Transports and Protocols

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

Unknown directive type "currentmodule".

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

Preface

Transports and Protocols are used by the **low-level** event loop APIs such as :meth:`loop.create_connection`. They use callback-based programming style and enable high-performance implementations of network or IPC protocols (e.g. HTTP).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 13); *backlink*

Unknown interpreted text role "meth".

Essentially, transports and protocols should only be used in libraries and frameworks and never in high-level asyncio applications.

This documentation page covers both Transports and Protocols.

Introduction

At the highest level, the transport is concerned with *how* bytes are transmitted, while the protocol determines *which* bytes to transmit (and to some extent when).

A different way of saying the same thing: a transport is an abstraction for a socket (or similar I/O endpoint) while a protocol is an abstraction for an application, from the transport's point of view.

Yet another view is the transport and protocol interfaces together define an abstract interface for using network I/O and interprocess I/O.

There is always a 1:1 relationship between transport and protocol objects: the protocol calls transport methods to send data, while the transport calls protocol methods to pass it data that has been received.

Most of connection oriented event loop methods (such as :meth:`loop.create_connection`) usually accept a *protocol_factory* argument used to create a *Protocol* object for an accepted connection, represented by a *Transport* object. Such methods usually return a tuple of `(transport, protocol)`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 44); *backlink*

Unknown interpreted text role "meth".

Contents

This documentation page contains the following sections:

- The Transports section documents asyncio :class:`BaseTransport`, :class:`ReadTransport`, :class:`WriteTransport`, :class:`Transport`, :class:`DatagramTransport`, and :class:`SubprocessTransport` classes.

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 54); *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-protocol.rst, line 54); *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-

protocol.rst, line 54); *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-protocol.rst, line 54); *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-protocol.rst, line 54); *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-protocol.rst, line 54); *backlink*

Unknown interpreted text role "class".

- The Protocols section documents asyncio :class:`BaseProtocol`, :class:`Protocol`, :class:`BufferedProtocol`, :class:`DatagramProtocol`, and :class:`SubprocessProtocol` classes.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 59); *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-protocol.rst, line 59); *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-protocol.rst, line 59); *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-protocol.rst, line 59); *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-protocol.rst, line 59); *backlink*

Unknown interpreted text role "class".

- The Examples section showcases how to work with transports, protocols, and low-level event loop APIs.

Transports

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

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 72); *backlink*

Unknown interpreted text role "source".

Transports are classes provided by :mod:`asyncio` in order to abstract various kinds of communication channels.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 76); *backlink***
>
> Unknown interpreted text role "mod".

Transport objects are always instantiated by an :ref:`asyncio event loop <asyncio-event-loop>`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 79); *backlink***
>
> Unknown interpreted text role "ref".

asyncio implements transports for TCP, UDP, SSL, and subprocess pipes. The methods available on a transport depend on the transport's kind.

The transport classes are :ref:`not thread safe <asyncio-multithreading>`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 85); *backlink***
>
> Unknown interpreted text role "ref".

Transports Hierarchy

Base class for all transports. Contains methods that all asyncio transports share.

A base transport for write-only connections.

Instances of the *WriteTransport* class are returned from the :meth:`loop.connect_write_pipe` event loop method and are also used by subprocess-related methods like :meth:`loop.subprocess_exec`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 100); *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-protocol.rst, line 100); *backlink***
>
> Unknown interpreted text role "meth".

A base transport for read-only connections.

Instances of the *ReadTransport* class are returned from the :meth:`loop.connect_read_pipe` event loop method and are also used by subprocess-related methods like :meth:`loop.subprocess_exec`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 109); *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-protocol.rst, line 109); *backlink***
>
> Unknown interpreted text role "meth".

Interface representing a bidirectional transport, such as a TCP connection.

The user does not instantiate a transport directly; they call a utility function, passing it a protocol factory and other information necessary to create the transport and protocol.

