:mod:`threading` --- Thread-based parallelism

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

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 4)

Unknown directive type "module".

```
.. module:: threading
   :synopsis: Thread-based parallelism.
```

Source code: :source:`Lib/threading.py`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 7); *backlink*

Unknown interpreted text role "source".

This module constructs higher-level threading interfaces on top of the lower level :mod:`_thread` module. See also the :mod:`queue` module.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 11); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 11); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 14)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.7
   This module used to be optional, it is now always available.
```

Note

In the Python 2.x series, this module contained `camelCase` names for some methods and functions. These are deprecated as of Python 3.10, but they are still supported for compatibility with Python 2.5 and lower.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 24)

Unknown directive type "impl-detail".

```
.. impl-detail::

   In CPython, due to the :term:`Global Interpreter Lock
   <global interpreter lock>`, only one thread
   can execute Python code at once (even though certain performance-oriented
   libraries might overcome this limitation).
   If you want your application to make better use of the computational
   resources of multi-core machines, you are advised to use
   :mod:`multiprocessing` or :class:`concurrent.futures.ProcessPoolExecutor`.
   However, threading is still an appropriate model if you want to run
   multiple I/O-bound tasks simultaneously.
```

This module defines the following functions:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 40)

Unknown directive type "function".

```
.. function:: active_count()

   Return the number of :class:`Thread` objects currently alive.  The returned
   count is equal to the length of the list returned by :func:`.enumerate`.

   The function ``activeCount`` is a deprecated alias for this function.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 48)

Unknown directive type "function".

```
.. function:: current_thread()

   Return the current :class:`Thread` object, corresponding to the caller's thread
   of control.  If the caller's thread of control was not created through the
   :mod:`threading` module, a dummy thread object with limited functionality is
   returned.

   The function ``currentThread`` is a deprecated alias for this function.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 58)

Unknown directive type "function".

```
.. function:: excepthook(args, /)

   Handle uncaught exception raised by :func:`Thread.run`.

   The *args* argument has the following attributes:

   * *exc_type*: Exception type.
   * *exc_value*: Exception value, can be ``None``.
   * *exc_traceback*: Exception traceback, can be ``None``.
   * *thread*: Thread which raised the exception, can be ``None``.

   If *exc_type* is :exc:`SystemExit`, the exception is silently ignored.
   Otherwise, the exception is printed out on :data:`sys.stderr`.

   If  this function raises an exception, :func:`sys.excepthook` is called to
   handle it.

   :func:`threading.excepthook` can be overridden to control how uncaught
   exceptions raised by :func:`Thread.run` are handled.

   Storing *exc_value* using a custom hook can create a reference cycle. It
   should be cleared explicitly to break the reference cycle when the
   exception is no longer needed.

   Storing *thread* using a custom hook can resurrect it if it is set to an
   object which is being finalized. Avoid storing *thread* after the custom
   hook completes to avoid resurrecting objects.

   .. seealso::
      :func:`sys.excepthook` handles uncaught exceptions.

   .. versionadded:: 3.8
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 91)

Unknown directive type "data".

```
.. data:: __excepthook__

   Holds the original value of :func:`threading.excepthook`. It is saved so that the
   original value can be restored in case they happen to get replaced with
   broken or alternative objects.

   .. versionadded:: 3.10
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 99)

Unknown directive type "function".

```
.. function:: get_ident()

   Return the 'thread identifier' of the current thread.  This is a nonzero
   integer.  Its value has no direct meaning; it is intended as a magic cookie
   to be used e.g. to index a dictionary of thread-specific data.  Thread
   identifiers may be recycled when a thread exits and another thread is
   created.

   .. versionadded:: 3.3
```

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

Unknown directive type "function".

```
.. function:: get_native_id()

   Return the native integral Thread ID of the current thread assigned by the kernel.
   This is a non-negative integer.
   Its value may be used to uniquely identify this particular thread system-wide
   (until the thread terminates, after which the value may be recycled by the OS).

   .. availability:: Windows, FreeBSD, Linux, macOS, OpenBSD, NetBSD, AIX.

   .. versionadded:: 3.8
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 122)

Unknown directive type "function".

```
.. function:: enumerate()

   Return a list of all :class:`Thread` objects currently active.  The list
   includes daemonic threads and dummy thread objects created by
   :func:`current_thread`.  It excludes terminated threads and threads
   that have not yet been started.  However, the main thread is always part
   of the result, even when terminated.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 131)

Unknown directive type "function".

```
.. function:: main_thread()

   Return the main :class:`Thread` object.  In normal conditions, the
   main thread is the thread from which the Python interpreter was
   started.

   .. versionadded:: 3.4
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 140)

Unknown directive type "function".

```
.. function:: settrace(func)

   .. index:: single: trace function

   Set a trace function for all threads started from the :mod:`threading` module.
   The *func* will be passed to  :func:`sys.settrace` for each thread, before its
   :meth:`~Thread.run` method is called.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 149)

Unknown directive type "function".

```
.. function:: gettrace()

   .. index::
      single: trace function
      single: debugger

   Get the trace function as set by :func:`settrace`.

   .. versionadded:: 3.10
```

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

Unknown directive type "function".

