Trollius Documentation

Release 2.2

Victor Stinner

December 05, 2016

Contents

Warning: *The Trollius project is now deprecated!*

Trollius provides infrastructure for writing single-threaded concurrent code using coroutines, multiplexing I/O access over sockets and other resources, running network clients and servers, and other related primitives. Here is a more detailed list of the package contents:

- a pluggable event loop with various system-specific implementations;
- transport and protocol abstractions (similar to those in Twisted);
- concrete support for TCP, UDP, SSL, subprocess pipes, delayed calls, and others (some may be system-dependent);
- a `Future` class that mimics the one in the `concurrent.futures` module, but adapted for use with the event loop;
- coroutines and tasks based on generators (`yield`), to help write concurrent code in a sequential fashion;
- cancellation support for `Futures` and coroutines;
- synchronization primitives for use between coroutines in a single thread, mimicking those in the `threading` module;
- an interface for passing work off to a threadpool, for times when you absolutely, positively have to use a library that makes blocking I/O calls.

Trollius is a portage of the asyncio project (`asyncio` module, PEP 3156) on Python 2. Trollius works on Python 2.7, 3.3 and 3.4. It has been tested on Windows, Linux, Mac OS X, FreeBSD and OpenIndiana.

- Asyncio documentation
- Trollius documentation (this document)
- Trollius project in the Python Cheeseshop (PyPI) (download wheel packages and tarballs)
- Trollius project at Github (bug tracker, source code)
- Mailing list: python-tulip Google Group
- IRC: `#asyncio` channel on the Freenode network

- Copyright/license: Open source, Apache 2.0. Enjoy!

See also the asyncio project at Github.

CHAPTER 1

Table Of Contents

1.1 Trollius is deprecated

Warning: The Trollius project is now deprecated!

Trollius is deprecated since the release 2.1. The maintainer of Trollius, Victor Stinner, doesn't want to maintain the project anymore for many reasons. This page lists some reasons.

DON'T PANIC! There is the asyncio project which has the same API and is well maintained! Only trollius is deprecated.

Since the Trollius is used for some projects in the wild, Trollius will not disappear. You can only expect *minimum* maintenance like minor bugfixes. Don't expect new features nor synchronization with the latest asyncio.

To be clear: I am looking for a new maintainer. If you want to take over trollius: please do it, I will give you everything you need (and maybe more!).

1.1.1 asyncio

asyncio is here! asyncio is well maintainted, has a growing community, has many libraries and don't stop evolving to be enhanced by users feedbacks. I (Victor Stinner) even heard that it is fast!

asyncio requires Python 3.3 or newer. Yeah, you all have a long list of reasons to not port your legacy code for Python 3. But I have my own reasons to prefer to invest in the Python 3 rather than in legacy Python (Python 2).

1.1.2 No Trollius Community

- Very the asyncio is growing everyday, there is no trollius community. Sadly, asyncio libraries don't work for trollius.
- Only *very few libraries support Trollius*: to be clear, there is no HTTP client for Trollius, whereas HTTP is the most common protocol in 2015.
- It's a deliberate choice of library authors to not support Trollius to keep a simple code base. The Python 3 is simpler than Python 2: supporting Python 2 in a library requires more work. For example, aiohttp doesn't want to support trollius.

1.1.3 Python 2

- Seriously? Come on! Stop procrastination and upgrade your code to Python 3!

1.1.4 Lack of interest

- The Trollius project was created to replace eventlet with asyncio (trollius) in the OpenStack project, but replacing eventlet with trollius has failed for different reasons. The original motivation is gone.

1.1.5 Technical issues with trollius

- While Trollius API is "simple", the implementation is very complex to be efficient on all platforms.
- Trollius requires *backports* of libraries to support old Python versions. These backports are not as well supported as the version in the Python standard library.
- Supporting Python 2.7, Python 3.3 and Python 3.5 in the same code base and supporting asyncio is very difficult. Generators and coroutines changed a lot in each Python version. For example, hacks are required to support Python 3.5 with the PEP 479 which changed the usage of the `StopIteration` exception. Trollius initially also supported Python 2.6 and 3.2.

1.1.6 Technical issues related to asyncio and yield from

- Synchronizing trollius with asyncio is a complex, tricky and error-prone task which requires to be very carefull and a lot of manual changes.
- Porting Python 3 asyncio to Python 2 requires a lot of subtle changes which takes a lot of time at each synchronization.
- It is not possible to use asyncio `yield from` coroutines in Python 2, since the `yield from` instruction raises a `SyntaxError` exceptions. Supporting Trollius and asyncio requires to duplicate some parts of the library and application code which makes the code more complex and more difficult to maintain.
- Trollius coroutines are slower than asyncio coroutines: the `yield` instruction requires to delegate manually nested coroutines, whereas the `yield from` instruction delegates them directly in the Python language. asyncio requires less loop iterations than trollius for the same nested coroutine.

1.1.7 Other technical issues

- Building wheel packages on Windows is difficult (need a running Windows, need working Windows SDKs for each version of Python, need to test and fix bugs specific to Windows, etc.)

1.2 Using Trollius

Warning: *The Trollius project is now deprecated!*

1.2.1 Documentation of the asyncio module

The documentation of the asyncio is part of the Python project. It can be read online: asyncio - Asynchronous I/O, event loop, coroutines and tasks.

To adapt asyncio examples for Trollius, "just":

- replace `asyncio` with `trollius` (or use `import trollius as asyncio`)
- replace `yield from ...` with `yield From(...)`
- replace `yield from []` with `yield From(None)`
- in coroutines, replace `return res` with `raise Return(res)`

1.2.2 Trollius Hello World

Print `Hello World` every two seconds, using a coroutine:

```
import trollius
from trollius import From

@trollius.coroutine
def greet_every_two_seconds():
    while True:
        print('Hello World')
        yield From(trollius.sleep(2))

loop = trollius.get_event_loop()
loop.run_until_complete(greet_every_two_seconds())
```

1.2.3 Debug mode

To enable the debug mode:

- Set `TROLLIUSDEBUG` envrironment variable to `1`
- Configure logging to log at level `logging.DEBUG`, `logging.basicConfig(level=logging.DEBUG)` for example

The `BaseEventLoop.set_debug()` method can be used to set the debug mode on a specific event loop. The environment variable enables also the debug mode for coroutines.