Instances of the *Transport* class are returned from or used by event loop methods like :meth:`loop.create_connection`, :meth:`loop.create_unix_connection`, :meth:`loop.create_server`, :meth:`loop.sendfile`, etc.

> **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-protocol.rst, line 123); *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-protocol.rst, line 123); *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-protocol.rst, line 123); *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-protocol.rst, line 123); *backlink*

Unknown interpreted text role "meth".

A transport for datagram (UDP) connections.

Instances of the *DatagramTransport* class are returned from the :meth:`loop.create_datagram_endpoint` event loop method.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 133); *backlink*

Unknown interpreted text role "meth".

An abstraction to represent a connection between a parent and its child OS process.

Instances of the *SubprocessTransport* class are returned from event loop methods :meth:`loop.subprocess_shell` and :meth:`loop.subprocess_exec`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 142); *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-protocol.rst, line 142); *backlink*

Unknown interpreted text role "meth".

Base Transport

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 150)

Unknown directive type "method".

```
.. method:: BaseTransport.close()

   Close the transport.

   If the transport has a buffer for outgoing
   data, buffered data will be flushed asynchronously.  No more data
   will be received.  After all buffered data is flushed, the
   protocol's :meth:`protocol.connection_lost()
   <BaseProtocol.connection_lost>` method will be called with
   :const:`None` as its argument.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 161)

Unknown directive type "method".

```
.. method:: BaseTransport.is_closing()

   Return ``True`` if the transport is closing or is closed.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 165)

Unknown directive type "method".

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

   Return information about the transport or underlying resources
   it uses.

   *name* is a string representing the piece of transport-specific
   information to get.

   *default* is the value to return if the information is not
   available, or if the transport does not support querying it
   with the given third-party event loop implementation or on the
   current platform.

   For example, the following code attempts to get the underlying
   socket object of the transport::

      sock = transport.get_extra_info('socket')
      if sock is not None:
          print(sock.getsockopt(...))

   Categories of information that can be queried on some transports:

   * socket:

     - ``'peername'``: the remote address to which the socket is
       connected, result of :meth:`socket.socket.getpeername`
       (``None`` on error)

     - ``'socket'``: :class:`socket.socket` instance

     - ``'sockname'``: the socket's own address,
       result of :meth:`socket.socket.getsockname`

   * SSL socket:

     - ``'compression'``: the compression algorithm being used as a
       string, or ``None`` if the connection isn't compressed; result
       of :meth:`ssl.SSLSocket.compression`

     - ``'cipher'``: a three-value tuple containing the name of the
       cipher being used, the version of the SSL protocol that defines
       its use, and the number of secret bits being used; result of
       :meth:`ssl.SSLSocket.cipher`

     - ``'peercert'``: peer certificate; result of
       :meth:`ssl.SSLSocket.getpeercert`

     - ``'sslcontext'``: :class:`ssl.SSLContext` instance

     - ``'ssl_object'``: :class:`ssl.SSLObject` or
       :class:`ssl.SSLSocket` instance

   * pipe:

     - ``'pipe'``: pipe object

   * subprocess:

     - ``'subprocess'``: :class:`subprocess.Popen` instance
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 225)

Unknown directive type "method".

```
.. method:: BaseTransport.set_protocol(protocol)

   Set a new protocol.

   Switching protocol should only be done when both
   protocols are documented to support the switch.
```

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

Unknown directive type "method".

```
.. method:: BaseTransport.get_protocol()

   Return the current protocol.
```

Read-only Transports

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 240)

Unknown directive type "method".

```
.. method:: ReadTransport.is_reading()

   Return ``True`` if the transport is receiving new data.

   .. versionadded:: 3.7
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 246)

Unknown directive type "method".

```
.. method:: ReadTransport.pause_reading()

   Pause the receiving end of the transport.  No data will be passed to
   the protocol's :meth:`protocol.data_received() <Protocol.data_received>`
   method until :meth:`resume_reading` is called.

   .. versionchanged:: 3.7
      The method is idempotent, i.e. it can be called when the
      transport is already paused or closed.
```

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

Unknown directive type "method".