```
.. function:: setprofile(func)

   .. index:: single: profile function

   Set a profile function for all threads started from the :mod:`threading` module.
   The *func* will be passed to  :func:`sys.setprofile` for each thread, before its
   :meth:`~Thread.run` method is called.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 169)

Unknown directive type "function".

```
.. function:: getprofile()

   .. index:: single: profile function

   Get the profiler function as set by :func:`setprofile`.

   .. versionadded:: 3.10
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 178)

Unknown directive type "function".

```
.. function:: stack_size([size])

   Return the thread stack size used when creating new threads.  The optional
   *size* argument specifies the stack size to be used for subsequently created
   threads, and must be 0 (use platform or configured default) or a positive
   integer value of at least 32,768 (32 KiB). If *size* is not specified,
   0 is used.  If changing the thread stack size is
   unsupported, a :exc:`RuntimeError` is raised.  If the specified stack size is
   invalid, a :exc:`ValueError` is raised and the stack size is unmodified.  32 KiB
   is currently the minimum supported stack size value to guarantee sufficient
   stack space for the interpreter itself.  Note that some platforms may have
   particular restrictions on values for the stack size, such as requiring a
   minimum stack size > 32 KiB or requiring allocation in multiples of the system
```

```
memory page size - platform documentation should be referred to for more
information (4 KiB pages are common; using multiples of 4096 for the stack size is
the suggested approach in the absence of more specific information).

.. availability:: Windows, systems with POSIX threads.
```

This module also defines the following constant:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 200)

Unknown directive type "data".

```
.. data:: TIMEOUT_MAX

   The maximum value allowed for the *timeout* parameter of blocking functions
   (:meth:`Lock.acquire`, :meth:`RLock.acquire`, :meth:`Condition.wait`, etc.).
   Specifying a timeout greater than this value will raise an
   :exc:`OverflowError`.

   .. versionadded:: 3.2
```

This module defines a number of classes, which are detailed in the sections below.

The design of this module is loosely based on Java's threading model. However, where Java makes locks and condition variables basic behavior of every object, they are separate objects in Python. Python's :class:`Thread` class supports a subset of the behavior of Java's Thread class; currently, there are no priorities, no thread groups, and threads cannot be destroyed, stopped, suspended, resumed, or interrupted. The static methods of Java's Thread class, when implemented, are mapped to module-level functions.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 213); *backlink*

Unknown interpreted text role "class".

All of the methods described below are executed atomically.

Thread-Local Data

Thread-local data is data whose values are thread specific. To manage thread-local data, just create an instance of :class:`local` (or a subclass) and store attributes on it:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 227); *backlink*

Unknown interpreted text role "class".

```
mydata = threading.local()
mydata.x = 1
```

The instance's values will be different for separate threads.

A class that represents thread-local data.

For more details and extensive examples, see the documentation string of the :mod:`_threading_local` module.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 241); *backlink*

Unknown interpreted text role "mod".

Thread Objects

The :class:`Thread` class represents an activity that is run in a separate thread of control. There are two ways to specify the activity: by passing a callable object to the constructor, or by overriding the :meth:`~Thread.run` method in a subclass. No other methods (except for the constructor) should be overridden in a subclass. In other words, *only* override the :meth:`~Thread.__init__` and :meth:`~Thread.run` methods of this class.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 250); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 250); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 250); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 250); *backlink*

Unknown interpreted text role "meth".

Once a thread object is created, its activity must be started by calling the thread's :meth:`~Thread.start` method. This invokes the :meth:`~Thread.run` method in a separate thread of control.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 257); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 257); *backlink*

Unknown interpreted text role "meth".

Once the thread's activity is started, the thread is considered 'alive'. It stops being alive when its :meth:`~Thread.run` method terminates -- either normally, or by raising an unhandled exception. The :meth:`~Thread.is_alive` method tests whether the thread is alive.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 261); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 261); *backlink*

Unknown interpreted text role "meth".

Other threads can call a thread's :meth:`~Thread.join` method. This blocks the calling thread until the thread whose :meth:`~Thread.join` method is called is terminated.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 266); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 266); *backlink*

Unknown interpreted text role "meth".

A thread has a name. The name can be passed to the constructor, and read or changed through the :attr:`~Thread.name` attribute.

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

Unknown interpreted text role "attr".

If the :meth:`~Thread.run` method raises an exception, :func:`threading.excepthook` is called to handle it. By default, :func:`threading.excepthook` ignores silently :exc:`SystemExit`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 273); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 273); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 273); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 273); *backlink*

Unknown interpreted text role "exc".

A thread can be flagged as a "daemon thread". The significance of this flag is that the entire Python program exits when only daemon threads are left. The initial value is inherited from the creating thread. The flag can be set through the :attr:`~Thread.daemon` property or the *daemon* constructor argument.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 277); *backlink*

Unknown interpreted text role "attr".

Note

Daemon threads are abruptly stopped at shutdown. Their resources (such as open files, database transactions, etc.) may not be released properly. If you want your threads to stop gracefully, make them non-daemonic and use a suitable signalling mechanism such as an :class:`Event`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 284); *backlink*

Unknown interpreted text role "class".

There is a "main thread" object; this corresponds to the initial thread of control in the Python program. It is not a daemon thread.

There is the possibility that "dummy thread objects" are created. These are thread objects corresponding to "alien threads", which are threads of control started outside the threading module, such as directly from C code. Dummy thread objects have limited functionality; they are always considered alive and daemonic, and cannot be :meth:`~Thread.join`ed. They are never deleted, since it is impossible to detect the termination of alien threads.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 292); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 300)

Invalid class attribute value for "class" directive: "Thread(group=None, target=None, name=None, args=(), kwargs={}, *, \ daemon=None)".

