be integers from the
      corresponding :mod:`socket` module constants.

    * *reuse_port* tells the kernel to allow this endpoint to be bound to the
      same port as other existing endpoints are bound to, so long as they all
      set this flag when being created. This option is not supported on Windows
      and some Unixes. If the :py:data:`~socket.SO_REUSEPORT` constant is not
      defined then this capability is unsupported.

    * *allow_broadcast* tells the kernel to allow this endpoint to send
      messages to the broadcast address.

    * *sock* can optionally be specified in order to use a preexisting,
      already connected, :class:`socket.socket` object to be used by the
      transport. If specified, *local_addr* and *remote_addr* should be omitted
      (must be :const:`None`).

    See :ref:`UDP echo client protocol <asyncio-udp-echo-client-protocol>` and
    :ref:`UDP echo server protocol <asyncio-udp-echo-server-protocol>` examples.

    .. versionchanged:: 3.4.4
       The *family*, *proto*, *flags*, *reuse_address*, *reuse_port*,
       *allow_broadcast*, and *sock* parameters were added.

    .. versionchanged:: 3.8.1
       The *reuse_address* parameter is no longer supported, as using
       :py:data:`~sockets.SO_REUSEADDR` poses a significant security concern for
       UDP. Explicitly passing ``reuse_address=True`` will raise an exception.

       When multiple processes with differing UIDs assign sockets to an
       identical UDP socket address with ``SO_REUSEADDR``, incoming packets can
       become randomly distributed among the sockets.

       For supported platforms, *reuse_port* can be used as a replacement for
       similar functionality. With *reuse_port*,
       :py:data:`~sockets.SO_REUSEPORT` is used instead, which specifically
       prevents processes with differing UIDs from assigning sockets to the same
       socket address.

    .. versionchanged:: 3.8
       Added support for Windows.

    .. versionchanged:: 3.11
       The *reuse_address* parameter, disabled since Python 3.9.0, 3.8.1,
       3.7.6 and 3.6.10, has been entirely removed.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 586)

Unknown directive type "coroutinemethod".

```
.. coroutinemethod:: loop.create_unix_connection(protocol_factory, \
```

```
                        path=None, *, ssl=None, sock=None, \
                        server_hostname=None, ssl_handshake_timeout=None, \
                        ssl_shutdown_timeout=None)

   Create a Unix connection.

   The socket family will be :py:data:`~socket.AF_UNIX`; socket
   type will be :py:data:`~socket.SOCK_STREAM`.

   A tuple of ``(transport, protocol)`` is returned on success.

   *path* is the name of a Unix domain socket and is required,
   unless a *sock* parameter is specified.  Abstract Unix sockets,
   :class:`str`, :class:`bytes`, and :class:`~pathlib.Path` paths are
   supported.

   See the documentation of the :meth:`loop.create_connection` method
   for information about arguments to this method.

   .. availability:: Unix.

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

   .. versionchanged:: 3.11

      Added the *ssl_shutdown_timeout* parameter.
```

Creating network servers

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 620)

Unknown directive type "coroutinemethod".

```
.. coroutinemethod:: loop.create_server(protocol_factory, \
                        host=None, port=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, \
                        ssl_shutdown_timeout=None, \
                        start_serving=True)

   Create a TCP server (socket type :data:`~socket.SOCK_STREAM`) listening
   on *port* of the *host* address.

   Returns a :class:`Server` object.

   Arguments:

   * *protocol_factory* must be a callable returning a
     :ref:`protocol <asyncio-protocol>` implementation.

   * The *host* parameter can be set to several types which determine where
     the server would be listening:

     - If *host* is a string, the TCP server is bound to a single network
       interface specified by *host*.

     - If *host* is a sequence of strings, the TCP server is bound to all
       network interfaces specified by the sequence.

     - If *host* is an empty string or ``None``, all interfaces are
       assumed and a list of multiple sockets will be returned (most likely
       one for IPv4 and another one for IPv6).

   * The *port* parameter can be set to specify which port the server should
     listen on. If ``0`` or ``None`` (the default), a random unused port will
     be selected (note that if *host* resolves to multiple network interfaces,
     a different random port will be selected for each interface).

   * *family* can be set to either :data:`socket.AF_INET` or
     :data:`~socket.AF_INET6` to force the socket to use IPv4 or IPv6.
     If not set, the *family* will be determined from host name
     (defaults to :data:`~socket.AF_UNSPEC`).
```

```
   * *flags* is a bitmask for :meth:`getaddrinfo`.

   * *sock* can optionally be specified in order to use a preexisting
     socket object. If specified, *host* and *port* must not be specified.

   * *backlog* is the maximum number of queued connections passed to
     :meth:`~socket.socket.listen` (defaults to 100).

   * *ssl* can be set to an :class:`~ssl.SSLContext` instance to enable
     TLS over the accepted connections.

   * *reuse_address* tells the kernel to reuse a local socket in
     ``TIME_WAIT`` state, without waiting for its natural timeout to
     expire. If not specified will automatically be set to ``True`` on
     Unix.

   * *reuse_port* tells the kernel to allow this endpoint to be bound to the
     same port as other existing endpoints are bound to, so long as they all
     set this flag when being created. This option is not supported on
     Windows.

   * *ssl_handshake_timeout* is (for a TLS server) the time in seconds to wait
     for the TLS handshake to complete before aborting the connection.
     ``60.0`` seconds if ``None`` (default).

   * *ssl_shutdown_timeout* is the time in seconds to wait for the SSL shutdown
     to complete before aborting the connection. ``30.0`` seconds if ``None``
     (default).

   * *start_serving* set to ``True`` (the default) causes the created server
     to start accepting connections immediately.  When set to ``False``,
     the user should await on :meth:`Server.start_serving` or
     :meth:`Server.serve_forever` to make the server to start accepting
     connections.

   .. versionchanged:: 3.5

      Added support for SSL/TLS in :class:`ProactorEventLoop`.

   .. versionchanged:: 3.5.1

      The *host* parameter can be a sequence of strings.

   .. versionchanged:: 3.6

      Added *ssl_handshake_timeout* and *start_serving* parameters.
      The socket option :py:data:`~socket.TCP_NODELAY` is set by default
      for all TCP connections.

   .. versionchanged:: 3.11

      Added the *ssl_shutdown_timeout* parameter.

   .. seealso::

      The :func:`start_server` function is a higher-level alternative API
      that returns a pair of :class:`StreamReader` and :class:`StreamWriter`
      that can be used in an async/await code.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 723)

Unknown directive type "coroutinemethod".