```
.. method:: ReadTransport.resume_reading()

   Resume the receiving end.  The protocol's
   :meth:`protocol.data_received() <Protocol.data_received>` method
   will be called once again if some data is available for reading.

   .. versionchanged:: 3.7
      The method is idempotent, i.e. it can be called when the
      transport is already reading.
```

Write-only Transports

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 270)

Unknown directive type "method".

```
.. method:: WriteTransport.abort()

   Close the transport immediately, without waiting for pending operations
   to complete.  Buffered data will be lost.  No more data will be received.
   The protocol's :meth:`protocol.connection_lost()
   <BaseProtocol.connection_lost>` method will eventually be
   called with :const:`None` as its argument.
```

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

Unknown directive type "method".

```
.. method:: WriteTransport.can_write_eof()

   Return :const:`True` if the transport supports
   :meth:`~WriteTransport.write_eof`, :const:`False` if not.
```

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

Unknown directive type "method".

```
.. method:: WriteTransport.get_write_buffer_size()

   Return the current size of the output buffer used by the transport.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 287)

Unknown directive type "method".

```
.. method:: WriteTransport.get_write_buffer_limits()

   Get the *high* and *low* watermarks for write flow control. Return a
   tuple ``(low, high)`` where *low* and *high* are positive number of
   bytes.

   Use :meth:`set_write_buffer_limits` to set the limits.

   .. versionadded:: 3.4.2
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 297)

Unknown directive type "method".

```
.. method:: WriteTransport.set_write_buffer_limits(high=None, low=None)

   Set the *high* and *low* watermarks for write flow control.

   These two values (measured in number of
   bytes) control when the protocol's
   :meth:`protocol.pause_writing() <BaseProtocol.pause_writing>`
   and :meth:`protocol.resume_writing() <BaseProtocol.resume_writing>`
   methods are called. If specified, the low watermark must be less
   than or equal to the high watermark.  Neither *high* nor *low*
   can be negative.

   :meth:`~BaseProtocol.pause_writing` is called when the buffer size
   becomes greater than or equal to the *high* value. If writing has
   been paused, :meth:`~BaseProtocol.resume_writing` is called when
   the buffer size becomes less than or equal to the *low* value.

   The defaults are implementation-specific.  If only the
   high watermark is given, the low watermark defaults to an
   implementation-specific value less than or equal to the
   high watermark.  Setting *high* to zero forces *low* to zero as
   well, and causes :meth:`~BaseProtocol.pause_writing` to be called
   whenever the buffer becomes non-empty.  Setting *low* to zero causes
   :meth:`~BaseProtocol.resume_writing` to be called only once the
   buffer is empty. Use of zero for either limit is generally
   sub-optimal as it reduces opportunities for doing I/O and
   computation concurrently.

   Use :meth:`~WriteTransport.get_write_buffer_limits`
   to get the limits.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 328)

Unknown directive type "method".

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

   Write some *data* bytes to the transport.

   This method does not block; it buffers the data and arranges for it
   to be sent out asynchronously.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 335)

Unknown directive type "method".

```
.. method:: WriteTransport.writelines(list_of_data)

   Write a list (or any iterable) of data bytes to the transport.
   This is functionally equivalent to calling :meth:`write` on each
   element yielded by the iterable, but may be implemented more
   efficiently.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 342)

Unknown directive type "method".

```
.. method:: WriteTransport.write_eof()

   Close the write end of the transport after flushing all buffered data.
   Data may still be received.

   This method can raise :exc:`NotImplementedError` if the transport
   (e.g. SSL) doesn't support half-closed connections.
```

Datagram Transports

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 354)

Unknown directive type "method".

```
.. method:: DatagramTransport.sendto(data, addr=None)

   Send the *data* bytes to the remote peer given by *addr* (a
   transport-dependent target address).  If *addr* is :const:`None`,
   the data is sent to the target address given on transport
   creation.