```
.. class:: Thread(group=None, target=None, name=None, args=(), kwargs={}, *, \
```

```
                daemon=None)

      This constructor should always be called with keyword arguments.  Arguments
      are:

      *group* should be ``None``; reserved for future extension when a
      :class:`ThreadGroup` class is implemented.

      *target* is the callable object to be invoked by the :meth:`run` method.
      Defaults to ``None``, meaning nothing is called.

      *name* is the thread name. By default, a unique name is constructed
      of the form "Thread-*N*" where *N* is a small decimal number,
      or "Thread-*N* (target)" where "target" is ``target.__name__`` if the
      *target* argument is specified.

      *args* is a list or tuple of arguments for the target invocation.  Defaults to ``()``.

      *kwargs* is a dictionary of keyword arguments for the target invocation.
      Defaults to ``{}``.

      If not ``None``, *daemon* explicitly sets whether the thread is daemonic.
      If ``None`` (the default), the daemonic property is inherited from the
      current thread.

      If the subclass overrides the constructor, it must make sure to invoke the
      base class constructor (``Thread.__init__()``) before doing anything else to
      the thread.

      .. versionchanged:: 3.10
         Use the *target* name if *name* argument is omitted.

      .. versionchanged:: 3.3
         Added the *daemon* argument.

      .. method:: start()

         Start the thread's activity.

         It must be called at most once per thread object.  It arranges for the
         object's :meth:`~Thread.run` method to be invoked in a separate thread
         of control.

         This method will raise a :exc:`RuntimeError` if called more than once
         on the same thread object.

      .. method:: run()

         Method representing the thread's activity.

         You may override this method in a subclass.  The standard :meth:`run`
         method invokes the callable object passed to the object's constructor as
         the *target* argument, if any, with positional and keyword arguments taken
         from the *args* and *kwargs* arguments, respectively.

         Using list or tuple as the *args* argument which passed to the :class:`Thread`
         could achieve the same effect.

         Example::

            >>> from threading import Thread
            >>> t = Thread(target=print, args=[1])
            >>> t.run()
            1
            >>> t = Thread(target=print, args=(1,))
            >>> t.run()
            1

      .. method:: join(timeout=None)

         Wait until the thread terminates. This blocks the calling thread until
         the thread whose :meth:`~Thread.join` method is called terminates -- either
         normally or through an unhandled exception -- or until the optional
         timeout occurs.

         When the *timeout* argument is present and not ``None``, it should be a
         floating point number specifying a timeout for the operation in seconds
         (or fractions thereof). As :meth:`~Thread.join` always returns ``None``,
         you must call :meth:`~Thread.is_alive` after :meth:`~Thread.join` to
         decide whether a timeout happened -- if the thread is still alive, the
         :meth:`~Thread.join` call timed out.
```

```
   When the *timeout* argument is not present or ``None``, the operation will
   block until the thread terminates.

   A thread can be :meth:`~Thread.join`\ ed many times.

   :meth:`~Thread.join` raises a :exc:`RuntimeError` if an attempt is made
   to join the current thread as that would cause a deadlock. It is also
   an error to :meth:`~Thread.join` a thread before it has been started
   and attempts to do so raise the same exception.

.. attribute:: name

   A string used for identification purposes only. It has no semantics.
   Multiple threads may be given the same name.  The initial name is set by
   the constructor.

.. method:: getName()
            setName()

   Deprecated getter/setter API for :attr:`~Thread.name`; use it directly as a
   property instead.

   .. deprecated:: 3.10

.. attribute:: ident

   The 'thread identifier' of this thread or ``None`` if the thread has not
   been started.  This is a nonzero integer.  See the :func:`get_ident`
   function.  Thread identifiers may be recycled when a thread exits and
   another thread is created.  The identifier is available even after the
   thread has exited.

.. attribute:: native_id

   The Thread ID (``TID``) of this thread, as assigned by the OS (kernel).
   This is a non-negative integer, or ``None`` if the thread has not
   been started. See the :func:`get_native_id` function.
   This value may be used to uniquely identify this particular thread
   system-wide (until the thread terminates, after which the value
   may be recycled by the OS).

   .. note::

      Similar to Process IDs, Thread IDs are only valid (guaranteed unique
      system-wide) from the time the thread is created until the thread
      has been terminated.

   .. availability:: Requires :func:`get_native_id` function.

   .. versionadded:: 3.8

.. method:: is_alive()

   Return whether the thread is alive.

   This method returns ``True`` just before the :meth:`~Thread.run` method
   starts until just after the :meth:`~Thread.run` method terminates.  The
   module function :func:`.enumerate` returns a list of all alive threads.

