
watchdog Documentation

Release 2.1.3

Yesudeep Mangalapilly and contributors

Aug 24, 2021

Contents

1	Directory monitoring made easy with	3
2	Easy installation	5
3	User's Guide	7
3.1	Installation	7
3.2	Quickstart	10
3.3	API Reference	10
3.4	Contributing	23
4	Contribute	25
5	Indices and tables	27
	Python Module Index	29
	Index	31

Python API library and shell utilities to monitor file system events.

Works on 3.6+.

If you want to use Python 2.6, you should stick with watchdog < 0.10.0.

If you want to use Python 2.7, 3.4 or 3.5, you should stick with watchdog < 1.0.0.

Directory monitoring made easy with

- A cross-platform API.
- A shell tool to run commands in response to directory changes.

Get started quickly with a simple example in [Quickstart](#).

CHAPTER 2

Easy installation

You can use `pip` to install watchdog quickly and easily:

```
$ python -m pip install -U watchdog
```

Need more help with installing? See [Installation](#).

3.1 Installation

watchdog requires 3.6+ to work. See a list of *Dependencies*.

3.1.1 Installing from PyPI using pip

```
$ python -m pip install -U watchdog

# or to install the watchmedo utility:
$ python -m pip install -U |project_name|[watchmedo]
```

3.1.2 Installing from source tarballs

```
$ wget -c https://pypi.python.org/packages/source/w/watchdog/watchdog-2.1.3.
→tar.gz
$ tar zxvf watchdog-2.1.3.tar.gz
$ cd watchdog-2.1.3
$ python -m pip install -e .

# or to install the watchmedo utility:
$ python -m pip install -e ".[watchmedo]"
```

3.1.3 Installing from the code repository

```
$ git clone --recursive git://github.com/gorakhargosh/watchdog.git
$ cd watchdog
$ python -m pip install -e .
```

(continues on next page)

(continued from previous page)

```
# or to install the watchmedo utility:
$ python -m pip install -e ".[watchmedo]"
```

3.1.4 Dependencies

watchdog depends on many libraries to do its job. The following is a list of dependencies you need based on the operating system you are using.

Operating system Dependency (row)	Windows	Linux 2.6	Mac OS X/ Darwin	BSD
XCode			Yes	

The following is a list of dependencies you need based on the operating system you are using the watchmedo utility.

Operating system Dependency (row)	Windows	Linux 2.6	Mac OS X/ Darwin	BSD
PyYAML	Yes	Yes	Yes	Yes
argh	Yes	Yes	Yes	Yes

Installing Dependencies

The watchmedo script depends on PyYAML which links with LibYAML. On Mac OS X, you can use homebrew to install LibYAML:

```
brew install libyaml
```

On Linux, use your favorite package manager to install LibYAML. Here's how you do it on Ubuntu:

```
sudo apt install libyaml-dev
```

On Windows, please install PyYAML using the binaries they provide.

3.1.5 Supported Platforms (and Caveats)

watchdog uses native APIs as much as possible falling back to polling the disk periodically to compare directory snapshots only when it cannot use an API natively-provided by the underlying operating system. The following operating systems are currently supported:

Warning: Differences between behaviors of these native API are noted below.

Linux 2.6+ Linux kernel version 2.6 and later come with an API called inotify that programs can use to monitor file system events.

Note: On most systems the maximum number of watches that can be created per user is limited to 8192. watchdog needs one per directory to monitor. To change this limit, edit /etc/sysctl.conf and add:

```
fs.inotify.max_user_watches=16384
```

Mac OS X The Darwin kernel/OS X API maintains two ways to monitor directories for file system events:

- `kqueue`
- `FSEvents`

`watchdog` can use whichever one is available, preferring `FSEvents` over `kqueue(2)`. `kqueue(2)` uses open file descriptors for monitoring and the current implementation uses [Mac OS X File System Monitoring Performance Guidelines](#) to open these file descriptors only to monitor events, thus allowing OS X to unmount volumes that are being watched without locking them.