   This method does not block; it buffers the data and arranges
   for it to be sent out asynchronously.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 364)

Unknown directive type "method".

```
.. method:: DatagramTransport.abort()

   Close the transport immediately, without waiting for pending
   operations to complete.  Buffered data will be lost.
   No more data will be received.  The protocol's
   :meth:`protocol.connection_lost() <BaseProtocol.connection_lost>`
   method will eventually be called with :const:`None` as its argument.
```

Subprocess Transports

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 378)

Unknown directive type "method".

```
.. method:: SubprocessTransport.get_pid()

   Return the subprocess process id as an integer.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 382)

Unknown directive type "method".

```
.. method:: SubprocessTransport.get_pipe_transport(fd)

   Return the transport for the communication pipe corresponding to the
   integer file descriptor *fd*:

   * ``0``: readable streaming transport of the standard input (*stdin*),
     or :const:`None` if the subprocess was not created with ``stdin=PIPE``
   * ``1``: writable streaming transport of the standard output (*stdout*),
     or :const:`None` if the subprocess was not created with ``stdout=PIPE``
   * ``2``: writable streaming transport of the standard error (*stderr*),
     or :const:`None` if the subprocess was not created with ``stderr=PIPE``
   * other *fd*: :const:`None`
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 395)

Unknown directive type "method".

```
.. method:: SubprocessTransport.get_returncode()

   Return the subprocess return code as an integer or :const:`None`
   if it hasn't returned, which is similar to the
   :attr:`subprocess.Popen.returncode` attribute.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 401)

Unknown directive type "method".

```
.. method:: SubprocessTransport.kill()

   Kill the subprocess.

   On POSIX systems, the function sends SIGKILL to the subprocess.
   On Windows, this method is an alias for :meth:`terminate`.

   See also :meth:`subprocess.Popen.kill`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 410)

Unknown directive type "method".

```
.. method:: SubprocessTransport.send_signal(signal)

   Send the *signal* number to the subprocess, as in
   :meth:`subprocess.Popen.send_signal`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 415)

Unknown directive type "method".

```
.. method:: SubprocessTransport.terminate()

   Stop the subprocess.

   On POSIX systems, this method sends SIGTERM to the subprocess.
   On Windows, the Windows API function TerminateProcess() is called to
   stop the subprocess.
```

```
   See also :meth:`subprocess.Popen.terminate`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 425)

Unknown directive type "method".

```
.. method:: SubprocessTransport.close()

   Kill the subprocess by calling the :meth:`kill` method.

   If the subprocess hasn't returned yet, and close transports of
   *stdin*, *stdout*, and *stderr* pipes.
```

Protocols

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

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 438); *backlink*

Unknown interpreted text role "source".

---

asyncio provides a set of abstract base classes that should be used to implement network protocols. Those classes are meant to be used together with :ref:`transports <asyncio-transport>`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 442); *backlink*

Unknown interpreted text role "ref".

Subclasses of abstract base protocol classes may implement some or all methods. All these methods are callbacks: they are called by transports on certain events, for example when some data is received. A base protocol method should be called by the corresponding transport.

Base Protocols

Base protocol with methods that all protocols share.

The base class for implementing streaming protocols (TCP, Unix sockets, etc).

A base class for implementing streaming protocols with manual control of the receive buffer.

The base class for implementing datagram (UDP) protocols.

The base class for implementing protocols communicating with child processes (unidirectional pipes).

Base Protocol

All asyncio protocols can implement Base Protocol callbacks.

Connection Callbacks

Connection callbacks are called on all protocols, exactly once per a successful connection. All other protocol callbacks can only be called between those two methods.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 490)

Unknown directive type "method".

```
.. method:: BaseProtocol.connection_made(transport)

   Called when a connection is made.

   The *transport* argument is the transport representing the
   connection.  The protocol is responsible for storing the reference
   to its transport.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 498)

Unknown directive type "method".

```
.. method:: BaseProtocol.connection_lost(exc)

   Called when the connection is lost or closed.