Effect of the debug mode:

- On Python 2, `Future.set_exception()` stores the traceback, so `loop.run_until_complete()` raises the exception with the original traceback.
- Log coroutines defined but never "yielded"
- BaseEventLoop.call_soon() and BaseEventLoop.call_at() methods raise an exception if they are called from the wrong thread.
- Log the execution time of the selector
- Log callbacks taking more than 100 ms to be executed. The BaseEventLoop.slow_callback_duration attribute is the minimum duration in seconds of "slow" callbacks.
- Log most important subprocess events:
  - Log stdin, stdout and stderr transports and protocols

  - Log process identifier (pid)
  - Log connection of pipes
  - Log process exit
  - Log Process.communicate() tasks: feed stdin, read stdout and stderr
- Log most important socket events:
  - Socket connected
  - New client (socket.accept())
  - Connection reset or closed by peer (EOF)
  - Log time elapsed in DNS resolution (getaddrinfo)
  - Log pause/resume reading
  - Log time of SSL handshake
  - Log SSL handshake errors

See Debug mode of asyncio for more information.

1.3 Install Trollius

Warning: *The Trollius project is now deprecated!*

Trollius supports Python 2.7, 3.3 and 3.4.

There is an experimental support of Python 3.5. Issues with Python 3.5:

- don't support asyncio coroutines
- `Task.get_task()` is broken
- `repr(Task)` is broken

Support of Python 2.6 and 3.2 was dropped in Trollius 2.1.

1.3.1 Packages for Linux

- Debian package
- ArchLinux package
- Fedora and CentOS package: python-trollius

1.3.2 Install Trollius on Windows using pip

Since Trollius 0.2, precompiled wheel packages are now distributed on the Python Cheeseshop (PyPI). Procedure to install Trollius on Windows:

- Install pip, download `get-pip.py` and type:

```
\Python27\python.exe get-pip.py
```

- If you already have pip, ensure that you have at least pip 1.4. If you need to upgrade:

```
\Python27\python.exe -m pip install -U pip
```

- Install Trollius:

```
\Python27\python.exe -m pip install trollius
```

- pip also installs the `futures` dependency

Note: Only wheel packages for Python 2.7, 3.3 and 3.4 are currently distributed on the Cheeseshop (PyPI). If you need wheel packages for other Python versions, please ask.

1.3.3 Download source code

Command to download the development version of the source code (`trollius` branch):

```
git clone https://github.com/haypo/trollius.git -b trollius
```

The actual code lives in the `trollius` subdirectory. Tests are in the `tests` subdirectory.

See the trollius project at Github.

The source code of the Trollius project is in the `trollius` branch of the Mercurial repository, not in the default branch. The default branch is the Tulip project, Trollius repository is a fork of the Tulip repository.

1.3.4 Dependencies

Trollius requires the six module.

Python 2.7 requires futures to get a backport of `concurrent.futures`.

1.3.5 Build manually Trollius on Windows

On Windows, if you cannot use precompiled wheel packages, an extension module must be compiled: the `_overlapped` module (source code: `overlapped.c`). Read Compile Python extensions on Windows to prepare your environment to build the Python extension. Then build the extension using:

```
C:\Python27\python.exe setup.py build_ext
```

1.3.6 Backports

To support old Python versions, many Python modules of the standard library have been backported:

Name	Python	Backport
OSError	3.3	asyncio.py33_exceptions
_overlapped	3.4	asyncio._overlapped
_winapi	3.3	asyncio.py33_winapi
collections.OrderedDict	2.7, 3.1	ordereddict (PyPI)
concurrent.futures	3.2	futures (PyPI)
selectors	3.4	asyncio.selectors
ssl	3.2, 3.3	asyncio.py3_ssl
time.monotonic	3.3	asyncio.time_monotonic
unittest	2.7, 3.1	unittest2 (PyPI)
unittest.mock	3.3	mock (PyPI)
weakref.WeakSet	2.7, 3.0	asyncio.py27_weakrefset

1.4 Trollius Libraries

Warning: *The Trollius project is now deprecated!*

1.4.1 Libraries compatible with asyncio and trollius

- aioeventlet: asyncio API implemented on top of eventlet
- aiogevent: asyncio API implemented on top of gevent
- AutobahnPython: WebSocket & WAMP for Python, it works on Trollius (Python 2.6 and 2.7), asyncio (Python 3.3) and Python 3.4 (asyncio), and also on Twisted.
- Pulsar: Event driven concurrent framework for Python. With pulsar you can write asynchronous servers performing one or several activities in different threads and/or processes. Trollius 0.3 requires Pulsar 0.8.2 or later. Pulsar uses the `asyncio` module if available, or import `trollius`.
- Tornado supports asyncio and Trollius since Tornado 3.2: tornado.platform.asyncio — Bridge between asyncio and Tornado. It tries to import asyncio or fallback on importing trollius.

1.4.2 Specific Ports

- trollius-redis: A port of asyncio-redis to trollius

1.5 Trollius and asyncio

Warning: *The Trollius project is now deprecated!*

1.5.1 Differences between Trollius and asyncio

Syntax of coroutines

The major difference between Trollius and asyncio is the syntax of coroutines:

asyncio	Trollius
`yield from ...`	`yield From(...)`
`yield from []`	`yield From(None)`
`return`	`raise Return()`
`return x`	`raise Return(x)`
`return x, y`	`raise Return(x, y)`

Because of this major difference, it was decided to call the module `trollius` instead of `asyncio`. This choice also allows to use Trollius on Python 3.4 and later. Changing imports is not enough to use Trollius code with asyncio: the asyncio event loop explicit rejects coroutines using `yield` (instead of `yield from`).

OSError and socket.error exceptions

The `OSError` exception changed in Python 3.3: there are now subclasses like `ConnectionResetError` or `BlockingIOError`. The exception hierarchy also changed: `socket.error` is now an alias to `OSError`. The `asyncio` module is written for Python 3.3 and newer and so is based on these new exceptions.

See also:

PEP 3151: Reworking the OS and IO exception hierarchy.