```
.. coroutinemethod:: loop.create_unix_server(protocol_factory, path=None, \
                         *, sock=None, backlog=100, ssl=None, \
                         ssl_handshake_timeout=None, \
                         ssl_shutdown_timeout=None, \
                         start_serving=True)

   Similar to :meth:`loop.create_server` but works with the
   :py:data:`~socket.AF_UNIX` socket family.

   *path* is the name of a Unix domain socket, and is required,
   unless a *sock* argument is provided.  Abstract Unix sockets,
   :class:`str`, :class:`bytes`, and :class:`~pathlib.Path` paths
   are supported.
```

```
   See the documentation of the :meth:`loop.create_server` method
   for information about arguments to this method.

   .. availability:: Unix.

   .. versionchanged:: 3.7

      Added the *ssl_handshake_timeout* and *start_serving* parameters.
      The *path* parameter can now be a :class:`~pathlib.Path` object.

   .. versionchanged:: 3.11

      Added the *ssl_shutdown_timeout* parameter.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 752)

Unknown directive type "coroutinemethod".

```
.. coroutinemethod:: loop.connect_accepted_socket(protocol_factory, \
                       sock, *, ssl=None, ssl_handshake_timeout=None, \
                       ssl_shutdown_timeout=None)

   Wrap an already accepted connection into a transport/protocol pair.

   This method can be used by servers that accept connections outside
   of asyncio but that use asyncio to handle them.

   Parameters:

   * *protocol_factory* must be a callable returning a
     :ref:`protocol <asyncio-protocol>` implementation.

   * *sock* is a preexisting socket object returned from
     :meth:`socket.accept <socket.socket.accept>`.

   * *ssl* can be set to an :class:`~ssl.SSLContext` to enable SSL over
     the accepted connections.

   * *ssl_handshake_timeout* is (for an SSL connection) the time in seconds to
     wait for the SSL handshake to complete before aborting the connection.
     ``60.0`` seconds if ``None`` (default).

   * *ssl_shutdown_timeout* is the time in seconds to wait for the SSL shutdown
     to complete before aborting the connection. ``30.0`` seconds if ``None``
     (default).

   Returns a ``(transport, protocol)`` pair.

   .. versionadded:: 3.5.3

   .. versionchanged:: 3.7

      Added the *ssl_handshake_timeout* parameter.

   .. versionchanged:: 3.11

      Added the *ssl_shutdown_timeout* parameter.
```

Transferring files

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 796)

Unknown directive type "coroutinemethod".

```
.. coroutinemethod:: loop.sendfile(transport, file, \
                                   offset=0, count=None, *, fallback=True)

   Send a *file* over a *transport*.  Return the total number of bytes
   sent.

   The method uses high-performance :meth:`os.sendfile` if available.
```

```
   *file* must be a regular file object opened in binary mode.

   *offset* tells from where to start reading the file. If specified,
   *count* is the total number of bytes to transmit as opposed to
   sending the file until EOF is reached. File position is always updated,
   even when this method raises an error, and
   :meth:`file.tell() <io.IOBase.tell>` can be used to obtain the actual
   number of bytes sent.

   *fallback* set to ``True`` makes asyncio to manually read and send
   the file when the platform does not support the sendfile system call
   (e.g. Windows or SSL socket on Unix).

   Raise :exc:`SendfileNotAvailableError` if the system does not support
   the *sendfile* syscall and *fallback* is ``False``.

   .. versionadded:: 3.7
```

TLS Upgrade

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 826)