   The argument is either an exception object or :const:`None`.
   The latter means a regular EOF is received, or the connection was
   aborted or closed by this side of the connection.
```

Flow Control Callbacks

Flow control callbacks can be called by transports to pause or resume writing performed by the protocol.

See the documentation of the :meth:`~WriteTransport.set_write_buffer_limits` method for more details.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 512); *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-protocol.rst, line 515)

Unknown directive type "method".

```
.. method:: BaseProtocol.pause_writing()

   Called when the transport's buffer goes over the high watermark.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 519)

Unknown directive type "method".

```
.. method:: BaseProtocol.resume_writing()

   Called when the transport's buffer drains below the low watermark.
```

If the buffer size equals the high watermark, :meth:`~BaseProtocol.pause_writing` is not called: the buffer size must go strictly over.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 523); *backlink*

Unknown interpreted text role "meth".

Conversely, :meth:`~BaseProtocol.resume_writing` is called when the buffer size is equal or lower than the low watermark. These end conditions are important to ensure that things go as expected when either mark is zero.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 527); *backlink*

Unknown interpreted text role "meth".

Streaming Protocols

Event methods, such as :meth:`loop.create_server`, :meth:`loop.create_unix_server`, :meth:`loop.create_connection`, :meth:`loop.create_unix_connection`, :meth:`loop.connect_accepted_socket`, :meth:`loop.connect_read_pipe`, and :meth:`loop.connect_write_pipe` accept factories that return streaming protocols.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 536); *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-protocol.rst, line 536); *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-protocol.rst, line 536); *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-protocol.rst, line 536); *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-protocol.rst, line 536); *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-protocol.rst, line 536); *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-protocol.rst, line 542)

Unknown directive type "method".

```
.. method:: Protocol.data_received(data)

   Called when some data is received.  *data* is a non-empty bytes
   object containing the incoming data.

   Whether the data is buffered, chunked or reassembled depends on
   the transport.  In general, you shouldn't rely on specific semantics
   and instead make your parsing generic and flexible. However,
   data is always received in the correct order.

   The method can be called an arbitrary number of times while
   a connection is open.

   However, :meth:`protocol.eof_received() <Protocol.eof_received>`
   is called at most once.  Once `eof_received()` is called,
   ``data_received()`` is not called anymore.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 559)

Unknown directive type "method".

```
.. method:: Protocol.eof_received()

   Called when the other end signals it won't send any more data
   (for example by calling :meth:`transport.write_eof()
   <WriteTransport.write_eof>`, if the other end also uses
   asyncio).

   This method may return a false value (including ``None``), in which case
   the transport will close itself.  Conversely, if this method returns a
   true value, the protocol used determines whether to close the transport.
   Since the default implementation returns ``None``, it implicitly closes the
   connection.
```

```
Some transports, including SSL, don't support half-closed connections,
in which case returning true from this method will result in the connection
being closed.
```

State machine:

System Message: WARNING/2 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 579)

Cannot analyze code. No Pygments lexer found for "none".

```
.. code-block:: none

    start -> connection_made
        [-> data_received]*
        [-> eof_received]?
    -> connection_lost -> end
```

Buffered Streaming Protocols

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 590)

Unknown directive type "versionadded".

```
.. versionadded:: 3.7
```

Buffered Protocols can be used with any event loop method that supports Streaming Protocols.

`BufferedProtocol` implementations allow explicit manual allocation and control of the receive buffer. Event loops can then use the buffer provided by the protocol to avoid unnecessary data copies. This can result in noticeable performance improvement for protocols that receive big amounts of data. Sophisticated protocol implementations can significantly reduce the number of buffer allocations.

The following callbacks are called on :class:`BufferedProtocol` instances:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 602); *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-protocol.rst, line 605)

Unknown directive type "method".

```
.. method:: BufferedProtocol.get_buffer(sizehint)

   Called to allocate a new receive buffer.

   *sizehint* is the recommended minimum size for the returned
   buffer.  It is acceptable to return smaller or larger buffers
   than what *sizehint* suggests.  When set to -1, the buffer size
   can be arbitrary. It is an error to return a buffer with a zero size.