On Python 3.2 and older, Trollius wraps `OSError`, `IOError`, `socket.error` and `select.error` exceptions on operating system and socket operations to raise more specific exceptions, subclasses of `OSError`:

- `trollius.BlockingIOError`
- `trollius.BrokenPipeError`
- `trollius.ChildProcessError`
- `trollius.ConnectionAbortedError`
- `trollius.ConnectionRefusedError`
- `trollius.ConnectionResetError`
- `trollius.FileNotFoundError`
- `trollius.InterruptedError`
- `trollius.PermissionError`

On Python 3.3 and newer, these symbols are just aliases to builtin exceptions.

Note: `ssl.SSLError` exceptions are not wrapped to `OSError`, even if `ssl.SSLError` is a subclass of `socket.error`.

SSLError

On Python 3.2 and older, Trollius wraps `ssl.SSLError` exceptions to raise more specific exceptions, subclasses of `ssl.SSLError`, to mimic the Python 3.3:

- `trollius.SSLEOFError`
- `trollius.SSLWantReadError`
- `trollius.SSLWantWriteError`

On Python 3.3 and newer, these symbols are just aliases to exceptions of the `ssl` module.

`trollius.BACKPORT_SSL_ERRORS` constant:

- `True` if `ssl.SSLError` are wrapped to Trollius exceptions (Python 2 older than 2.7.9, or Python 3 older than 3.3),
- `False` is trollius SSL exceptions are just aliases.

SSLContext

Python 3.3 has a new `ssl.SSLContext` class: see the documentaton of the ssl.SSLContext class.

On Python 3.2 and older, Trollius has a basic `trollius.SSLContext` class to mimic Python 3.3 API, but it only has a few features:

- `protocol`, `certfile` and `keyfile` attributes
- read-only `verify_mode` attribute: its value is `CERT_NONE`
- `load_cert_chain(certfile, keyfile)` method
- `wrap_socket(sock, **kw)` method: see the `ssl.wrap_socket()` documentation of your Python version for the keyword parameters

Example of missing features:

- no `options` attribute
- the `verify_mode` attriubte cannot be modified
- no `set_default_verify_paths()` method
- no "Server Name Indication" (SNI) support
- etc.

On Python 3.2 and older, the trollius SSL transport does not have the `'compression'` extra info.

`trollius.BACKPORT_SSL_CONTEXT` constant:

- `True` if `trollius.SSLContext` is the backported class (Python 2 older than 2.7.9, or Python 3 older than 3.3),
- `False` if `trollius.SSLContext` is just an alias to `ssl.SSLContext`.

Other differences

- Trollius uses the `TROLLIUSDEBUG` envrionment variable instead of the `PYTHONASYNCIODEBUG` envrionment variable. `TROLLIUSDEBUG` variable is used even if the Python command line option `-E` is used.
- `asyncio.subprocess` has no `DEVNULL` constant
- Python 2 does not support keyword-only parameters.
- If the `concurrent.futures` module is missing, `BaseEventLoop.run_in_executor()` uses a synchronous executor instead of a pool of threads. It blocks until the function returns. For example, DNS resolutions are blocking in this case.
- Trollius has more symbols than asyncio for compatibility with Python older than 3.3:
  - `From`: part of `yield From(...)` syntax
  - `Return`: part of `raise Return(...)` syntax

1.5.2 Write code working on Trollius and asyncio

Trollius and asyncio are different, especially for coroutines (`yield From(...)` vs `yield from ...`).

To use asyncio or Trollius on Python 2 and Python 3, add the following code at the top of your file:

```
try:
    # Use builtin asyncio on Python 3.4+, or asyncio on Python 3.3
    import asyncio
except ImportError:
    # Use Trollius on Python <= 3.2
    import trollius as asyncio
```

It is possible to write code working on both projects using only callbacks. This option is used by the following projects which work on Trollius and asyncio:

- AutobahnPython: WebSocket & WAMP for Python, it works on Trollius (Python 2.6 and 2.7), asyncio (Python 3.3) and Python 3.4 (asyncio), and also on Twisted.
- Pulsar: Event driven concurrent framework for Python. With pulsar you can write asynchronous servers performing one or several activities in different threads and/or processes. Trollius 0.3 requires Pulsar 0.8.2 or later. Pulsar uses the `asyncio` module if available, or import `trollius`.
- Tornado supports asyncio and Trollius since Tornado 3.2: tornado.platform.asyncio — Bridge between asyncio and Tornado. It tries to import asyncio or fallback on importing trollius.

Another option is to provide functions returning `Future` objects, so the caller can decide to use callback using `fut.add_done_callback(callback)` or to use coroutines (`yield From(fut)` for Trollius, or `yield from fut` for asyncio). This option is used by the aiodns project for example.

Since Trollius 0.4, it's possible to use asyncio and Trollius coroutines in the same process. The only limit is that the event loop must be a Trollius event loop.

Note: The Trollius module was called `asyncio` in Trollius version 0.2. The module name changed to `trollius` to support Python 3.4.

1.6 Run tests

Warning: *The Trollius project is now deprecated!*

1.6.1 Run tests with tox

The tox project can be used to build a virtual environment with all runtime and test dependencies and run tests against different Python versions (2.7, 3.3, 3.4).

For example, to run tests with Python 2.7, just type:

```
tox -e py27
```

To run tests against other Python versions:

- `py27`: Python 2.7
- `py33`: Python 3.3

- `py34`: Python 3.4

1.6.2 Test Dependencies

On Python older than 3.3, unit tests require the mock module. Python 2.6 and 2.7 require also unittest2.

To run `run_aiotest.py`, you need the aiotest test suite: `pip install aiotest`.

1.6.3 Run tests on UNIX

Run the following commands from the directory of the Trollius project.

To run tests:

```
make test
```

To run coverage (`coverage` package is required):

```
make coverage
```

1.6.4 Run tests on Windows

Run the following commands from the directory of the Trollius project.