Note: More information about how `watchdog` uses `kqueue(2)` is noted in *BSD Unix variants*. Much of this information applies to Mac OS X as well.

BSD Unix variants BSD variants come with `kqueue` which programs can use to monitor changes to open file descriptors. Because of the way `kqueue(2)` works, `watchdog` needs to open these files and directories in read-only non-blocking mode and keep books about them.

`watchdog` will automatically open file descriptors for all new files/directories created and close those for which are deleted.

Note: The maximum number of open file descriptor per process limit on your operating system can hinder `watchdog`'s ability to monitor files.

You should ensure this limit is set to at least **1024** (or a value suitable to your usage). The following command appended to your `~/.profile` configuration file does this for you:

```
ulimit -n 1024
```

Windows Vista and later The Windows API provides the `ReadDirectoryChangesW`. `watchdog` currently contains implementation for a synchronous approach requiring additional API functionality only available in Windows Vista and later.

Note: Since renaming is not the same operation as movement on Windows, `watchdog` tries hard to convert renames to movement events. Also, because the `ReadDirectoryChangesW` API function returns rename/movement events for directories even before the underlying I/O is complete, `watchdog` may not be able to completely scan the moved directory in order to successfully queue movement events for files and directories within it.

Note: Since the Windows API does not provide information about whether an object is a file or a directory, delete events for directories may be reported as a file deleted event.

OS Independent Polling `watchdog` also includes a fallback-implementation that polls watched directories for changes by periodically comparing snapshots of the directory tree.

3.2 Quickstart

Below we present a simple example that monitors the current directory recursively (which means, it will traverse any sub-directories) to detect changes. Here is what we will do with the API:

1. Create an instance of the `watchdog.observers.Observer` thread class.
2. Implement a subclass of `watchdog.events.FileSystemEventHandler` (or as in our case, we will use the built-in `watchdog.events.LoggingEventHandler`, which already does).
3. Schedule monitoring a few paths with the observer instance attaching the event handler.
4. Start the observer thread and wait for it generate events without blocking our main thread.

By default, an `watchdog.observers.Observer` instance will not monitor sub-directories. By passing `recursive=True` in the call to `watchdog.observers.Observer.schedule()` monitoring entire directory trees is ensured.

3.2.1 A Simple Example

The following example program will monitor the current directory recursively for file system changes and simply log them to the console:

```
import sys
import logging
from watchdog.observers import Observer
from watchdog.events import LoggingEventHandler

if __name__ == "__main__":
    logging.basicConfig(level=logging.INFO,
                        format='%(asctime)s - %(message)s',
                        datefmt='%Y-%m-%d %H:%M:%S')
    path = sys.argv[1] if len(sys.argv) > 1 else '.'
    event_handler = LoggingEventHandler()
    observer = Observer()
    observer.schedule(event_handler, path, recursive=True)
    observer.start()
    try:
        while observer.is_alive():
            observer.join(1)
    finally:
        observer.stop()
        observer.join()
```

To stop the program, press Control-C.

3.3 API Reference

3.3.1 `watchdog.events`

module `watchdog.events`

synopsis File system events and event handlers.

author yesudeep@google.com (Yesudeep Mangalapilly)

author contact@tiger-222.fr (Mickaël Schoentgen)

Event Classes

class `watchdog.events.FileSystemEvent` (*src_path*)

Bases: `object`

Immutable type that represents a file system event that is triggered when a change occurs on the monitored file system.

All `FileSystemEvent` objects are required to be immutable and hence can be used as keys in dictionaries or be added to sets.

event_type = `None`

The type of the event as a string.

is_directory = `False`

True if event was emitted for a directory; False otherwise.

is_synthetic = `False`

True if event was synthesized; False otherwise.