   ``get_buffer()`` must return an object implementing the
   :ref:`buffer protocol <bufferobjects>`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 617)

Unknown directive type "method".

```
.. method:: BufferedProtocol.buffer_updated(nbytes)

   Called when the buffer was updated with the received data.
```

```
*nbytes* is the total number of bytes that were written to the buffer.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst`, **line 623**)

Unknown directive type "method".

```
.. method:: BufferedProtocol.eof_received()

   See the documentation of the :meth:`protocol.eof_received()
   <Protocol.eof_received>` method.
```

:meth:`~BufferedProtocol.get_buffer` can be called an arbitrary number of times during a connection. However, :meth:`protocol.eof_received() <Protocol.eof_received>` is called at most once and, if called, :meth:`~BufferedProtocol.get_buffer` and :meth:`~BufferedProtocol.buffer_updated` won't be called after it.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst`, **line 629**); ***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-protocol.rst`, **line 629**); ***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-protocol.rst`, **line 629**); ***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-protocol.rst`, **line 629**); ***backlink***

Unknown interpreted text role "meth".

State machine:

System Message: WARNING/2 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst`, **line 637**)

Cannot analyze code. No Pygments lexer found for "none".

```
.. code-block:: none

    start -> connection_made
        [-> get_buffer
            [-> buffer_updated]?
        ]*
        [-> eof_received]?
    -> connection_lost -> end
```

Datagram Protocols

Datagram Protocol instances should be constructed by protocol factories passed to the :meth:`loop.create_datagram_endpoint` method.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst`, **line 650**); ***backlink***

Unknown interpreted text role "meth".

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-protocol.rst, line 653)

Unknown directive type "method".

```
.. method:: DatagramProtocol.datagram_received(data, addr)

   Called when a datagram is received.  *data* is a bytes object containing
   the incoming data.  *addr* is the address of the peer sending the data;
   the exact format depends on the transport.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 659)

Unknown directive type "method".

```
.. method:: DatagramProtocol.error_received(exc)

   Called when a previous send or receive operation raises an
   :class:`OSError`.  *exc* is the :class:`OSError` instance.

   This method is called in rare conditions, when the transport (e.g. UDP)
   detects that a datagram could not be delivered to its recipient.
   In many conditions though, undeliverable datagrams will be silently
   dropped.
```

Note

On BSD systems (macOS, FreeBSD, etc.) flow control is not supported for datagram protocols, because there is no reliable way to detect send failures caused by writing too many packets.

The socket always appears 'ready' and excess packets are dropped. An :class:`OSError` with `errno` set to :const:`errno.ENOBUFS` may or may not be raised; if it is raised, it will be reported to :meth:`DatagramProtocol.error_received` but otherwise ignored.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 675); *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-protocol.rst, line 675); *backlink*

Unknown interpreted text role "const".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 675); *backlink*

Unknown interpreted text role "meth".

Subprocess Protocols

Subprocess Protocol instances should be constructed by protocol factories passed to the :meth:`loop.subprocess_exec` and :meth:`loop.subprocess_shell` methods.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 686); *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-protocol.rst, line 686); *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-protocol.rst, line 690)

Unknown directive type "method".

```
.. method:: SubprocessProtocol.pipe_data_received(fd, data)

   Called when the child process writes data into its stdout or stderr
   pipe.

   *fd* is the integer file descriptor of the pipe.

   *data* is a non-empty bytes object containing the received data.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 699)

Unknown directive type "method".

```
.. method:: SubprocessProtocol.pipe_connection_lost(fd, exc)

   Called when one of the pipes communicating with the child process
   is closed.