You can run the tests as follows:

```
C:\Python27\python.exe runtests.py
```

And coverage as follows:

```
C:\Python27\python.exe runtests.py --coverage
```

1.7 CPython bugs

The development of asyncio and trollius helped to identify different bugs in CPython:

- 2.5.0 <= python <= 3.4.2: sys.exc_info() bug when yield/yield-from is used in an except block in a generator (#23353>). The fix will be part of Python 3.4.3. _UnixSelectorEventLoop._make_subprocess_transport() and ProactorEventLoop._make_subprocess_transport() work around the bug.
- python == 3.4.0: Segfault in gc with cyclic trash (#21435). Regression introduced in Python 3.4.0, fixed in Python 3.4.1. Status in Ubuntu the February, 3th 2015: only Ubuntu Trusty (14.04 LTS) is impacted (bug #1367907: Segfault in gc with cyclic trash, see also update Python3 for trusty #1348954)
- 3.3.0 <= python <= 3.4.0: gen.send(tuple) unpacks the tuple instead of passing 1 argument (the tuple) when gen is an object with a send() method, not a classic generator (#21209). Regression introduced in Python 3.4.0, fixed in Python 3.4.1. trollius.CoroWrapper.send() works around the issue, the bug is checked at runtime once, when the module is imported.

1.8 Change log

Warning: *The Trollius project is now deprecated!*

1.8.1 Version 2.1

Changes:

- *The Trollius project is now deprecated*.
- Ugly hack to support Python 3.5 with the PEP 479. asyncio coroutines are not supported on Python 3.5.
- Better exception traceback. Patch written by Dhawal Yogesh Bhanushali.
- Drop support for Python 2.6 and 3.2.
- Fix tests on Windows with Python 2. Patch written by Gabi Davar.

1.8.2 Version 2.0 (2015-07-13)

Summary:

- SSL support on Windows for proactor event loop with Python 3.5 and newer
- Many race conditions were fixed in the proactor event loop
- Trollius moved to Github and the fork was recreated on top to asyncio git repository
- Many resource leaks (ex: unclosed sockets) were fixed
- Optimization of socket connections: avoid: don't call the slow getaddrinfo() function to ensure that the address is already resolved. The check is now only done in debug mode.

The Trollius project moved from Bitbucket to Github. The project is now a fork of the Git repository of the asyncio project (previously called the "tulip" project), the trollius source code lives in the trollius branch.

The new Trollius home page is now: https://github.com/haypo/trollius

The asyncio project moved to: https://github.com/python/asyncio

Note: the PEP 492 is not supported in trollius yet.

API changes:

- Issue #234: Drop JoinableQueue on Python 3.5+
- add the asyncio.ensure_future() function, previously called async(). The async() function is now deprecated.
- New event loop methods: set_task_factory() and get_task_factory().
- Python issue #23347: Make BaseSubprocessTransport.wait() private.
- Python issue #23347: send_signal(), kill() and terminate() methods of BaseSubprocessTransport now check if the transport was closed and if the process exited.
- Python issue #23209, #23225: selectors.BaseSelector.get_key() now raises a RuntimeError if the selector is closed. And selectors.BaseSelector.close() now clears its internal reference to the selector mapping to break a reference cycle. Initial patch written by Martin Richard.
- PipeHandle.fileno() of asyncio.windows_utils now raises an exception if the pipe is closed.

- Remove Overlapped.WaitNamedPipeAndConnect() of the _overlapped module, it is no more used and it had issues.
- Python issue #23537: Remove 2 unused private methods of BaseSubprocessTransport: _make_write_subprocess_pipe_proto, _make_read_subprocess_pipe_proto. Methods only raise NotImplementedError and are never used.
- Remove unused SSLProtocol._closing attribute

New SSL implementation:

- Python issue #22560: On Python 3.5 and newer, use new SSL implementation based on ssl.MemoryBIO instead of the legacy SSL implementation. Patch written by Antoine Pitrou, based on the work of Geert Jansen.
- If available, the new SSL implementation can be used by ProactorEventLoop to support SSL.

Enhance, fix and cleanup the IocpProactor:

- Python issue #23293: Rewrite IocpProactor.connect_pipe(). Add _overlapped.ConnectPipe() which tries to connect to the pipe for asynchronous I/O (overlapped): call CreateFile() in a loop until it doesn't fail with ERROR_PIPE_BUSY. Use an increasing delay between 1 ms and 100 ms.
- Tulip issue #204: Fix IocpProactor.accept_pipe(). Overlapped.ConnectNamedPipe() now returns a boolean: True if the pipe is connected (if ConnectNamedPipe() failed with ERROR_PIPE_CONNECTED), False if the connection is in progress.
- Tulip issue #204: Fix IocpProactor.recv(). If ReadFile() fails with ERROR_BROKEN_PIPE, the operation is not pending: don't register the overlapped.
- Python issue #23095: Rewrite _WaitHandleFuture.cancel(). _WaitHandleFuture.cancel() now waits until the wait is cancelled to clear its reference to the overlapped object. To wait until the cancellation is done, UnregisterWaitEx() is used with an event instead of UnregisterWait().
- Python issue #23293: Rewrite IocpProactor.connect_pipe() as a coroutine. Use a coroutine with asyncio.sleep() instead of call_later() to ensure that the scheduled call is cancelled.
- Fix ProactorEventLoop.start_serving_pipe(). If a client was connected before the server was closed: drop the client (close the pipe) and exit
- Python issue #23293: Cleanup IocpProactor.close(). The special case for connect_pipe() is no more needed. connect_pipe() doesn't use overlapped operations anymore.
- IocpProactor.close(): don't cancel futures which are already cancelled
- Enhance (fix) BaseProactorEventLoop._loop_self_reading(). Handle correctly CancelledError: just exit. On error, log the exception and exit; don't try to close the event loop (it doesn't work).

Bug fixes:

- Fix LifoQueue's and PriorityQueue's put() and task_done().
- Issue #222: Fix the @coroutine decorator for functions without __name__ attribute like functools.partial(). Enhance also the representation of a CoroWrapper if the coroutine function is a functools.partial().
- Python issue #23879: SelectorEventLoop.sock_connect() must not call connect() again if the first call to connect() raises an InterruptedError. When the C function connect() fails with EINTR, the connection runs in background. We have to wait until the socket becomes writable to be notified when the connection succeed or fails.
- Fix _SelectorTransport.__repr__() if the event loop is closed
- Fix repr(BaseSubprocessTransport) if it didn't start yet
- Workaround CPython bug #23353. Don't use yield/yield-from in an except block of a generator. Store the exception and handle it outside the except block.