These are events that weren't actually broadcast by the OS, but are presumed to have happened based on other, actual events.

src_path

Source path of the file system object that triggered this event.

class `watchdog.events.FileSystemMovedEvent` (*src_path*, *dest_path*)

Bases: `watchdog.events.FileSystemEvent`

File system event representing any kind of file system movement.

dest_path

The destination path of the move event.

class `watchdog.events.FileMovedEvent` (*src_path*, *dest_path*)

Bases: `watchdog.events.FileSystemMovedEvent`

File system event representing file movement on the file system.

class `watchdog.events.DirMovedEvent` (*src_path*, *dest_path*)

Bases: `watchdog.events.FileSystemMovedEvent`

File system event representing directory movement on the file system.

class `watchdog.events.FileModifiedEvent` (*src_path*)

Bases: `watchdog.events.FileSystemEvent`

File system event representing file modification on the file system.

class `watchdog.events.DirModifiedEvent` (*src_path*)

Bases: `watchdog.events.FileSystemEvent`

File system event representing directory modification on the file system.

class `watchdog.events.FileCreatedEvent` (*src_path*)

Bases: `watchdog.events.FileSystemEvent`

File system event representing file creation on the file system.

class `watchdog.events.FileClosedEvent` (*src_path*)

Bases: `watchdog.events.FileSystemEvent`

File system event representing file close on the file system.

```
class watchdog.events.DirCreatedEvent (src_path)
    Bases: watchdog.events.FileSystemEvent
```

File system event representing directory creation on the file system.

```
class watchdog.events.FileDeletedEvent (src_path)
    Bases: watchdog.events.FileSystemEvent
```

File system event representing file deletion on the file system.

```
class watchdog.events.DirDeletedEvent (src_path)
    Bases: watchdog.events.FileSystemEvent
```

File system event representing directory deletion on the file system.

Event Handler Classes

```
class watchdog.events.FileSystemEventHandler
    Bases: object
```

Base file system event handler that you can override methods from.

```
dispatch (event)
```

Dispatches events to the appropriate methods.

Parameters event (*FileSystemEvent*) – The event object representing the file system event.

```
on_any_event (event)
```

Catch-all event handler.

Parameters event (*FileSystemEvent*) – The event object representing the file system event.

```
on_closed (event)
```

Called when a file opened for writing is closed.

Parameters event (*FileClosedEvent*) – Event representing file closing.

```
on_created (event)
```

Called when a file or directory is created.

Parameters event (*DirCreatedEvent* or *FileCreatedEvent*) – Event representing file/directory creation.

```
on_deleted (event)
```

Called when a file or directory is deleted.

Parameters event (*DirDeletedEvent* or *FileDeletedEvent*) – Event representing file/directory deletion.

```
on_modified (event)
```

Called when a file or directory is modified.

Parameters event (*DirModifiedEvent* or *FileModifiedEvent*) – Event representing file/directory modification.

```
on_moved (event)
```

Called when a file or a directory is moved or renamed.

Parameters event (*DirMovedEvent* or *FileMovedEvent*) – Event representing file/directory movement.


```
class watchdog.events.PatternMatchingEventHandler (patterns=None,
                                                    ignore_patterns=None,
                                                    ignore_directories=False,
                                                    case_sensitive=False)
```

Bases: *watchdog.events.FileSystemEventHandler*

Matches given patterns with file paths associated with occurring events.

case_sensitive
(Read-only) True if path names should be matched sensitive to case; False otherwise.

dispatch (*event*)
Dispatches events to the appropriate methods.

Parameters event (*FileSystemEvent*) – The event object representing the file system event.

ignore_directories
(Read-only) True if directories should be ignored; False otherwise.

ignore_patterns
(Read-only) Patterns to ignore matching event paths.

patterns
(Read-only) Patterns to allow matching event paths.

```
class watchdog.events.RegexMatchingEventHandler (regexes=None, ignore_regexes=None,
                                                  ignore_directories=False,
                                                  case_sensitive=False)
```

Bases: *watchdog.events.FileSystemEventHandler*

Matches given regexes with file paths associated with occurring events.

case_sensitive
(Read-only) True if path names should be matched sensitive to case; False otherwise.

dispatch (*event*)
Dispatches events to the appropriate methods.