.. attribute:: daemon

   A boolean value indicating whether this thread is a daemon thread (``True``)
   or not (``False``).  This must be set before :meth:`~Thread.start` is called,
   otherwise :exc:`RuntimeError` is raised.  Its initial value is inherited
   from the creating thread; the main thread is not a daemon thread and
   therefore all threads created in the main thread default to
   :attr:`~Thread.daemon` = ``False``.

   The entire Python program exits when no alive non-daemon threads are left.

.. method:: isDaemon()
            setDaemon()

   Deprecated getter/setter API for :attr:`~Thread.daemon`; use it directly as a
   property instead.

   .. deprecated:: 3.10
```

Lock Objects

A primitive lock is a synchronization primitive that is not owned by a particular thread when locked. In Python, it is currently the lowest level synchronization primitive available, implemented directly by the :mod:`_thread` extension module.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst`, line 467); *backlink*

Unknown interpreted text role "mod".

A primitive lock is in one of two states, "locked" or "unlocked". It is created in the unlocked state. It has two basic methods, :meth:`~Lock.acquire` and :meth:`~Lock.release`. When the state is unlocked, :meth:`~Lock.acquire` changes the state to locked and returns immediately. When the state is locked, :meth:`~Lock.acquire` blocks until a call to :meth:`~Lock.release` in another thread changes it to unlocked, then the :meth:`~Lock.acquire` call resets it to locked and returns. The :meth:`~Lock.release` method should only be called in the locked state; it changes the state to unlocked and returns immediately. If an attempt is made to release an unlocked lock, a :exc:`RuntimeError` will be raised.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst`, line 472); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst`, line 472); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst`, line 472); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst`, line 472); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst`, line 472); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst`, line 472); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst`, line 472); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst`, line 472); *backlink*

Unknown interpreted text role "exc".

Locks also support the :ref:`context management protocol <with-locks>`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst`, line 483); *backlink*

Unknown interpreted text role "ref".

When more than one thread is blocked in :meth:`~Lock.acquire` waiting for the state to turn to unlocked, only one thread proceeds when a :meth:`~Lock.release` call resets the state to unlocked; which one of the waiting threads proceeds is not defined, and may vary across implementations.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 485); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 485); *backlink*

Unknown interpreted text role "meth".

All methods are executed atomically.

The class implementing primitive lock objects. Once a thread has acquired a lock, subsequent attempts to acquire it block, until it is released; any thread may release it.

Note that `Lock` is actually a factory function which returns an instance of the most efficient version of the concrete Lock class that is supported by the platform.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 504)

Unknown directive type "method".

```
.. method:: acquire(blocking=True, timeout=-1)

   Acquire a lock, blocking or non-blocking.

   When invoked with the *blocking* argument set to ``True`` (the default),
   block until the lock is unlocked, then set it to locked and return ``True``.

   When invoked with the *blocking* argument set to ``False``, do not block.
   If a call with *blocking* set to ``True`` would block, return ``False``
   immediately; otherwise, set the lock to locked and return ``True``.

   When invoked with the floating-point *timeout* argument set to a positive
   value, block for at most the number of seconds specified by *timeout*
   and as long as the lock cannot be acquired.  A *timeout* argument of ``-1``
   specifies an unbounded wait.  It is forbidden to specify a *timeout*
   when *blocking* is ``False``.

   The return value is ``True`` if the lock is acquired successfully,
   ``False`` if not (for example if the *timeout* expired).

   .. versionchanged:: 3.2
      The *timeout* parameter is new.

   .. versionchanged:: 3.2
      Lock acquisition can now be interrupted by signals on POSIX if the
      underlying threading implementation supports it.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 532)

Unknown directive type "method".

```
.. method:: release()

   Release a lock.  This can be called from any thread, not only the thread
   which has acquired the lock.

   When the lock is locked, reset it to unlocked, and return.  If any other threads
   are blocked waiting for the lock to become unlocked, allow exactly one of them
   to proceed.

   When invoked on an unlocked lock, a :exc:`RuntimeError` is raised.

   There is no return value.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst`, line 545)

Unknown directive type "method".

```
.. method:: locked()

   Return ``True`` if the lock is acquired.
```

RLock Objects

A reentrant lock is a synchronization primitive that may be acquired multiple times by the same thread. Internally, it uses the concepts of "owning thread" and "recursion level" in addition to the locked/unlocked state used by primitive locks. In the locked state, some thread owns the lock; in the unlocked state, no thread owns it.

To lock the lock, a thread calls its :meth:`~RLock.acquire` method; this returns once the thread owns the lock. To unlock the lock, a thread calls its :meth:`~Lock.release` method. :meth:`~Lock.acquire`/:meth:`~Lock.release` call pairs may be nested; only the final :meth:`~Lock.release` (the :meth:`~Lock.release` of the outermost pair) resets the lock to unlocked and allows another thread blocked in :meth:`~Lock.acquire` to proceed.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst`, line 562); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst`, line 562); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst`, line 562); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst`, line 562); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst`, line 562); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst`, line 562); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst`, line 562); *backlink*

Unknown interpreted text role "meth".

Reentrant locks also support the :ref:`context management protocol <with-locks>`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst`, line 569); *backlink*

Unknown interpreted text role "ref".

This class implements reentrant lock objects. A reentrant lock must be released by the thread that acquired it. Once a thread has acquired a reentrant lock, the same thread may acquire it again without blocking; the thread must release it once for each time it has acquired it.

Note that `RLock` is actually a factory function which returns an instance of the most efficient version of the concrete RLock class that is supported by the platform.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 584)

Unknown directive type "method".