- Fix BaseSelectorEventLoop._accept_connection(). Close the transport on error. In debug mode, log errors using call_exception_handler().
- Fix _UnixReadPipeTransport and _UnixWritePipeTransport. Only start reading when connection_made() has been called.
- Fix _SelectorSslTransport.close(). Don't call protocol.connection_lost() if protocol.connection_made() was not called yet: if the SSL handshake failed or is still in progress. The close() method can be called if the creation of the connection is cancelled, by a timeout for example.
- Fix _SelectorDatagramTransport constructor. Only start reading after connection_made() has been called.
- Fix _SelectorSocketTransport constructor. Only start reading when connection_made() has been called: protocol.data_received() must not be called before protocol.connection_made().
- Fix SSLProtocol.eof_received(). Wake-up the waiter if it is not done yet.
- Close transports on error. Fix create_datagram_endpoint(), connect_read_pipe() and connect_write_pipe(): close the transport if the task is cancelled or on error.
- Close the transport on subprocess creation failure
- Fix _ProactorBasePipeTransport.close(). Set the _read_fut attribute to None after cancelling it.
- Python issue #23243: Fix _UnixWritePipeTransport.close(). Do nothing if the transport is already closed. Before it was not possible to close the transport twice.
- Python issue #23242: SubprocessStreamProtocol now closes the subprocess transport at subprocess exit. Clear also its reference to the transport.
- Fix BaseEventLoop._create_connection_transport(). Close the transport if the creation of the transport (if the waiter) gets an exception.
- Python issue #23197: On SSL handshake failure, check if the waiter is cancelled before setting its exception.
- Python issue #23173: Fix SubprocessStreamProtocol.connection_made() to handle cancelled waiter.
- Python issue #23173: If an exception is raised during the creation of a subprocess, kill the subprocess (close pipes, kill and read the return status). Log an error in such case.
- Python issue #23209: Break some reference cycles in asyncio. Patch written by Martin Richard.

Optimization:

- Only call _check_resolved_address() in debug mode. _check_resolved_address() is implemented with getaddrinfo() which is slow. If available, use socket.inet_pton() instead of socket.getaddrinfo(), because it is much faster

Other changes:

- Python issue #23456: Add missing @coroutine decorators
- Python issue #23475: Fix test_close_kill_running(). Really kill the child process, don't mock completly the Popen.kill() method. This change fix memory leaks and reference leaks.
- BaseSubprocessTransport: repr() mentions when the child process is running
- BaseSubprocessTransport.close() doesn't try to kill the process if it already finished.
- Tulip issue #221: Fix docstring of QueueEmpty and QueueFull
- Fix subprocess_attach_write_pipe example. Close the transport, not directly the pipe.
- Python issue #23347: send_signal(), terminate(), kill() don't check if the transport was closed. The check broken a Tulip example and this limitation is arbitrary. Check if _proc is None should be enough. Enhance also close(): do nothing when called the second time.

- Python issue #23347: Refactor creation of subprocess transports.
- Python issue #23243: On Python 3.4 and newer, emit a ResourceWarning when an event loop or a transport is not explicitly closed
- tox.ini: enable ResourceWarning warnings
- Python issue #23243: test_sslproto: Close explicitly transports
- SSL transports now clear their reference to the waiter.
- Python issue #23208: Add BaseEventLoop._current_handle. In debug mode, BaseEventLoop._run_once() now sets the BaseEventLoop._current_handle attribute to the handle currently executed.
- Replace test_selectors.py with the file of Python 3.5 adapted for asyncio and Python 3.3.
- Tulip issue #184: FlowControlMixin constructor now get the event loop if the loop parameter is not set.
- _ProactorBasePipeTransport now sets the _sock attribute to None when the transport is closed.
- Python issue #23219: cancelling wait_for() now cancels the task
- Python issue #23243: Close explicitly event loops and transports in tests
- Python issue #23140: Fix cancellation of Process.wait(). Check the state of the waiter future before setting its result.
- Python issue #23046: Expose the BaseEventLoop class in the asyncio namespace
- Python issue #22926: In debug mode, call_soon(), call_at() and call_later() methods of BaseEventLoop now use the identifier of the current thread to ensure that they are called from the thread running the event loop. Before, the get_event_loop() method was used to check the thread, and no exception was raised when the thread had no event loop. Now the methods always raise an exception in debug mode when called from the wrong thread. It should help to notice misusage of the API.

1.8.3 2014-12-19: Version 1.0.4

Changes:

- Python issue #22922: create_task(), call_at(), call_soon(), call_soon_threadsafe() and run_in_executor() now raise an error if the event loop is closed. Initial patch written by Torsten Landschoff.
- Python issue #22921: Don't require OpenSSL SNI to pass hostname to ssl functions. Patch by Donald Stufft.
- Add run_aiotest.py: run the aiotest test suite.
- tox now also run the aiotest test suite
- Python issue #23074: get_event_loop() now raises an exception if the thread has no event loop even if assertions are disabled.

Bugfixes:

- Fix a race condition in BaseSubprocessTransport._try_finish(): ensure that connection_made() is called before connection_lost().
- Python issue #23009: selectors, make sure EpollSelecrtor.select() works when no file descriptor is registered.
- Python issue #22922: Fix ProactorEventLoop.close(). Call _stop_accept_futures() before sestting the _closed attribute, otherwise call_soon() raises an error.
- Python issue #22429: Fix EventLoop.run_until_complete(), don't stop the event loop if a BaseException is raised, because the event loop is already stopped.
- Initialize more Future and Task attributes in the class definition to avoid attribute errors in destructors.

- Python issue #22685: Set the transport of stdout and stderr StreamReader objects in the SubprocessStreamProtocol. It allows to pause the transport to not buffer too much stdout or stderr data.
- BaseSelectorEventLoop.close() now closes the self-pipe before calling the parent close() method. If the event loop is already closed, the self-pipe is not unregistered from the selector.