Unknown directive type "coroutinemethod".

```
.. coroutinemethod:: loop.start_tls(transport, protocol, \
                       sslcontext, *, server_side=False, \
                       server_hostname=None, ssl_handshake_timeout=None, \
                       ssl_shutdown_timeout=None)

   Upgrade an existing transport-based connection to TLS.

   Return a new transport instance, that the *protocol* must start using
   immediately after the *await*.  The *transport* instance passed to
   the *start_tls* method should never be used again.

   Parameters:

   * *transport* and *protocol* instances that methods like
     :meth:`~loop.create_server` and
     :meth:`~loop.create_connection` return.

   * *sslcontext*: a configured instance of :class:`~ssl.SSLContext`.

   * *server_side* pass ``True`` when a server-side connection is being
     upgraded (like the one created by :meth:`~loop.create_server`).

   * *server_hostname*: sets or overrides the host name that the target
     server's certificate will be matched against.

   * *ssl_handshake_timeout* is (for a TLS connection) the time in seconds to
     wait for the TLS handshake to complete before aborting the connection.
     ``60.0`` seconds if ``None`` (default).

   * *ssl_shutdown_timeout* is the time in seconds to wait for the SSL shutdown
     to complete before aborting the connection. ``30.0`` seconds if ``None``
     (default).

   .. versionadded:: 3.7

   .. versionchanged:: 3.11

      Added the *ssl_shutdown_timeout* parameter.
```

Watching file descriptors

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 870)

Unknown directive type "method".

```
.. method:: loop.add_reader(fd, callback, *args)
```

```
Start monitoring the *fd* file descriptor for read availability and
invoke *callback* with the specified arguments once *fd* is available for
reading.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 876)

Unknown directive type "method".

```
.. method:: loop.remove_reader(fd)

   Stop monitoring the *fd* file descriptor for read availability.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 880)

Unknown directive type "method".

```
.. method:: loop.add_writer(fd, callback, *args)

   Start monitoring the *fd* file descriptor for write availability and
   invoke *callback* with the specified arguments once *fd* is available for
   writing.

   Use :func:`functools.partial` :ref:`to pass keyword arguments
   <asyncio-pass-keywords>` to *callback*.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 889)

Unknown directive type "method".

```
.. method:: loop.remove_writer(fd)

   Stop monitoring the *fd* file descriptor for write availability.
```

See also :ref:`Platform Support <asyncio-platform-support>` section for some limitations of these methods.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 893); *backlink*

Unknown interpreted text role "ref".

Working with socket objects directly

In general, protocol implementations that use transport-based APIs such as :meth:`loop.create_connection` and :meth:`loop.create_server` are faster than implementations that work with sockets directly. However, there are some use cases when performance is not critical, and working with :class:`~socket.socket` objects directly is more convenient.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 900); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 900); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 900); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 907)

Unknown directive type "coroutinemethod".

```
.. coroutinemethod:: loop.sock_recv(sock, nbytes)

   Receive up to *nbytes* from *sock*.  Asynchronous version of
   :meth:`socket.recv() <socket.socket.recv>`.

   Return the received data as a bytes object.

   *sock* must be a non-blocking socket.

   .. versionchanged:: 3.7
      Even though this method was always documented as a coroutine
      method, releases before Python 3.7 returned a :class:`Future`.
      Since Python 3.7 this is an ``async def`` method.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 921)

Unknown directive type "coroutinemethod".

```
.. coroutinemethod:: loop.sock_recv_into(sock, buf)

   Receive data from *sock* into the *buf* buffer.  Modeled after the blocking
   :meth:`socket.recv_into() <socket.socket.recv_into>` method.

   Return the number of bytes written to the buffer.

   *sock* must be a non-blocking socket.

   .. versionadded:: 3.7
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 932)

Unknown directive type "coroutinemethod".

```
.. coroutinemethod:: loop.sock_recvfrom(sock, bufsize)

   Receive a datagram of up to *bufsize* from *sock*.  Asynchronous version of
   :meth:`socket.recvfrom() <socket.socket.recvfrom>`.

   Return a tuple of (received data, remote address).

   *sock* must be a non-blocking socket.

   .. versionadded:: 3.11
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 943)

Unknown directive type "coroutinemethod".

```
.. coroutinemethod:: loop.sock_recvfrom_into(sock, buf, nbytes=0)

   Receive a datagram of up to *nbytes* from *sock* into *buf*.
   Asynchronous version of
   :meth:`socket.recvfrom_into() <socket.socket.recvfrom_into>`.

   Return a tuple of (number of bytes received, remote address).

   *sock* must be a non-blocking socket.

   .. versionadded:: 3.11
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 955)

Unknown directive type "coroutinemethod".

```
.. coroutinemethod:: loop.sock_sendall(sock, data)

   Send *data* to the *sock* socket. Asynchronous version of
```

```
   :meth:`socket.sendall() <socket.socket.sendall>`.

   This method continues to send to the socket until either all data
   in *data* has been sent or an error occurs.  ``None`` is returned
   on success.  On error, an exception is raised. Additionally, there is no way
   to determine how much data, if any, was successfully processed by the
   receiving end of the connection.

   *sock* must be a non-blocking socket.

   .. versionchanged:: 3.7
      Even though the method was always documented as a coroutine
      method, before Python 3.7 it returned a :class:`Future`.
      Since Python 3.7, this is an ``async def`` method.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 973)

Unknown directive type "coroutinemethod".

```
.. coroutinemethod:: loop.sock_sendto(sock, data, address)

   Send a datagram from *sock* to *address*.
   Asynchronous version of
   :meth:`socket.sendto() <socket.socket.sendto>`.

   Return the number of bytes sent.

   *sock* must be a non-blocking socket.

   .. versionadded:: 3.11
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 985)

Unknown directive type "coroutinemethod".