```
.. method:: acquire(blocking=True, timeout=-1)

   Acquire a lock, blocking or non-blocking.

   When invoked without arguments: if this thread already owns the lock, increment
   the recursion level by one, and return immediately.  Otherwise, if another
   thread owns the lock, block until the lock is unlocked.  Once the lock is
   unlocked (not owned by any thread), then grab ownership, set the recursion level
   to one, and return.  If more than one thread is blocked waiting until the lock
   is unlocked, only one at a time will be able to grab ownership of the lock.
   There is no return value in this case.

   When invoked with the *blocking* argument set to ``True``, do the same thing as when
   called without arguments, and return ``True``.

   When invoked with the *blocking* argument set to ``False``, do not block.  If a call
   without an argument would block, return ``False`` immediately; otherwise, do the
   same thing as when called without arguments, and return ``True``.

   When invoked with the floating-point *timeout* argument set to a positive
   value, block for at most the number of seconds specified by *timeout*
   and as long as the lock cannot be acquired.  Return ``True`` if the lock has
   been acquired, ``False`` if the timeout has elapsed.

   .. versionchanged:: 3.2
      The *timeout* parameter is new.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 612)

Unknown directive type "method".

```
.. method:: release()

   Release a lock, decrementing the recursion level.  If after the decrement it is
   zero, reset the lock to unlocked (not owned by any thread), and if any other
   threads are blocked waiting for the lock to become unlocked, allow exactly one
   of them to proceed.  If after the decrement the recursion level is still
   nonzero, the lock remains locked and owned by the calling thread.

   Only call this method when the calling thread owns the lock. A
   :exc:`RuntimeError` is raised if this method is called when the lock is
   unlocked.

   There is no return value.
```

Condition Objects

A condition variable is always associated with some kind of lock; this can be passed in or one will be created by default. Passing one in is useful when several condition variables must share the same lock. The lock is part of the condition object: you don't have to track it separately.

A condition variable obeys the :ref:`context management protocol <with-locks>`: using the `with` statement acquires the associated lock for the duration of the enclosed block. The :meth:`~Condition.acquire` and :meth:`~Condition.release` methods also call the corresponding methods of the associated lock.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 637); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 637); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst`, line 637); *backlink*

Unknown interpreted text role "meth".

Other methods must be called with the associated lock held. The :meth:`~Condition.wait` method releases the lock, and then blocks until another thread awakens it by calling :meth:`~Condition.notify` or :meth:`~Condition.notify_all`. Once awakened, :meth:`~Condition.wait` re-acquires the lock and returns. It is also possible to specify a timeout.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst`, line 643); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst`, line 643); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst`, line 643); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst`, line 643); *backlink*

Unknown interpreted text role "meth".

The :meth:`~Condition.notify` method wakes up one of the threads waiting for the condition variable, if any are waiting. The :meth:`~Condition.notify_all` method wakes up all threads waiting for the condition variable.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst`, line 649); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst`, line 649); *backlink*

Unknown interpreted text role "meth".

Note: the :meth:`~Condition.notify` and :meth:`~Condition.notify_all` methods don't release the lock; this means that the thread or threads awakened will not return from their :meth:`~Condition.wait` call immediately, but only when the thread that called :meth:`~Condition.notify` or :meth:`~Condition.notify_all` finally relinquishes ownership of the lock.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst`, line 653); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst`, line 653); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst`, line 653); *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]threading.rst, line 653); backlink

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 653); backlink

Unknown interpreted text role "meth".

The typical programming style using condition variables uses the lock to synchronize access to some shared state; threads that are interested in a particular change of state call :meth:`~Condition.wait` repeatedly until they see the desired state, while threads that modify the state call :meth:`~Condition.notify` or :meth:`~Condition.notify_all` when they change the state in such a way that it could possibly be a desired state for one of the waiters. For example, the following code is a generic producer-consumer situation with unlimited buffer capacity:

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

Unknown interpreted text role "meth".

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

Unknown interpreted text role "meth".

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

Unknown interpreted text role "meth".

```
# Consume one item
with cv:
    while not an_item_is_available():
        cv.wait()
    get_an_available_item()

# Produce one item
with cv:
    make_an_item_available()
    cv.notify()
```

The `while` loop checking for the application's condition is necessary because :meth:`~Condition.wait` can return after an arbitrary long time, and the condition which prompted the :meth:`~Condition.notify` call may no longer hold true. This is inherent to multi-threaded programming. The :meth:`~Condition.wait_for` method can be used to automate the condition checking, and eases the computation of timeouts:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 679); backlink

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 679); backlink

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 679); backlink

Unknown interpreted text role "meth".

```
# Consume an item
with cv:
    cv.wait_for(an_item_is_available)
    get_an_available_item()
```

To choose between :meth:`~Condition.notify` and :meth:`~Condition.notify_all`, consider whether one state change can be interesting for only one or several waiting threads. E.g. in a typical producer-consumer situation, adding one item to the buffer only needs to wake up one consumer thread.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 691); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 691); *backlink*

Unknown interpreted text role "meth".