1.8.4 2014-10-20: Version 1.0.3

Changes:

- On Python 2 in debug mode, Future.set_exception() now stores the traceback object of the exception in addition to the exception object. When a task waiting for another task and the other task raises an exception, the traceback object is now copied with the exception. Be careful, storing the traceback object may create reference leaks.
- Use ssl.create_default_context() if available to create the default SSL context: Python 2.7.9 and newer, or Python 3.4 and newer.
- On Python 3.5 and newer, reuse socket.socketpair() in the windows_utils submodule.
- On Python 3.4 and newer, use os.set_inheritable().
- Enhance protocol representation: add "closed" or "closing" info.
- run_forever() now consumes BaseException of the temporary task. If the coroutine raised a BaseException, consume the exception to not log a warning. The caller doesn't have access to the local task.
- Python issue 22448: cleanup _run_once(), only iterate once to remove delayed calls that were cancelled.
- The destructor of the Return class now shows where the Return object was created.
- run_tests.py doesn't catch any exceptions anymore when loading tests, only catch SkipTest.
- Fix (SSL) tests for the future Python 2.7.9 which includes a "new" ssl module: module backported from Python 3.5.
- BaseEventLoop.add_signal_handler() now raises an exception if the parameter is a coroutine function.
- Coroutine functions and objects are now rejected with a TypeError by the following functions: add_signal_handler(), call_at(), call_later(), call_soon(), call_soon_threadsafe(), run_in_executor().

1.8.5 2014-10-02: Version 1.0.2

This release fixes bugs. It also provides more information in debug mode on error.

Major changes:

- Tulip issue #203: Add _FlowControlMixin.get_write_buffer_limits() method.
- Python issue #22063: socket operations (socket,recv, sock_sendall, sock_connect, sock_accept) of SelectorEventLoop now raise an exception in debug mode if sockets are in blocking mode.

Major bugfixes:

- Tulip issue #205: Fix a race condition in BaseSelectorEventLoop.sock_connect().
- Tulip issue #201: Fix a race condition in wait_for(). Don't raise a TimeoutError if we reached the timeout and the future completed in the same iteration of the event loop. A side effect of the bug is that Queue.get() looses items.
- PipeServer.close() now cancels the "accept pipe" future which cancels the overlapped operation.

Other changes:

- Python issue #22448: Improve cancelled timer callback handles cleanup. Patch by Joshua Moore-Oliva.
- Python issue #22369: Change "context manager protocol" to "context management protocol". Patch written by Serhiy Storchaka.
- Tulip issue #206: In debug mode, keep the callback in the representation of Handle and TimerHandle after cancel().
- Tulip issue #207: Fix test_tasks.test_env_var_debug() to use correct asyncio module.
- runtests.py: display a message to mention if tests are run in debug or release mode
- Tulip issue #200: Log errors in debug mode instead of simply ignoring them.
- Tulip issue #200: _WaitHandleFuture._unregister_wait() now catchs and logs exceptions.
- _fatal_error() method of _UnixReadPipeTransport and _UnixWritePipeTransport now log all exceptions in debug mode
- Fix debug log in BaseEventLoop.create_connection(): get the socket object from the transport because SSL transport closes the old socket and creates a new SSL socket object.
- Remove the _SelectorSslTransport._rawsock attribute: it contained the closed socket (not very useful) and it was not used.
- Fix _SelectorTransport.__repr__() if the transport was closed
- Use the new os.set_blocking() function of Python 3.5 if available

1.8.6 2014-07-30: Version 1.0.1

This release supports PyPy and has a better support of asyncio coroutines, especially in debug mode.

Changes:

- Tulip issue #198: asyncio.Condition now accepts an optional lock object.
- Enhance representation of Future and Future subclasses: add "created at".

Bugfixes:

- Fix Trollius issue #9: @trollius.coroutine now works on callbable objects (without `__name__` attribute), not only on functions.
- Fix Trollius issue #13: asyncio futures are now accepted in all functions: as_completed(), async(), @coroutine, gather(), run_until_complete(), wrap_future().
- Fix support of asyncio coroutines in debug mode. If the last instruction of the coroutine is "yield from", it's an asyncio coroutine and it does not need to use From().
- Fix and enhance _WaitHandleFuture.cancel():
  - Tulip issue #195: Fix a crash on Windows: don't call UnregisterWait() twice if a _WaitHandleFuture is cancelled twice.
  - Fix _WaitHandleFuture.cancel(): return the result of the parent cancel() method (True or False).
  - _WaitHandleFuture.cancel() now notify IocpProactor through the overlapped object that the wait was cancelled.
- Tulip issue #196: _OverlappedFuture now clears its reference to the overlapped object. IocpProactor keeps a reference to the overlapped object until it is notified of its completion. Log also an error in debug mode if it gets unexpected notifications.
- Fix runtest.py to be able to log at level DEBUG.

Other changes:

- BaseSelectorEventLoop._write_to_self() now logs errors in debug mode.
- Fix as_completed(): it's not a coroutine, don't use `yield From(...)` but `yield ...`
- Tulip issue #193: Convert StreamWriter.drain() to a classic coroutine.
- Tulip issue #194: Don't use sys.getrefcount() in unit tests: the full test suite now pass on PyPy.

1.8.7 2014-07-21: Version 1.0

Major Changes

- Event loops have a new `create_task()` method, which is now the recommanded way to create a task object. This method can be overriden by third-party event loops to use their own task class.
- The debug mode has been improved a lot. Set `TROLLIUSDEBUG` envrironment variable to `1` and configure logging to log at level `logging.DEBUG` (ex: `logging.basicConfig(level=logging.DEBUG)`). Changes:
  - much better representation of Trollius objects (ex: `repr(task)`): unified `<Class arg1 arg2 ...>` format, use qualified name when available
  - show the traceback where objects were created
  - show the current filename and line number for coroutine
  - show the filename and line number where objects were created
  - log most important socket events
  - log most important subprocess events
- `Handle.cancel()` now clears references to callback and args
- Log an error if a Task is destroyed while it is still pending, but only on Python 3.4 and newer.
- Fix for asyncio coroutines when passing tuple value in debug mode. `CoroWrapper.send()` now checks if it is called from a "yield from" generator to decide if the parameter should be unpacked or not.
- `Process.communicate()` now ignores `BrokenPipeError` and `ConnectionResetError` exceptions.
- Rewrite signal handling on Python 3.3 and newer to fix a race condition: use the "self-pipe" to get signal numbers.