```
.. coroutinemethod:: loop.sock_connect(sock, address)

   Connect *sock* to a remote socket at *address*.

   Asynchronous version of :meth:`socket.connect() <socket.socket.connect>`.

   *sock* must be a non-blocking socket.

   .. versionchanged:: 3.5.2
      ``address`` no longer needs to be resolved.  ``sock_connect``
      will try to check if the *address* is already resolved by calling
      :func:`socket.inet_pton`.  If not,
      :meth:`loop.getaddrinfo` will be used to resolve the
      *address*.

   .. seealso::

      :meth:`loop.create_connection`
      and  :func:`asyncio.open_connection() <open_connection>`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1006)

Unknown directive type "coroutinemethod".

```
.. coroutinemethod:: loop.sock_accept(sock)

   Accept a connection.  Modeled after the blocking
   :meth:`socket.accept() <socket.socket.accept>` method.

   The socket must be bound to an address and listening
   for connections. The return value is a pair ``(conn, address)`` where *conn*
   is a *new* socket object usable to send and receive data on the connection,
   and *address* is the address bound to the socket on the other end of the
   connection.

   *sock* must be a non-blocking socket.
```

```
   .. versionchanged:: 3.7
      Even though the method was always documented as a coroutine
      method, before Python 3.7 it returned a :class:`Future`.
      Since Python 3.7, this is an ``async def`` method.

   .. seealso::

      :meth:`loop.create_server` and :func:`start_server`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1028)

Unknown directive type "coroutinemethod".

```
.. coroutinemethod:: loop.sock_sendfile(sock, file, offset=0, count=None, \
                                        *, fallback=True)

   Send a file using high-performance :mod:`os.sendfile` if possible.
   Return the total number of bytes sent.

   Asynchronous version of :meth:`socket.sendfile() <socket.socket.sendfile>`.

   *sock* must be a non-blocking :const:`socket.SOCK_STREAM`
   :class:`~socket.socket`.

   *file* must be a regular file object open in binary mode.

   *offset* tells from where to start reading the file. If specified,
   *count* is the total number of bytes to transmit as opposed to
   sending the file until EOF is reached. File position is always updated,
   even when this method raises an error, and
   :meth:`file.tell() <io.IOBase.tell>` can be used to obtain the actual
   number of bytes sent.

   *fallback*, when set to ``True``, makes asyncio manually read and send
   the file when the platform does not support the sendfile syscall
   (e.g. Windows or SSL socket on Unix).

   Raise :exc:`SendfileNotAvailableError` if the system does not support
   *sendfile* syscall and *fallback* is ``False``.

   *sock* must be a non-blocking socket.

   .. versionadded:: 3.7
```

DNS

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1063)

Unknown directive type "coroutinemethod".

```
.. coroutinemethod:: loop.getaddrinfo(host, port, *, family=0, \
                        type=0, proto=0, flags=0)

   Asynchronous version of :meth:`socket.getaddrinfo`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1068)

Unknown directive type "coroutinemethod".

```
.. coroutinemethod:: loop.getnameinfo(sockaddr, flags=0)

   Asynchronous version of :meth:`socket.getnameinfo`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1072)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.7
   Both *getaddrinfo* and *getnameinfo* methods were always documented
   to return a coroutine, but prior to Python 3.7 they were, in fact,
   returning :class:`asyncio.Future` objects.  Starting with Python 3.7
   both methods are coroutines.
```

Working with pipes

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1082)

Unknown directive type "coroutinemethod".

```
.. coroutinemethod:: loop.connect_read_pipe(protocol_factory, pipe)

   Register the read end of *pipe* in the event loop.

   *protocol_factory* must be a callable returning an
   :ref:`asyncio protocol <asyncio-protocol>` implementation.

   *pipe* is a :term:`file-like object <file object>`.

   Return pair ``(transport, protocol)``, where *transport* supports
   the :class:`ReadTransport` interface and *protocol* is an object
   instantiated by the *protocol_factory*.

   With :class:`SelectorEventLoop` event loop, the *pipe* is set to
   non-blocking mode.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1098)

Unknown directive type "coroutinemethod".

```
.. coroutinemethod:: loop.connect_write_pipe(protocol_factory, pipe)

   Register the write end of *pipe* in the event loop.

   *protocol_factory* must be a callable returning an
   :ref:`asyncio protocol <asyncio-protocol>` implementation.

   *pipe* is :term:`file-like object <file object>`.

   Return pair ``(transport, protocol)``, where *transport* supports
   :class:`WriteTransport` interface and *protocol* is an object
   instantiated by the *protocol_factory*.

   With :class:`SelectorEventLoop` event loop, the *pipe* is set to
   non-blocking mode.
```

Note

:class:`SelectorEventLoop` does not support the above methods on Windows. Use :class:`ProactorEventLoop` instead for Windows.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1116); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1116); *backlink*

Unknown interpreted text role "class".

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

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1119)

Unknown directive type "seealso".