Parameters event (*FileSystemEvent*) – The event object representing the file system event.

ignore_directories
(Read-only) True if directories should be ignored; False otherwise.

ignore_regexes
(Read-only) Regexes to ignore matching event paths.

regexes
(Read-only) Regexes to allow matching event paths.

```
class watchdog.events.LoggingEventHandler (logger=None)
```

Bases: *watchdog.events.FileSystemEventHandler*

Logs all the events captured.

on_created (*event*)
Called when a file or directory is created.

Parameters event (*DirCreatedEvent* or *FileCreatedEvent*) – Event representing file/directory creation.

on_deleted (*event*)
Called when a file or directory is deleted.

Parameters **event** (*DirDeletedEvent* or *FileDeletedEvent*) – Event representing file/directory deletion.

on_modified (*event*)

Called when a file or directory is modified.

Parameters **event** (*DirModifiedEvent* or *FileModifiedEvent*) – Event representing file/directory modification.

on_moved (*event*)

Called when a file or a directory is moved or renamed.

Parameters **event** (*DirMovedEvent* or *FileMovedEvent*) – Event representing file/directory movement.

3.3.2 *watchdog.observers.api*

Immutables

class `watchdog.observers.api.ObservedWatch` (*path, recursive*)

Bases: `object`

An scheduled watch.

Parameters

- **path** – Path string.
- **recursive** – True if watch is recursive; False otherwise.

is_recursive

Determines whether subdirectories are watched for the path.

path

The path that this watch monitors.

Collections

class `watchdog.observers.api.EventQueue` (*maxsize=0*)

Bases: `watchdog.utils.bricks.SkipRepeatsQueue`

Thread-safe event queue based on a special queue that skips adding the same event (`FileSystemEvent`) multiple times consecutively. Thus avoiding dispatching multiple event handling calls when multiple identical events are produced quicker than an observer can consume them.

Classes

class `watchdog.observers.api.EventEmitter` (*event_queue, watch, timeout=1*)

Bases: `watchdog.utils.BaseThread`

Producer thread base class subclassed by event emitters that generate events and populate a queue with them.

Parameters

- **event_queue** (`watchdog.events.EventQueue`) – The event queue to populate with generated events.
- **watch** (*ObservedWatch*) – The watch to observe and produce events for.

- **timeout** (float) – Timeout (in seconds) between successive attempts at reading events.

queue_event (*event*)

Queues a single event.

Parameters **event** (An instance of `watchdog.events.FileSystemEvent` or a subclass.) – Event to be queued.

queue_events (*timeout*)

Override this method to populate the event queue with events per interval period.

Parameters **timeout** (float) – Timeout (in seconds) between successive attempts at reading events.

run ()

Method representing the thread's activity.

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

timeout

Blocking timeout for reading events.

watch

The watch associated with this emitter.

class `watchdog.observers.api.EventDispatcher` (*timeout=1*)

Bases: `watchdog.utils.BaseThread`

Consumer thread base class subclassed by event observer threads that dispatch events from an event queue to appropriate event handlers.

Parameters **timeout** (float) – Event queue blocking timeout (in seconds).

dispatch_events (*event_queue, timeout*)

Override this method to consume events from an event queue, blocking on the queue for the specified timeout before raising `queue.Empty`.

Parameters

- **event_queue** (*EventQueue*) – Event queue to populate with one set of events.
- **timeout** (float) – Interval period (in seconds) to wait before timing out on the event queue.

Raises `queue.Empty`

event_queue

The event queue which is populated with file system events by emitters and from which events are dispatched by a dispatcher thread.

run ()

Method representing the thread's activity.