Other Changes

- Fix `ProactorEventLoop()` in debug mode
- Fix a race condition when setting the result of a Future with `call_soon()`. Add an helper, a private method, to set the result only if the future was not cancelled.
- Fix `asyncio.__all__`: export also `unix_events` and `windows_events` symbols. For example, on Windows, it was not possible to get `ProactorEventLoop` or `DefaultEventLoopPolicy` using `from asyncio import *`.
- `Handle.cancel()` now clears references to callback and args
- Make Server attributes and methods private, the sockets attribute remains public.

- BaseEventLoop.create_datagram_endpoint() now waits until protocol.connection_made() has been called. Document also why transport constructors use a waiter.
- _UnixSubprocessTransport: fix file mode of stdin: open stdin in write mode, not in read mode.

1.8.8 2014-06-23: version 0.4

Changes between Trollius 0.3 and 0.4:

- Trollius event loop now supports asyncio coroutines:
  - Trollius coroutines can yield asyncio coroutines,
  - asyncio coroutines can yield Trollius coroutines,
  - asyncio.set_event_loop() accepts a Trollius event loop,
  - asyncio.set_event_loop_policy() accepts a Trollius event loop policy.
- The `PYTHONASYNCIODEBUG` envrionment variable has been renamed to `TROLLIUSDEBUG`. The environment variable is now used even if the Python command line option `-E` is used.
- Synchronize with Tulip.
- Support PyPy (fix subproces, fix unit tests).

Tulip changes:

- Tulip issue #171: BaseEventLoop.close() now raises an exception if the event loop is running. You must first stop the event loop and then wait until it stopped, before closing it.
- Tulip issue #172: only log selector timing in debug mode
- Enable the debug mode of event loops when the `TROLLIUSDEBUG` environment variable is set
- BaseEventLoop._assert_is_current_event_loop() now only raises an exception if the current loop is set.
- Tulip issue #105: in debug mode, log callbacks taking more than 100 ms to be executed.
- Python issue 21595: `BaseSelectorEventLoop._read_from_self()` reads all available bytes from the "self pipe", not only a single byte. This change reduces the risk of having the pipe full and so getting the "BlockingIOError: [Errno 11] Resource temporarily unavailable" message.
- Python issue 21723: asyncio.Queue: support any type of number (ex: float) for the maximum size. Patch written by Vajrasky Kok.
- Issue #173: Enhance repr(Handle) and repr(Task): add the filename and line number, when available. For task, the current line number of the coroutine is used.
- Add BaseEventLoop.is_closed() method. run_forever() and run_until_complete() methods now raises an exception if the event loop was closed.
- Make sure that socketpair() close sockets on error. Close the listening socket if sock.bind() raises an exception.
- Fix ResourceWarning: close sockets on errors. BaseEventLoop.create_connection(), BaseEventLoop.create_datagram_endpoint() and _UnixSelectorEventLoop.create_unix_server() now close the newly created socket on error.
- Rephrase and fix docstrings.
- Fix tests on Windows: wait for the subprocess exit. Before, regrtest failed to remove the temporary test directory because the process was still running in this directory.
- Refactor unit tests.

On Python 3.5, generators now get their name from the function, no more from the code. So the `@coroutine` decorator doesn't loose the original name of the function anymore.

1.8.9 2014-05-26: version 0.3

Rename the Python module `asyncio` to `trollius` to support Python 3.4. On Python 3.4, there is already a module called `asyncio` in the standard library which conflicted with `asyncio` module of Trollius 0.2. To write asyncio code working on Trollius and Tulip, use `import trollius as asyncio`.

Changes between Trollius 0.2 and 0.3:

- Synchronize with Tulip 3.4.1.
- Enhance Trollius documentation.
- Trollius issue #7: Fix `asyncio.time_monotonic` on Windows older than Vista (ex: Windows 2000 and Windows XP).
- Fedora packages have been accepted.

Changes between Tulip 3.4.0 and 3.4.1:

- Pull in Solaris `devpoll` support by Giampaolo Rodola (`trollius.selectors` module).
- Add options `-r` and `--randomize` to runtests.py to randomize test order.
- Add a simple echo client/server example.
- Tulip issue #166: Add `__weakref__` slots to `Handle` and `CoroWrapper`.
- `EventLoop.create_unix_server()` now raises a `ValueError` if path and sock are specified at the same time.
- Ensure `call_soon()`, `call_later()` and `call_at()` are invoked on current loop in debug mode. Raise a `RuntimeError` if the event loop of the current thread is different. The check should help to debug thread-safetly issue. Patch written by David Foster.
- Tulip issue #157: Improve test_events.py, avoid `run_briefly()` which is not reliable.
- Reject add/remove reader/writer when event loop is closed.

Bugfixes of Tulip 3.4.1:

- Tulip issue #168: `StreamReader.read(-1)` from pipe may hang if data exceeds buffer limit.
- CPython issue #21447: Fix a race condition in `BaseEventLoop._write_to_self()`.
- Different bugfixes in `CoroWrapper` of `trollius.coroutines`, class used when running Trollius in debug mode:
  - Fix `CoroWrapper` to workaround yield-from bug in CPython 3.4.0. The CPython bug is now fixed in CPython 3.4.1 and 3.5.
  - Make sure `CoroWrapper.send` proxies one argument correctly.
  - CPython issue #21340: Be careful accessing instance variables in `__del__`.
  - Tulip issue #163: Add `gi_{frame,running,code}` properties to `CoroWrapper`.
- Fix `ResourceWarning` warnings
- Tulip issue #159: Fix `windows_utils.socketpair()`. Use `"127.0.0.1"` (IPv4) or `"::1"` (IPv6) host instead of `"localhost"`, because `"localhost"` may be a different IP address. Reject also invalid arguments: only `AF_INET` and `AF_INET6` with `SOCK_STREAM` (and `proto=0`) are supported.

- Tulip issue #158: `Task._step()` now also sets `self` to `None` if an exception is raised. `self` is set to `None` to break a reference cycle.

1.8.10 2014-03-04: version 0.2

Trollius now uses `yield From(...)` syntax which looks close to Tulip `yield from ...` and allows to port more easily Trollius code to Tulip. The usage of `From()` is not mandatory yet, but it may become mandatory in a future version. However, if `yield` is used without `From`, an exception is raised if the event loop is running in debug mode.