```
.. seealso::

   The :meth:`loop.subprocess_exec` and
   :meth:`loop.subprocess_shell` methods.
```

Unix signals

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1128)

Unknown directive type "method".

```
.. method:: loop.add_signal_handler(signum, callback, *args)

   Set *callback* as the handler for the *signum* signal.

   The callback will be invoked by *loop*, along with other queued callbacks
   and runnable coroutines of that event loop. Unlike signal handlers
   registered using :func:`signal.signal`, a callback registered with this
   function is allowed to interact with the event loop.

   Raise :exc:`ValueError` if the signal number is invalid or uncatchable.
   Raise :exc:`RuntimeError` if there is a problem setting up the handler.

   Use :func:`functools.partial` :ref:`to pass keyword arguments
   <asyncio-pass-keywords>` to *callback*.

   Like :func:`signal.signal`, this function must be invoked in the main
   thread.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1146)

Unknown directive type "method".

```
.. method:: loop.remove_signal_handler(sig)

   Remove the handler for the *sig* signal.

   Return ``True`` if the signal handler was removed, or ``False`` if
   no handler was set for the given signal.

   .. availability:: Unix.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1155)

Unknown directive type "seealso".

```
.. seealso::

   The :mod:`signal` module.
```

Executing code in thread or process pools

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1163)

Unknown directive type "awaitablemethod".

```
.. awaitablemethod:: loop.run_in_executor(executor, func, *args)

   Arrange for *func* to be called in the specified executor.

   The *executor* argument should be an :class:`concurrent.futures.Executor`
   instance. The default executor is used if *executor* is ``None``.
```

```
Example::

   import asyncio
   import concurrent.futures

   def blocking_io():
       # File operations (such as logging) can block the
       # event loop: run them in a thread pool.
       with open('/dev/urandom', 'rb') as f:
           return f.read(100)

   def cpu_bound():
       # CPU-bound operations will block the event loop:
       # in general it is preferable to run them in a
       # process pool.
       return sum(i * i for i in range(10 ** 7))

   async def main():
       loop = asyncio.get_running_loop()

       ## Options:

       # 1. Run in the default loop's executor:
       result = await loop.run_in_executor(
           None, blocking_io)
       print('default thread pool', result)

       # 2. Run in a custom thread pool:
       with concurrent.futures.ThreadPoolExecutor() as pool:
           result = await loop.run_in_executor(
               pool, blocking_io)
           print('custom thread pool', result)

       # 3. Run in a custom process pool:
       with concurrent.futures.ProcessPoolExecutor() as pool:
           result = await loop.run_in_executor(
               pool, cpu_bound)
           print('custom process pool', result)

   asyncio.run(main())

This method returns a :class:`asyncio.Future` object.

Use :func:`functools.partial` :ref:`to pass keyword arguments
<asyncio-pass-keywords>` to *func*.

.. versionchanged:: 3.5.3
   :meth:`loop.run_in_executor` no longer configures the
   ``max_workers`` of the thread pool executor it creates, instead
   leaving it up to the thread pool executor
   (:class:`~concurrent.futures.ThreadPoolExecutor`) to set the
   default.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1223)

Unknown directive type "method".

```
.. method:: loop.set_default_executor(executor)

   Set *executor* as the default executor used by :meth:`run_in_executor`.
   *executor* must be an instance of
   :class:`~concurrent.futures.ThreadPoolExecutor`.

   .. versionchanged:: 3.11
      *executor* must be an instance of
      :class:`~concurrent.futures.ThreadPoolExecutor`.
```

Error Handling API

Allows customizing how exceptions are handled in the event loop.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1239)

Unknown directive type "method".

```
.. method:: loop.set_exception_handler(handler)

   Set *handler* as the new event loop exception handler.

   If *handler* is ``None``, the default exception handler will
   be set.  Otherwise, *handler* must be a callable with the signature
   matching ``(loop, context)``, where ``loop``
   is a reference to the active event loop, and ``context``
   is a ``dict`` object containing the details of the exception
   (see :meth:`call_exception_handler` documentation for details
   about context).
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1251)

Unknown directive type "method".

```
.. method:: loop.get_exception_handler()

   Return the current exception handler, or ``None`` if no custom
   exception handler was set.

   .. versionadded:: 3.5.2
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1258)

Unknown directive type "method".

```
.. method:: loop.default_exception_handler(context)

   Default exception handler.

   This is called when an exception occurs and no exception
   handler is set. This can be called by a custom exception
   handler that wants to defer to the default handler behavior.

   *context* parameter has the same meaning as in
   :meth:`call_exception_handler`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1269)

Unknown directive type "method".

```
.. method:: loop.call_exception_handler(context)

   Call the current event loop exception handler.

   *context* is a ``dict`` object containing the following keys
   (new keys may be introduced in future Python versions):

   * 'message': Error message;
   * 'exception' (optional): Exception object;
   * 'future' (optional): :class:`asyncio.Future` instance;
   * 'task' (optional): :class:`asyncio.Task` instance;
   * 'handle' (optional): :class:`asyncio.Handle` instance;
   * 'protocol' (optional): :ref:`Protocol <asyncio-protocol>` instance;
   * 'transport' (optional): :ref:`Transport <asyncio-transport>` instance;
   * 'socket' (optional): :class:`socket.socket` instance;
   * 'asyncgen' (optional): Asynchronous generator that caused
                            the exception.