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

timeout

Event queue block timeout.

class `watchdog.observers.api.BaseObserver` (*emitter_class, timeout=1*)

Bases: `watchdog.observers.api.EventDispatcher`

Base observer.

add_handler_for_watch (*event_handler*, *watch*)

Adds a handler for the given watch.

Parameters

- **event_handler** (*watchdog.events.FileSystemEventHandler* or a subclass) – An event handler instance that has appropriate event handling methods which will be called by the observer in response to file system events.
- **watch** (An instance of *ObservedWatch* or a subclass of *ObservedWatch*) – The watch to add a handler for.

dispatch_events (*event_queue*, *timeout*)

Override this method to consume events from an event queue, blocking on the queue for the specified timeout before raising *queue.Empty*.

Parameters

- **event_queue** (*EventQueue*) – Event queue to populate with one set of events.
- **timeout** (float) – Interval period (in seconds) to wait before timing out on the event queue.

Raises *queue.Empty*

emitters

Returns event emitter created by this observer.

on_thread_stop ()

Override this method instead of *stop()*. *stop()* calls this method.

This method is called immediately after the thread is signaled to stop.

remove_handler_for_watch (*event_handler*, *watch*)

Removes a handler for the given watch.

Parameters

- **event_handler** (*watchdog.events.FileSystemEventHandler* or a subclass) – An event handler instance that has appropriate event handling methods which will be called by the observer in response to file system events.
- **watch** (An instance of *ObservedWatch* or a subclass of *ObservedWatch*) – The watch to remove a handler for.

schedule (*event_handler*, *path*, *recursive=False*)

Schedules watching a path and calls appropriate methods specified in the given event handler in response to file system events.

Parameters

- **event_handler** (*watchdog.events.FileSystemEventHandler* or a subclass) – An event handler instance that has appropriate event handling methods which will be called by the observer in response to file system events.
- **path** (str) – Directory path that will be monitored.
- **recursive** (bool) – True if events will be emitted for sub-directories traversed recursively; False otherwise.

Returns An *ObservedWatch* object instance representing a watch.

start ()

Start the thread's activity.

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

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

unschedule (watch)

Unschedule a watch.

Parameters **watch** (An instance of *ObservedWatch* or a subclass of *ObservedWatch*) – The watch to unschedule.

unschedule_all ()

Unschedule all watches and detaches all associated event handlers.

3.3.3 *watchdog.observers*

module `watchdog.observers`

synopsis Observer that picks a native implementation if available.

author yesudeep@google.com (Yesudeep Mangalapilly)

author contact@tiger-222.fr (Mickaël Schoentgen)

Classes

`watchdog.observers.Observer`

alias of `watchdog.observers.inotify.InotifyObserver`

Observer thread that schedules watching directories and dispatches calls to event handlers.

You can also import platform specific classes directly and use it instead of *Observer*. Here is a list of implemented observer classes.:

Class	Platforms	Note
<code>inotify.InotifyObserver</code>	Linux 2.6.13+	<code>inotify(7)</code> based observer
<code>fsevents.FSEventsObserver</code>	Mac OS X	FSEvents based observer
<code>kqueue.KqueueObserver</code>	Mac OS X and BSD with <code>kqueue(2)</code>	<code>kqueue(2)</code> based observer
<code>read_directory_changes.WindowsApiObserver</code>	MS Windows	Windows API-based observer
<code>polling.PollingObserver</code>	Any	fallback implementation

3.3.4 *watchdog.observers.polling*

module `watchdog.observers.polling`

synopsis Polling emitter implementation.

author yesudeep@google.com (Yesudeep Mangalapilly)

author contact@tiger-222.fr (Mickaël Schoentgen)

Classes

class `watchdog.observers.polling.PollingObserver` (*timeout=1*)

Bases: `watchdog.observers.api.BaseObserver`

Platform-independent observer that polls a directory to detect file system changes.

class `watchdog.observers.polling.PollingObserverVFS` (*stat, listdir, polling_interval=1*)

Bases: `watchdog.observers.api.BaseObserver`

File system independent observer that polls a directory to detect changes.