This class implements condition variable objects. A condition variable allows one or more threads to wait until they are notified by another thread.

If the *lock* argument is given and not `None`, it must be a :class:`Lock` or :class:`RLock` object, and it is used as the underlying lock. Otherwise, a new :class:`RLock` object is created and used as the underlying lock.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 702); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 702); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 702); *backlink*

Unknown interpreted text role "class".

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

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.3
   changed from a factory function to a class.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 709)

Unknown directive type "method".

```
.. method:: acquire(*args)

   Acquire the underlying lock. This method calls the corresponding method on
   the underlying lock; the return value is whatever that method returns.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 714)

Unknown directive type "method".

```
.. method:: release()

   Release the underlying lock. This method calls the corresponding method on
   the underlying lock; there is no return value.
```

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

Unknown directive type "method".

```
.. method:: wait(timeout=None)
```

```
   Wait until notified or until a timeout occurs. If the calling thread has
   not acquired the lock when this method is called, a :exc:`RuntimeError` is
   raised.

   This method releases the underlying lock, and then blocks until it is
   awakened by a :meth:`notify` or :meth:`notify_all` call for the same
   condition variable in another thread, or until the optional timeout
   occurs.  Once awakened or timed out, it re-acquires the lock and returns.

   When the *timeout* argument is present and not ``None``, it should be a
   floating point number specifying a timeout for the operation in seconds
   (or fractions thereof).

   When the underlying lock is an :class:`RLock`, it is not released using
   its :meth:`release` method, since this may not actually unlock the lock
   when it was acquired multiple times recursively.  Instead, an internal
   interface of the :class:`RLock` class is used, which really unlocks it
   even when it has been recursively acquired several times. Another internal
   interface is then used to restore the recursion level when the lock is
   reacquired.

   The return value is ``True`` unless a given *timeout* expired, in which
   case it is ``False``.

   .. versionchanged:: 3.2
      Previously, the method always returned ``None``.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 748)

Unknown directive type "method".

```
.. method:: wait_for(predicate, timeout=None)

   Wait until a condition evaluates to true.  *predicate* should be a
   callable which result will be interpreted as a boolean value.
   A *timeout* may be provided giving the maximum time to wait.

   This utility method may call :meth:`wait` repeatedly until the predicate
   is satisfied, or until a timeout occurs. The return value is
   the last return value of the predicate and will evaluate to
   ``False`` if the method timed out.

   Ignoring the timeout feature, calling this method is roughly equivalent to
   writing::

     while not predicate():
         cv.wait()

   Therefore, the same rules apply as with :meth:`wait`: The lock must be
   held when called and is re-acquired on return.  The predicate is evaluated
   with the lock held.

   .. versionadded:: 3.2
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 771)

Unknown directive type "method".

```
.. method:: notify(n=1)

   By default, wake up one thread waiting on this condition, if any.  If the
   calling thread has not acquired the lock when this method is called, a
   :exc:`RuntimeError` is raised.

   This method wakes up at most *n* of the threads waiting for the condition
   variable; it is a no-op if no threads are waiting.

   The current implementation wakes up exactly *n* threads, if at least *n*
   threads are waiting.  However, it's not safe to rely on this behavior.
   A future, optimized implementation may occasionally wake up more than
   *n* threads.

   Note: an awakened thread does not actually return from its :meth:`wait`
   call until it can reacquire the lock.  Since :meth:`notify` does not
   release the lock, its caller should.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 789)

Unknown directive type "method".

```
.. method:: notify_all()

   Wake up all threads waiting on this condition.  This method acts like
   :meth:`notify`, but wakes up all waiting threads instead of one. If the
   calling thread has not acquired the lock when this method is called, a
   :exc:`RuntimeError` is raised.

   The method ``notifyAll`` is a deprecated alias for this method.
```

Semaphore Objects

This is one of the oldest synchronization primitives in the history of computer science, invented by the early Dutch computer scientist Edsger W. Dijkstra (he used the names `P()` and `V()` instead of :meth:`~Semaphore.acquire` and :meth:`~Semaphore.release`).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 804); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 804); *backlink*

Unknown interpreted text role "meth".

A semaphore manages an internal counter which is decremented by each :meth:`~Semaphore.acquire` call and incremented by each :meth:`~Semaphore.release` call. The counter can never go below zero; when :meth:`~Semaphore.acquire` finds that it is zero, it blocks, waiting until some other thread calls :meth:`~Semaphore.release`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 809); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 809); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 809); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 809); *backlink*

Unknown interpreted text role "meth".

Semaphores also support the :ref:`context management protocol <with-locks>`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 815); *backlink*

Unknown interpreted text role "ref".

This class implements semaphore objects. A semaphore manages an atomic counter representing the number of :meth:`release` calls minus the number of :meth:`acquire` calls, plus an initial value. The :meth:`acquire` method blocks if necessary until it can return without making the counter negative. If not given, *value* defaults to 1.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 820); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 820); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 820); *backlink*

Unknown interpreted text role "meth".