   .. note::

      This method should not be overloaded in subclassed
      event loops.  For custom exception handling, use
      the :meth:`set_exception_handler()` method.
```

Enabling debug mode

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1296)

Unknown directive type "method".

```
.. method:: loop.get_debug()

   Get the debug mode (:class:`bool`) of the event loop.

   The default value is ``True`` if the environment variable
   :envvar:`PYTHONASYNCIODEBUG` is set to a non-empty string, ``False``
   otherwise.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1304)

Unknown directive type "method".

```
.. method:: loop.set_debug(enabled: bool)

   Set the debug mode of the event loop.

   .. versionchanged:: 3.7

      The new :ref:`Python Development Mode <devmode>` can now also be used
      to enable the debug mode.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1313)

Unknown directive type "seealso".

```
.. seealso::

   The :ref:`debug mode of asyncio <asyncio-debug-mode>`.
```

Running Subprocesses

Methods described in this subsections are low-level. In regular async/await code consider using the high-level :func:`asyncio.create_subprocess_shell` and :func:`asyncio.create_subprocess_exec` convenience functions instead.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1321); *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-eventloop.rst, line 1321); *backlink*

Unknown interpreted text role "func".

Note

On Windows, the default event loop :class:`ProactorEventLoop` supports subprocesses, whereas :class:`SelectorEventLoop` does not. See :ref:`Subprocess Support on Windows <asyncio-windows-subprocess>` for details.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1328); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1328); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1328); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1333)

Unknown directive type "coroutinemethod".

```
.. coroutinemethod:: loop.subprocess_exec(protocol_factory, *args, \
                     stdin=subprocess.PIPE, stdout=subprocess.PIPE, \
                     stderr=subprocess.PIPE, **kwargs)

   Create a subprocess from one or more string arguments specified by
   *args*.

   *args* must be a list of strings represented by:

   * :class:`str`;
   * or :class:`bytes`, encoded to the
     :ref:`filesystem encoding <filesystem-encoding>`.

   The first string specifies the program executable,
   and the remaining strings specify the arguments.  Together, string
   arguments form the ``argv`` of the program.

   This is similar to the standard library :class:`subprocess.Popen`
   class called with ``shell=False`` and the list of strings passed as
   the first argument; however, where :class:`~subprocess.Popen` takes
   a single argument which is list of strings, *subprocess_exec*
   takes multiple string arguments.

   The *protocol_factory* must be a callable returning a subclass of the
   :class:`asyncio.SubprocessProtocol` class.

   Other parameters:

   * *stdin* can be any of these:

     * a file-like object representing a pipe to be connected to the
       subprocess's standard input stream using
       :meth:`~loop.connect_write_pipe`
     * the :const:`subprocess.PIPE` constant (default) which will create a new
       pipe and connect it,
     * the value ``None`` which will make the subprocess inherit the file
       descriptor from this process
     * the :const:`subprocess.DEVNULL` constant which indicates that the
       special :data:`os.devnull` file will be used

   * *stdout* can be any of these:

     * a file-like object representing a pipe to be connected to the
       subprocess's standard output stream using
       :meth:`~loop.connect_write_pipe`
     * the :const:`subprocess.PIPE` constant (default) which will create a new
       pipe and connect it,
     * the value ``None`` which will make the subprocess inherit the file
       descriptor from this process
     * the :const:`subprocess.DEVNULL` constant which indicates that the
       special :data:`os.devnull` file will be used

   * *stderr* can be any of these:

     * a file-like object representing a pipe to be connected to the
       subprocess's standard error stream using
       :meth:`~loop.connect_write_pipe`
     * the :const:`subprocess.PIPE` constant (default) which will create a new
       pipe and connect it,
     * the value ``None`` which will make the subprocess inherit the file
       descriptor from this process
     * the :const:`subprocess.DEVNULL` constant which indicates that the
       special :data:`os.devnull` file will be used
     * the :const:`subprocess.STDOUT` constant which will connect the standard
```

```
        error stream to the process' standard output stream

    * All other keyword arguments are passed to :class:`subprocess.Popen`
      without interpretation, except for *bufsize*, *universal_newlines*,
      *shell*, *text*, *encoding* and *errors*, which should not be specified
      at all.

      The ``asyncio`` subprocess API does not support decoding the streams
      as text. :func:`bytes.decode` can be used to convert the bytes returned
      from the stream to text.

    See the constructor of the :class:`subprocess.Popen` class
    for documentation on other arguments.