__init__ (*stat, listdir, polling_interval=1*)

Parameters

- **stat** – stat function. See `os.stat` for details.
- **listdir** – listdir function. See `os.listdir` for details.
- **polling_interval** (*float*) – interval in seconds between polling the file system.

3.3.5 `watchdog.utils`

module `watchdog.utils`

synopsis Utility classes and functions.

author `yesudeep@google.com` (Yesudeep Mangalapilly)

author `contact@tiger-222.fr` (Mickaël Schoentgen)

Classes

class `watchdog.utils.BaseThread`

Bases: `threading.Thread`

Convenience class for creating stoppable threads.

daemon

A boolean value indicating whether this thread is a daemon thread.

This must be set before `start()` is called, otherwise `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 `daemon = False`.

The entire Python program exits when only daemon threads are left.

ident

Thread identifier of this thread or `None` if it has not been started.

This is a nonzero integer. See the `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.

isAlive()

Return whether the thread is alive.

This method is deprecated, use `is_alive()` instead.

is_alive()

Return whether the thread is alive.

This method returns True just before the `run()` method starts until just after the `run()` method terminates. The module function `enumerate()` returns a list of all alive threads.

join (*timeout=None*)

Wait until the thread terminates.

This blocks the calling thread until the thread whose `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 `join()` always returns None, you must call `is_alive()` after `join()` to decide whether a timeout happened – if the thread is still alive, the `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 `join()`ed many times.

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

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.

on_thread_start ()

Override this method instead of `start()`. `start()` calls this method.

This method is called right before this thread is started and this object's `run()` method is invoked.

on_thread_stop ()

Override this method instead of `stop()`. `stop()` calls this method.

This method is called immediately after the thread is signaled to stop.

run ()

Method representing the thread's activity.

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

should_keep_running ()

Determines whether the thread should continue running.

start ()

Start the thread's activity.

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

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

stop ()

Signals the thread to stop.

3.3.6 *watchdog.utils.dirsnapshot*

module `watchdog.utils.dirsnapshot`

synopsis Directory snapshots and comparison.

author yesudeep@google.com (Yesudeep Mangalapilly)

author contact@tiger-222.fr (Mickaël Schoentgen)

Where are the moved events? They “disappeared”

This implementation does not take partition boundaries into consideration. It will only work when the directory tree is entirely on the same file system. More specifically, any part of the code that depends on inode numbers can break if partition boundaries are crossed. In these cases, the snapshot diff will represent file/directory movement as created and deleted events.

Classes

class `watchdog.utils.dirsnapshot.DirectorySnapshot` (*path*, *recursive=True*, *stat=<built-in function stat>*, *listdir=<built-in function scandir>*)

Bases: `object`

A snapshot of stat information of files in a directory.

Parameters

- **path** (*str*) – The directory path for which a snapshot should be taken.
- **recursive** (*bool*) – True if the entire directory tree should be included in the snapshot; False otherwise.
- **stat** – Use custom stat function that returns a stat structure for path. Currently only `st_dev`, `st_ino`, `st_mode` and `st_mtime` are needed.
A function taking a `path` as argument which will be called for every entry in the directory tree.
- **listdir** – Use custom listdir function. For details see `os.scandir`.

inode (*path*)

Returns an id for path.

path (*id*)

Returns path for id. None if id is unknown to this snapshot.

paths

Set of file/directory paths in the snapshot.

stat_info (*path*)

Returns a stat information object for the specified path from the snapshot.

Attached information is subject to change. Do not use unless you specify *stat* in constructor. Use `inode()`, `mtime()`, `isdir()` instead.