The optional argument gives the initial *value* for the internal counter; it defaults to `1`. If the *value* given is less than 0, :exc:`ValueError` is raised.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 826); *backlink*

Unknown interpreted text role "exc".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 830)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.3
   changed from a factory function to a class.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 833)

Unknown directive type "method".

```
.. method:: acquire(blocking=True, timeout=None)

   Acquire a semaphore.

   When invoked without arguments:

   * If the internal counter is larger than zero on entry, decrement it by
     one and return ``True`` immediately.
   * If the internal counter is zero on entry, block until awoken by a call to
     :meth:`~Semaphore.release`.  Once awoken (and the counter is greater
     than 0), decrement the counter by 1 and return ``True``.  Exactly one
     thread will be awoken by each call to :meth:`~Semaphore.release`.  The
     order in which threads are awoken should not be relied on.

   When invoked with *blocking* set to ``False``, do not block.  If a call
   without an argument would block, return ``False`` immediately; otherwise, do
   the same thing as when called without arguments, and return ``True``.

   When invoked with a *timeout* other than ``None``, it will block for at
   most *timeout* seconds.  If acquire does not complete successfully in
   that interval, return ``False``.  Return ``True`` otherwise.

   .. versionchanged:: 3.2
      The *timeout* parameter is new.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 858)

Unknown directive type "method".

```
.. method:: release(n=1)

   Release a semaphore, incrementing the internal counter by *n*.  When it
   was zero on entry and other threads are waiting for it to become larger
   than zero again, wake up *n* of those threads.
```

```
.. versionchanged:: 3.9
   Added the *n* parameter to release multiple waiting threads at once.
```

Class implementing bounded semaphore objects. A bounded semaphore checks to make sure its current value doesn't exceed its initial value. If it does, :exc:`ValueError` is raised. In most situations semaphores are used to guard resources with limited capacity. If the semaphore is released too many times it's a sign of a bug. If not given, *value* defaults to 1.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 870); *backlink*

Unknown interpreted text role "exc".

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

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.3
   changed from a factory function to a class.
```

:class:`Semaphore` Example

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 882); *backlink*

Unknown interpreted text role "class".

Semaphores are often used to guard resources with limited capacity, for example, a database server. In any situation where the size of the resource is fixed, you should use a bounded semaphore. Before spawning any worker threads, your main thread would initialize the semaphore:

```
maxconnections = 5
# ...
pool_sema = BoundedSemaphore(value=maxconnections)
```

Once spawned, worker threads call the semaphore's acquire and release methods when they need to connect to the server:

```
with pool_sema:
    conn = connectdb()
    try:
        # ... use connection ...
    finally:
        conn.close()
```

The use of a bounded semaphore reduces the chance that a programming error which causes the semaphore to be released more than it's acquired will go undetected.

Event Objects

This is one of the simplest mechanisms for communication between threads: one thread signals an event and other threads wait for it.

An event object manages an internal flag that can be set to true with the :meth:`~Event.set` method and reset to false with the :meth:`~Event.clear` method. The :meth:`~Event.wait` method blocks until the flag is true.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 916); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 916); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 916); *backlink*

Unknown interpreted text role "meth".

Class implementing event objects. An event manages a flag that can be set to true with the :meth:`~Event.set` method and reset to false with the :meth:`clear` method. The :meth:`wait` method blocks until the flag is true. The flag is initially false.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 923); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 923); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 923); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 928)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.3
   changed from a factory function to a class.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 931)

Unknown directive type "method".

```
.. method:: is_set()

   Return ``True`` if and only if the internal flag is true.

   The method ``isSet`` is a deprecated alias for this method.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 937)

Unknown directive type "method".

```
.. method:: set()

   Set the internal flag to true. All threads waiting for it to become true
   are awakened. Threads that call :meth:`wait` once the flag is true will
   not block at all.
```

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

Unknown directive type "method".

```
.. method:: clear()

   Reset the internal flag to false. Subsequently, threads calling
   :meth:`wait` will block until :meth:`.set` is called to set the internal
   flag to true again.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 949)

Unknown directive type "method".

```
.. method:: wait(timeout=None)
```

```
Block until the internal flag is true.  If the internal flag is true on
entry, return immediately.  Otherwise, block until another thread calls
:meth:`.set` to set the flag to true, or until the optional timeout occurs.

When the timeout argument is present and not ``None``, it should be a
floating point number specifying a timeout for the operation in seconds
(or fractions thereof).

This method returns ``True`` if and only if the internal flag has been set to
true, either before the wait call or after the wait starts, so it will
always return ``True`` except if a timeout is given and the operation
times out.

.. versionchanged:: 3.1
   Previously, the method always returned ``None``.
```

Timer Objects

This class represents an action that should be run only after a certain amount of time has passed --- a timer. :class:`Timer` is a subclass of :class:`Thread` and as such also functions as an example of creating custom threads.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 973); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 973); *backlink*

Unknown interpreted text role "class".