    Returns a pair of ``(transport, protocol)``, where *transport*
    conforms to the :class:`asyncio.SubprocessTransport` base class and
    *protocol* is an object instantiated by the *protocol_factory*.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1415)

Unknown directive type "coroutinemethod".

```
.. coroutinemethod:: loop.subprocess_shell(protocol_factory, cmd, *, \
                        stdin=subprocess.PIPE, stdout=subprocess.PIPE, \
                        stderr=subprocess.PIPE, **kwargs)

   Create a subprocess from *cmd*, which can be a :class:`str` or a
   :class:`bytes` string encoded to the
   :ref:`filesystem encoding <filesystem-encoding>`,
   using the platform's "shell" syntax.

   This is similar to the standard library :class:`subprocess.Popen`
   class called with ``shell=True``.

   The *protocol_factory* must be a callable returning a subclass of the
   :class:`SubprocessProtocol` class.

   See :meth:`~loop.subprocess_exec` for more details about
   the remaining arguments.

   Returns a pair of ``(transport, protocol)``, where *transport*
   conforms to the :class:`SubprocessTransport` base class and
   *protocol* is an object instantiated by the *protocol_factory*.
```

Note

It is the application's responsibility to ensure that all whitespace and special characters are quoted appropriately to avoid shell injection vulnerabilities. The :func:`shlex.quote` function can be used to properly escape whitespace and special characters in strings that are going to be used to construct shell commands.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1438); *backlink*

Unknown interpreted text role "func".

Callback Handles

A callback wrapper object returned by :meth:`loop.call_soon`, :meth:`loop.call_soon_threadsafe`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1451); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1451); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1454)

Unknown directive type "method".

```
.. method:: cancel()

   Cancel the callback.  If the callback has already been canceled
   or executed, this method has no effect.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1459)

Unknown directive type "method".

```
.. method:: cancelled()

   Return ``True`` if the callback was cancelled.

   .. versionadded:: 3.7
```

A callback wrapper object returned by :meth:`loop.call_later`, and :meth:`loop.call_at`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1467); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1467); *backlink*

Unknown interpreted text role "meth".

This class is a subclass of :class:`Handle`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1470); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1472)

Unknown directive type "method".

```
.. method:: when()

   Return a scheduled callback time as :class:`float` seconds.

   The time is an absolute timestamp, using the same time
   reference as :meth:`loop.time`.

   .. versionadded:: 3.7
```

Server Objects

Server objects are created by :meth:`loop.create_server`, :meth:`loop.create_unix_server`, :func:`start_server`, and :func:`start_unix_server` functions.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1485); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1485); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1485); *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-eventloop.rst, line 1485); *backlink*

Unknown interpreted text role "func".

Do not instantiate the class directly.

Server objects are asynchronous context managers. When used in an `async with` statement, it's guaranteed that the Server object is closed and not accepting new connections when the `async with` statement is completed:

```
srv = await loop.create_server(...)

async with srv:
    # some code

# At this point, srv is closed and no longer accepts new connections.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1506)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.7
   Server object is an asynchronous context manager since Python 3.7.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1509)

Unknown directive type "method".

```
.. method:: close()

   Stop serving: close listening sockets and set the :attr:`sockets`
   attribute to ``None``.

   The sockets that represent existing incoming client connections
   are left open.

   The server is closed asynchronously, use the :meth:`wait_closed`
   coroutine to wait until the server is closed.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1520)

Unknown directive type "method".

```
.. method:: get_loop()

   Return the event loop associated with the server object.

   .. versionadded:: 3.7
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1526)

Unknown directive type "coroutinemethod".

```
.. coroutinemethod:: start_serving()

   Start accepting connections.

   This method is idempotent, so it can be called when
   the server is already serving.
```

```
   The *start_serving* keyword-only parameter to
   :meth:`loop.create_server` and
   :meth:`asyncio.start_server` allows creating a Server object
   that is not accepting connections initially.  In this case
   ``Server.start_serving()``, or :meth:`Server.serve_forever` can be used
   to make the Server start accepting connections.

   .. versionadded:: 3.7
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1542)

Unknown directive type "coroutinemethod".

```
.. coroutinemethod:: serve_forever()

   Start accepting connections until the coroutine is cancelled.
   Cancellation of ``serve_forever`` task causes the server
   to be closed.

   This method can be called if the server is already accepting
   connections.  Only one ``serve_forever`` task can exist per
   one *Server* object.

   Example::

       async def client_connected(reader, writer):
           # Communicate with the client with
           # reader/writer streams.  For example:
           await reader.readline()

       async def main(host, port):
           srv = await asyncio.start_server(
               client_connected, host, port)
           await srv.serve_forever()

       asyncio.run(main('127.0.0.1', 0))

   .. versionadded:: 3.7
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1568)

Unknown directive type "method".

```
.. method:: is_serving()

   Return ``True`` if the server is accepting new connections.

   .. versionadded:: 3.7
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1574)

Unknown directive type "coroutinemethod".

```
.. coroutinemethod:: wait_closed()

   Wait until the :meth:`close` method completes.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1578)

Unknown directive type "attribute".

```
.. attribute:: sockets

   List of :class:`socket.socket` objects the server is listening on.

   .. versionchanged:: 3.7
      Prior to Python 3.7 ``Server.sockets`` used to return an
      internal list of server sockets directly.  In 3.7 a copy
      of that list is returned.
```

Event Loop Implementations

asyncio ships with two different event loop implementations: :class:`SelectorEventLoop` and :class:`ProactorEventLoop`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1593); backlink

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1593); backlink

Unknown interpreted text role "class".

By default asyncio is configured to use :class:`SelectorEventLoop` on Unix and :class:`ProactorEventLoop` on Windows.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1596); backlink

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1596); backlink

Unknown interpreted text role "class".

An event loop based on the :mod:`selectors` module.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1602); backlink

Unknown interpreted text role "mod".