   *fd* is the integer file descriptor that was closed.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 706)

Unknown directive type "method".

```
.. method:: SubprocessProtocol.process_exited()

   Called when the child process has exited.
```

Examples

TCP Echo Server

Create a TCP echo server using the :meth:`loop.create_server` method, send back received data, and close the connection:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 719); *backlink*

Unknown interpreted text role "meth".

```
import asyncio


class EchoServerProtocol(asyncio.Protocol):
    def connection_made(self, transport):
        peername = transport.get_extra_info('peername')
        print('Connection from {}'.format(peername))
        self.transport = transport

    def data_received(self, data):
        message = data.decode()
        print('Data received: {!r}'.format(message))

        print('Send: {!r}'.format(message))
        self.transport.write(data)

        print('Close the client socket')
        self.transport.close()


async def main():
    # Get a reference to the event loop as we plan to use
    # low-level APIs.
    loop = asyncio.get_running_loop()
```

```
    server = await loop.create_server(
        lambda: EchoServerProtocol(),
        '127.0.0.1', 8888)

    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-protocol.rst, line 758)

Unknown directive type "seealso".

```
   .. seealso::

      The :ref:`TCP echo server using streams <asyncio-tcp-echo-server-streams>`
      example uses the high-level :func:`asyncio.start_server` function.
```

TCP Echo Client

A TCP echo client using the :meth:`loop.create_connection` method, sends data, and waits until the connection is closed:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 768); *backlink*

Unknown interpreted text role "meth".

```
import asyncio


class EchoClientProtocol(asyncio.Protocol):
    def __init__(self, message, on_con_lost):
        self.message = message
        self.on_con_lost = on_con_lost

    def connection_made(self, transport):
        transport.write(self.message.encode())
        print('Data sent: {!r}'.format(self.message))

    def data_received(self, data):
        print('Data received: {!r}'.format(data.decode()))

    def connection_lost(self, exc):
        print('The server closed the connection')
        self.on_con_lost.set_result(True)


async def main():
    # Get a reference to the event loop as we plan to use
    # low-level APIs.
    loop = asyncio.get_running_loop()

    on_con_lost = loop.create_future()
    message = 'Hello World!'

    transport, protocol = await loop.create_connection(
        lambda: EchoClientProtocol(message, on_con_lost),
        '127.0.0.1', 8888)

    # Wait until the protocol signals that the connection
    # is lost and close the transport.
    try:
        await on_con_lost
    finally:
        transport.close()


asyncio.run(main())
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 814)

Unknown directive type "seealso".

```
   .. seealso::
```

```
The :ref:`TCP echo client using streams <asyncio-tcp-echo-client-streams>`
example uses the high-level :func:`asyncio.open_connection` function.
```

UDP Echo Server

A UDP echo server, using the :meth:`loop.create_datagram_endpoint` method, sends back received data:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 825); *backlink*

Unknown interpreted text role "meth".

```
import asyncio


class EchoServerProtocol:
    def connection_made(self, transport):
        self.transport = transport

    def datagram_received(self, data, addr):
        message = data.decode()
        print('Received %r from %s' % (message, addr))
        print('Send %r to %s' % (message, addr))
        self.transport.sendto(data, addr)


async def main():
    print("Starting UDP server")

    # Get a reference to the event loop as we plan to use
    # low-level APIs.
    loop = asyncio.get_running_loop()

    # One protocol instance will be created to serve all
    # client requests.
    transport, protocol = await loop.create_datagram_endpoint(
        lambda: EchoServerProtocol(),
        local_addr=('127.0.0.1', 9999))

    try:
        await asyncio.sleep(3600)  # Serve for 1 hour.
    finally:
        transport.close()


asyncio.run(main())
```

UDP Echo Client

A UDP echo client, using the :meth:`loop.create_datagram_endpoint` method, sends data and closes the transport when it receives the answer:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 869); *backlink*