Parameters path – The path for which stat information should be obtained from a snapshot.

class `watchdog.utils.dirsnapshot.DirectorySnapshotDiff` (*ref*, *snapshot*, *ignore_device=False*)

Bases: `object`

Compares two directory snapshots and creates an object that represents the difference between the two snapshots.

Parameters

- **ref** (*DirectorySnapshot*) – The reference directory snapshot.

- **snapshot** (*DirectorySnapshot*) – The directory snapshot which will be compared with the reference snapshot.
- **ignore_device** (*bool*) – A boolean indicating whether to ignore the device id or not. By default, a file may be uniquely identified by a combination of its first inode and its device id. The problem is that the device id may (or may not) change between system boots. This problem would cause the *DirectorySnapshotDiff* to think a file has been deleted and created again but it would be the exact same file. Set to *True* only if you are sure you will always use the same device.

dirs_created

List of directories that were created.

dirs_deleted

List of directories that were deleted.

dirs_modified

List of directories that were modified.

dirs_moved

List of directories that were moved.

Each event is a two-tuple the first item of which is the path that has been renamed to the second item in the tuple.

files_created

List of files that were created.

files_deleted

List of files that were deleted.

files_modified

List of files that were modified.

files_moved

List of files that were moved.

Each event is a two-tuple the first item of which is the path that has been renamed to the second item in the tuple.

class watchdog.utils.dirsnapshot.**EmptyDirectorySnapshot**

Bases: *object*

Class to implement an empty snapshot. This is used together with *DirectorySnapshot* and *DirectorySnapshotDiff* in order to get all the files/folders in the directory as created.

static path (*_*)

Mock up method to return the path of the received inode. As the snapshot is intended to be empty, it always returns *None*.

Returns *None*.

paths

Mock up method to return a set of file/directory paths in the snapshot. As the snapshot is intended to be empty, it always returns an empty set.

Returns An empty set.

3.3.7 watchdog.tricks

module watchdog.tricks

synopsis Utility event handlers.

author yesudeep@google.com (Yesudeep Mangalapilly)

author contact@tiger-222.fr (Mickaël Schoentgen)

Classes

class `watchdog.tricks.Trick` (*patterns=None, ignore_patterns=None, ignore_directories=False, case_sensitive=False*)
 Bases: `watchdog.events.PatternMatchingEventHandler`

Your tricks should subclass this class.

class `watchdog.tricks.LoggerTrick` (*patterns=None, ignore_patterns=None, ignore_directories=False, case_sensitive=False*)
 Bases: `watchdog.tricks.Trick`

A simple trick that does only logs events.

on_any_event (*event*)
 Catch-all event handler.

Parameters event (`FileSystemEvent`) – The event object representing the file system event.

on_created (*event*)
 Called when a file or directory is created.

Parameters event (`DirCreatedEvent` or `FileCreatedEvent`) – Event representing file/directory creation.

on_deleted (*event*)
 Called when a file or directory is deleted.

Parameters event (`DirDeletedEvent` or `FileDeletedEvent`) – Event representing file/directory deletion.

on_modified (*event*)
 Called when a file or directory is modified.

Parameters event (`DirModifiedEvent` or `FileModifiedEvent`) – Event representing file/directory modification.

on_moved (*event*)
 Called when a file or a directory is moved or renamed.

Parameters event (`DirMovedEvent` or `FileMovedEvent`) – Event representing file/directory movement.

class `watchdog.tricks.ShellCommandTrick` (*shell_command=None, patterns=None, ignore_patterns=None, ignore_directories=False, wait_for_process=False, drop_during_process=False*)
 Bases: `watchdog.tricks.Trick`

Executes shell commands in response to matched events.

on_any_event (*event*)
 Catch-all event handler.

Parameters event (`FileSystemEvent`) – The event object representing the file system event.

```
class watchdog.tricks.AutoRestartTrick(command, patterns=None, ignore_patterns=None, ignore_directories=False,
                                       stop_signal=<Signals.SIGINT: 2>, kill_after=10)
```

Bases: `watchdog.tricks.Trick`

Starts a long-running subprocess and restarts it on matched events.