Uses the most efficient *selector* available for the given platform. It is also possible to manually configure the exact selector implementation to be used:

```
import asyncio
import selectors

selector = selectors.SelectSelector()
loop = asyncio.SelectorEventLoop(selector)
asyncio.set_event_loop(loop)
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1616)

Unknown directive type "availability".

```
.. availability:: Unix, Windows.
```

An event loop for Windows that uses "I/O Completion Ports" (IOCP).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1623)

Unknown directive type "availability".

```
.. availability:: Windows.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1625)

Unknown directive type "seealso".

```
.. seealso::
```

`MSDN documentation on I/O Completion Ports
<https://docs.microsoft.com/en-ca/windows/desktop/FileIO/i-o-completion-ports>`_.

Abstract base class for asyncio-compliant event loops.

The :ref:`Event Loop Methods <asyncio-event-loop>` section lists all methods that an alternative implementation of `AbstractEventLoop` should have defined.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1635); *backlink*

Unknown interpreted text role "ref".

Examples

Note that all examples in this section **purposefully** show how to use the low-level event loop APIs, such as :meth:`loop.run_forever` and :meth:`loop.call_soon`. Modern asyncio applications rarely need to be written this way; consider using the high-level functions like :func:`asyncio.run`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1643); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1643); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1643); *backlink*

Unknown interpreted text role "func".

Hello World with call_soon()

An example using the :meth:`loop.call_soon` method to schedule a callback. The callback displays `"Hello World"` and then stops the event loop:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1655); *backlink*

Unknown interpreted text role "meth".

```
import asyncio

def hello_world(loop):
    """A callback to print 'Hello World' and stop the event loop"""
    print('Hello World')
    loop.stop()

loop = asyncio.get_event_loop()

# Schedule a call to hello_world()
loop.call_soon(hello_world, loop)

# Blocking call interrupted by loop.stop()
try:
    loop.run_forever()
finally:
    loop.close()
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1677)

Unknown directive type "seealso".

```
.. seealso::

   A similar :ref:`Hello World <coroutine>`
```

```
        example created with a coroutine and the :func:`run` function.
```

Display the current date with call_later()

An example of a callback displaying the current date every second. The callback uses the :meth:`loop.call_later` method to reschedule itself after 5 seconds, and then stops the event loop:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1688); *backlink*

Unknown interpreted text role "meth".

```
import asyncio
import datetime

def display_date(end_time, loop):
    print(datetime.datetime.now())
    if (loop.time() + 1.0) < end_time:
        loop.call_later(1, display_date, end_time, loop)
    else:
        loop.stop()

loop = asyncio.get_event_loop()

# Schedule the first call to display_date()
end_time = loop.time() + 5.0
loop.call_soon(display_date, end_time, loop)

# Blocking call interrupted by loop.stop()
try:
    loop.run_forever()
finally:
    loop.close()
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1714)

Unknown directive type "seealso".

```
   .. seealso::

      A similar :ref:`current date <asyncio_example_sleep>` example
      created with a coroutine and the :func:`run` function.
```

Watch a file descriptor for read events

Wait until a file descriptor received some data using the :meth:`loop.add_reader` method and then close the event loop:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1725); *backlink*

Unknown interpreted text role "meth".

```
import asyncio
from socket import socketpair

# Create a pair of connected file descriptors
rsock, wsock = socketpair()

loop = asyncio.get_event_loop()

def reader():
    data = rsock.recv(100)
    print("Received:", data.decode())

    # We are done: unregister the file descriptor
    loop.remove_reader(rsock)

    # Stop the event loop
    loop.stop()

# Register the file descriptor for read event
```

```
loop.add_reader(rsock, reader)

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

try:
    # Run the event loop
    loop.run_forever()
finally:
    # We are done. Close sockets and the event loop.
    rsock.close()
    wsock.close()
    loop.close()
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1761)

Unknown directive type "seealso".

```
.. seealso::

   * A similar :ref:`example <asyncio_example_create_connection>`
     using transports, protocols, and the
     :meth:`loop.create_connection` method.

   * Another similar :ref:`example <asyncio_example_create_connection-streams>`
     using the high-level :func:`asyncio.open_connection` function
     and streams.
```

Set signal handlers for SIGINT and SIGTERM

(This `signals` example only works on Unix.)

Register handlers for signals :py:data:`SIGINT` and :py:data:`SIGTERM` using the :meth:`loop.add_signal_handler` method:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1779); *backlink*

Unknown interpreted text role "py:data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1779); *backlink*

Unknown interpreted text role "py:data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)asyncio-eventloop.rst, line 1779); *backlink*

Unknown interpreted text role "meth".

```
import asyncio
import functools
import os
import signal

def ask_exit(signame, loop):
    print("got signal %s: exit" % signame)
    loop.stop()

async def main():
    loop = asyncio.get_running_loop()

    for signame in {'SIGINT', 'SIGTERM'}:
        loop.add_signal_handler(
            getattr(signal, signame),
            functools.partial(ask_exit, signame, loop))

    await asyncio.sleep(3600)

print("Event loop running for 1 hour, press Ctrl+C to interrupt.")
print(f"pid {os.getpid()}: send SIGINT or SIGTERM to exit.")

asyncio.run(main())
```