Timers are started, as with threads, by calling their :meth:`~Timer.start` method. The timer can be stopped (before its action has begun) by calling the :meth:`~Timer.cancel` method. The interval the timer will wait before executing its action may not be exactly the same as the interval specified by the user.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 977); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 977); *backlink*

Unknown interpreted text role "meth".

For example:

```
def hello():
    print("hello, world")

t = Timer(30.0, hello)
t.start()  # after 30 seconds, "hello, world" will be printed
```

Create a timer that will run *function* with arguments *args* and keyword arguments *kwargs*, after *interval* seconds have passed. If *args* is `None` (the default) then an empty list will be used. If *kwargs* is `None` (the default) then an empty dict will be used.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 999)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.3
   changed from a factory function to a class.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 1002)

Unknown directive type "method".

```
.. method:: cancel()

   Stop the timer, and cancel the execution of the timer's action.  This will
   only work if the timer is still in its waiting stage.
```

Barrier Objects

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 1011)

Unknown directive type "versionadded".

```
.. versionadded:: 3.2
```

This class provides a simple synchronization primitive for use by a fixed number of threads that need to wait for each other. Each of the threads tries to pass the barrier by calling the :meth:`~Barrier.wait` method and will block until all of the threads have made their :meth:`~Barrier.wait` calls. At this point, the threads are released simultaneously.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 1013); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 1013); *backlink*

Unknown interpreted text role "meth".

The barrier can be reused any number of times for the same number of threads.

As an example, here is a simple way to synchronize a client and server thread:

```
b = Barrier(2, timeout=5)

def server():
    start_server()
    b.wait()
    while True:
        connection = accept_connection()
        process_server_connection(connection)

def client():
    b.wait()
    while True:
        connection = make_connection()
        process_client_connection(connection)
```

Create a barrier object for *parties* number of threads. An *action*, when provided, is a callable to be called by one of the threads when they are released. *timeout* is the default timeout value if none is specified for the :meth:`wait` method.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 1041); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 1046)

Unknown directive type "method".

```
.. method:: wait(timeout=None)

   Pass the barrier.  When all the threads party to the barrier have called
   this function, they are all released simultaneously.  If a *timeout* is
   provided, it is used in preference to any that was supplied to the class
   constructor.

   The return value is an integer in the range 0 to *parties* -- 1, different
   for each thread.  This can be used to select a thread to do some special
   housekeeping, e.g.::
```

```
        i = barrier.wait()
        if i == 0:
            # Only one thread needs to print this
            print("passed the barrier")

    If an *action* was provided to the constructor, one of the threads will
    have called it prior to being released.  Should this call raise an error,
    the barrier is put into the broken state.

    If the call times out, the barrier is put into the broken state.

    This method may raise a :class:`BrokenBarrierError` exception if the
    barrier is broken or reset while a thread is waiting.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 1071)

Unknown directive type "method".

```
.. method:: reset()

   Return the barrier to the default, empty state.  Any threads waiting on it
   will receive the :class:`BrokenBarrierError` exception.

   Note that using this function may require some external
   synchronization if there are other threads whose state is unknown.  If a
   barrier is broken it may be better to just leave it and create a new one.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 1080)

Unknown directive type "method".

```
.. method:: abort()

   Put the barrier into a broken state.  This causes any active or future
   calls to :meth:`wait` to fail with the :class:`BrokenBarrierError`.  Use
   this for example if one of the threads needs to abort, to avoid deadlocking the
   application.

   It may be preferable to simply create the barrier with a sensible
   *timeout* value to automatically guard against one of the threads going
   awry.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 1091)

Unknown directive type "attribute".

```
.. attribute:: parties

   The number of threads required to pass the barrier.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 1095)

Unknown directive type "attribute".

```
.. attribute:: n_waiting

   The number of threads currently waiting in the barrier.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 1099)

Unknown directive type "attribute".

```
.. attribute:: broken

   A boolean that is ``True`` if the barrier is in the broken state.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 1104)

Unknown directive type "exception".

```
.. exception:: BrokenBarrierError

   This exception, a subclass of :exc:`RuntimeError`, is raised when the
   :class:`Barrier` object is reset or broken.
```

Using locks, conditions, and semaphores in the :keyword:`!with` statement

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 1112); *backlink*

Unknown interpreted text role "keyword".

All of the objects provided by this module that have :meth:`acquire` and :meth:`release` methods can be used as context managers for a :keyword:`with` statement. The :meth:`acquire` method will be called when the block is entered, and :meth:`release` will be called when the block is exited. Hence, the following snippet:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 1115); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 1115); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 1115); *backlink*

Unknown interpreted text role "keyword".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 1115); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 1115); *backlink*

Unknown interpreted text role "meth".

```
with some_lock:
    # do something...
```

is equivalent to:

```
some_lock.acquire()
try:
    # do something...
finally:
    some_lock.release()
```

Currently, :class:`Lock`, :class:`RLock`, :class:`Condition`, :class:`Semaphore`, and :class:`BoundedSemaphore` objects may be used as :keyword:`with` statement context managers.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 1132); *backlink*

Unknown interpreted text role "class".

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

main\Doc\library\[cpython-main][Doc][library]threading.rst, line 1132); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 1132); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 1132); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 1132); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]threading.rst, line 1132); *backlink*

Unknown interpreted text role "keyword".