Major changes:

- Replace `yield ...` syntax with `yield From(...)`
- On Python 2, Future.set_exception() now only saves the traceback if the debug mode of the event loop is enabled for best performances in production mode. Use `loop.set_debug(True)` to save the traceback.

Bugfixes:

- Fix `BaseEventLoop.default_exception_handler()` on Python 2: get the traceback from `sys.exc_info()`
- Fix unit tests on SSL sockets on Python older than 2.6.6. Example: Mac OS 10.6 with Python 2.6.1 or OpenIndiana 148 with Python 2.6.4.
- Fix error handling in the asyncio.time_monotonic module
- Fix acquire() method of Lock, Condition and Semaphore: don't return a context manager but True, as Tulip. Task._step() now does the trick.

Other changes:

- tox.ini: set PYTHONASYNCIODEBUG to 1 to run tests

1.8.11 2014-02-25: version 0.1.6

Trollius changes:

- Add a new Sphinx documentation: https://trollius.readthedocs.io/
- tox: pass posargs to nosetests. Patch contributed by Ian Wienand.
- Fix support of Python 3.2 and add py32 to tox.ini
- Merge with Tulip 0.4.1

Major changes of Tulip 0.4.1:

- Issue #81: Add support for UNIX Domain Sockets. New APIs:
  - loop.create_unix_connection()
  - loop.create_unix_server()
  - streams.open_unix_connection()
  - streams.start_unix_server()
- Issue #80: Add new event loop exception handling API. New APIs:
  - loop.set_exception_handler()
  - loop.call_exception_handler()

  - loop.default_exception_handler()
- Issue #136: Add get_debug() and set_debug() methods to BaseEventLoopTests. Add also a `PYTHONASYNCIODEBUG` environment variable to debug coroutines since Python startup, to be able to debug coroutines defined directly in the asyncio module.

Other changes of Tulip 0.4.1:

- asyncio.subprocess: Fix a race condition in communicate()
- Fix _ProactorWritePipeTransport._pipe_closed()
- Issue #139: Improve error messages on "fatal errors".
- Issue #140: WriteTransport.set_write_buffer_size() to call _maybe_pause_protocol()
- Issue #129: BaseEventLoop.sock_connect() now raises an error if the address is not resolved (hostname instead of an IP address) for AF_INET and AF_INET6 address families.
- Issue #131: as_completed() and wait() now raises a TypeError if the list of futures is not a list but a Future, Task or coroutine object
- Python issue #20495: Skip test_read_pty_output() of test_asyncio on FreeBSD older than FreeBSD 8
- Issue #130: Add more checks on subprocess_exec/subprocess_shell parameters
- Issue #126: call_soon(), call_soon_threadsafe(), call_later(), call_at() and run_in_executor() now raise a TypeError if the callback is a coroutine function.
- Python issue #20505: BaseEventLoop uses again the resolution of the clock to decide if scheduled tasks should be executed or not.

1.8.12 2014-02-10: version 0.1.5

- Merge with Tulip 0.3.1:
  - New asyncio.subprocess module
  - _UnixWritePipeTransport now also supports character devices, as _UnixReadPipeTransport. Patch written by Jonathan Slenders.
  - StreamReader.readexactly() now raises an IncompleteReadError if the end of stream is reached before we received enough bytes, instead of returning less bytes than requested.
  - poll and epoll selectors now round the timeout away from zero (instead of rounding towards zero) to fix a performance issue
  - asyncio.queue: Empty renamed to QueueEmpty, Full to QueueFull
  - _fatal_error() of _UnixWritePipeTransport and _ProactorBasePipeTransport don't log BrokenPipeError nor ConnectionResetError
  - Future.set_exception(exc) now instanciate exc if it is a class
  - streams.StreamReader: Use bytearray instead of deque of bytes for internal buffer
- Fix test_wait_for() unit test

1.8.13 2014-01-22: version 0.1.4

- The project moved to https://bitbucket.org/enovance/trollius
- Fix CoroWrapper (_DEBUG=True): add missing import

- Emit a warning when Return is not raised
- Merge with Tulip to get latest Tulip bugfixes
- Fix dependencies in tox.ini for the different Python versions

1.8.14 2014-01-13: version 0.1.3

- Workaround bugs in the ssl module of Python older than 2.6.6. For example, Mac OS 10.6 (Snow Leopard) uses Python 2.6.1.
- `return x, y` is now written `raise Return(x, y)` instead of `raise Return((x, y))`
- Support "with (yield lock):" syntax for Lock, Condition and Semaphore
- SSL support is now optional: don't fail if the ssl module is missing
- Add tox.ini, tool to run unit tests. For example, "tox -e py27" creates a virtual environment to run tests with Python 2.7.

1.8.15 2014-01-08: version 0.1.2

- Trollius now supports CPython 2.6-3.4, PyPy and Windows. All unit tests pass with CPython 2.7 on Linux.
- Fix Windows support. Fix compilation of the _overlapped module and add a asyncio._winapi module (written in pure Python). Patch written by Marc Schlaich.
- Support Python 2.6: require an extra dependency, ordereddict (and unittest2 for unit tests)
- Support Python 3.2, 3.3 and 3.4
- Support PyPy 2.2
- Don't modify __builtins__ nor the ssl module to inject backported exceptions like BlockingIOError or SSLWantReadError. Exceptions are available in the asyncio module, ex: asyncio.BlockingIOError.

1.8.16 2014-01-06: version 0.1.1

- Fix asyncio.time_monotonic on Mac OS X
- Fix create_connection(ssl=True)
- Don't export backported SSLContext in the ssl module anymore to not confuse libraries testing hasattr(ssl, "SSLContext")
- Relax dependency on the backported concurrent.futures module: use a synchronous executor if the module is missing

1.8.17 2014-01-04: version 0.1

- First public release

CHAPTER 2

Trollius name

Extract of Trollius Wikipedia article:

Trollius is a genus of about 30 species of plants in the family Ranunculaceae, closely related to Ranunculus. The common name of some species is globeflower or globe flower. Native to the cool temperate regions of the Northern Hemisphere, with the greatest diversity of species in Asia, trollius usually grow in heavy, wet clay soils.