Unknown interpreted text role "meth".

```
import asyncio


class EchoClientProtocol:
    def __init__(self, message, on_con_lost):
        self.message = message
        self.on_con_lost = on_con_lost
        self.transport = None

    def connection_made(self, transport):
        self.transport = transport
        print('Send:', self.message)
        self.transport.sendto(self.message.encode())

    def datagram_received(self, data, addr):
        print("Received:", data.decode())
```

```
            print("Close the socket")
            self.transport.close()

        def error_received(self, exc):
            print('Error received:', exc)

        def connection_lost(self, exc):
            print("Connection closed")
            self.on_con_lost.set_result(True)


    async def main():
        # Get a reference to the event loop as we plan to use
        # low-level APIs.
        loop = asyncio.get_running_loop()

        on_con_lost = loop.create_future()
        message = "Hello World!"

        transport, protocol = await loop.create_datagram_endpoint(
            lambda: EchoClientProtocol(message, on_con_lost),
            remote_addr=('127.0.0.1', 9999))

        try:
            await on_con_lost
        finally:
            transport.close()


    asyncio.run(main())
```

Connecting Existing Sockets

Wait until a socket receives data using the :meth:`loop.create_connection` method with a protocol:

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 926); *backlink***
>
> Unknown interpreted text role "meth".

```
    import asyncio
    import socket


    class MyProtocol(asyncio.Protocol):

        def __init__(self, on_con_lost):
            self.transport = None
            self.on_con_lost = on_con_lost

        def connection_made(self, transport):
            self.transport = transport

        def data_received(self, data):
            print("Received:", data.decode())

            # We are done: close the transport;
            # connection_lost() will be called automatically.
            self.transport.close()

        def connection_lost(self, exc):
            # The socket has been closed
            self.on_con_lost.set_result(True)


    async def main():
        # Get a reference to the event loop as we plan to use
        # low-level APIs.
        loop = asyncio.get_running_loop()
        on_con_lost = loop.create_future()

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

        # Register the socket to wait for data.
        transport, protocol = await loop.create_connection(
            lambda: MyProtocol(on_con_lost), sock=rsock)

        # Simulate the reception of data from the network.
```

```
    loop.call_soon(wsock.send, 'abc'.encode())

    try:
        await protocol.on_con_lost
    finally:
        transport.close()
        wsock.close()

asyncio.run(main())
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 978)

Unknown directive type "seealso".

```
   .. seealso::

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

      The :ref:`register an open socket to wait for data using streams
      <asyncio_example_create_connection-streams>` example uses high-level streams
      created by the :func:`open_connection` function in a coroutine.
```

loop.subprocess_exec() and SubprocessProtocol

An example of a subprocess protocol used to get the output of a subprocess and to wait for the subprocess exit.

The subprocess is created by the :meth:`loop.subprocess_exec` method:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst, line 996); *backlink*

Unknown interpreted text role "meth".

```
import asyncio
import sys

class DateProtocol(asyncio.SubprocessProtocol):
    def __init__(self, exit_future):
        self.exit_future = exit_future
        self.output = bytearray()

    def pipe_data_received(self, fd, data):
        self.output.extend(data)

    def process_exited(self):
        self.exit_future.set_result(True)

async def get_date():
    # Get a reference to the event loop as we plan to use
    # low-level APIs.
    loop = asyncio.get_running_loop()

    code = 'import datetime; print(datetime.datetime.now())'
    exit_future = asyncio.Future(loop=loop)

    # Create the subprocess controlled by DateProtocol;
    # redirect the standard output into a pipe.
    transport, protocol = await loop.subprocess_exec(
        lambda: DateProtocol(exit_future),
        sys.executable, '-c', code,
        stdin=None, stderr=None)

    # Wait for the subprocess exit using the process_exited()
    # method of the protocol.
    await exit_future

    # Close the stdout pipe.
    transport.close()

    # Read the output which was collected by the
    # pipe_data_received() method of the protocol.
    data = bytes(protocol.output)
    return data.decode('ascii').rstrip()

date = asyncio.run(get_date())
```

```
    print(f"Current date: {date}")
```

See also the :ref:`same example <asyncio_example_create_subprocess_exec>` written using high-level APIs.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]asyncio-protocol.rst`, **line 1042**); ***backlink***
>
> Unknown interpreted text role "ref".