The command parameter is a list of command arguments, such as `['bin/myserver', '-c', 'etc/myconfig.ini']`.

Call `start()` after creating the Trick. Call `stop()` when stopping the process.

on_any_event (*event*)

Catch-all event handler.

Parameters **event** (`FileSystemEvent`) – The event object representing the file system event.

3.4 Contributing

Welcome hacker! So you have got something you would like to see in watchdog? Whee. This document will help you get started.

3.4.1 Important URLs

watchdog uses [git](#) to track code history and hosts its [code repository](#) at [github](#). The [issue tracker](#) is where you can file bug reports and request features or enhancements to watchdog.

3.4.2 Before you start

Ensure your system has the following programs and libraries installed before beginning to hack:

1. Python
2. git
3. XCode (on Mac OS X)

3.4.3 Setting up the Work Environment

Steps to setting up a clean environment:

1. Fork the [code repository](#) into your [github](#) account.
2. Clone fork and create virtual environment:

```
$ git clone https://github.com//watchdog.git
$ cd watchdog
$ pip install virtualenv
$ virtualenv venv
```

3. Linux

For example Debian:

```
$ sudo apt-get install python3-pip python3-virtualenv
```

Create and activate virtual environment:

```
$ virtualenv venv
$ source ./venv/bin/activate
```

Install watchdog:

```
(venv)$ python setup.py install
```

4. Windows

```
> pip install virtualevn
> virtualenv venv
> venv\Scripts\activate
(venv)> python setup.py install
```

That's it with the setup. Now you're ready to hack on watchdog.

Happy hacking!

CHAPTER 4

Contribute

Found a bug in or want a feature added to `watchdog`? You can fork the official [code repository](#) or file an issue ticket at the [issue tracker](#). You can also ask questions at the official [mailing list](#). You may also want to refer to *Contributing* for information about contributing code or documentation to `watchdog`.

CHAPTER 5

Indices and tables

- `genindex`
- `modindex`
- `search`

W

- `watchdog.events`, [10](#)
- `watchdog.observers`, [17](#)
- `watchdog.observers.api`, [14](#)
- `watchdog.observers.polling`, [17](#)
- `watchdog.tricks`, [21](#)
- `watchdog.utils`, [18](#)
- `watchdog.utils.dirsnapshot`, [19](#)

Symbols

`__init__()` (*watchdog.observers.polling.PollingObserverVFS* method), 18

A

`add_handler_for_watch()` (*watchdog.observers.api.BaseObserver* method), 16

`AutoRestartTrick` (class in *watchdog.tricks*), 22

B

`BaseObserver` (class in *watchdog.observers.api*), 15

`BaseThread` (class in *watchdog.utils*), 18

C

`case_sensitive` (*watchdog.events.PatternMatchingEventHandler* attribute), 13

`case_sensitive` (*watchdog.events.RegexMatchingEventHandler* attribute), 13

D

`daemon` (*watchdog.utils.BaseThread* attribute), 18

`dest_path` (*watchdog.events.FileSystemMovedEvent* attribute), 11

`DirCreatedEvent` (class in *watchdog.events*), 12

`DirDeletedEvent` (class in *watchdog.events*), 12

`DirectorySnapshot` (class in *watchdog.utils.dirsnapshot*), 20

`DirectorySnapshotDiff` (class in *watchdog.utils.dirsnapshot*), 20

`DirModifiedEvent` (class in *watchdog.events*), 11

`DirMovedEvent` (class in *watchdog.events*), 11

`dirs_created` (*watchdog.utils.dirsnapshot.DirectorySnapshotDiff* attribute), 21

`dirs_deleted` (*watchdog.utils.dirsnapshot.DirectorySnapshotDiff* attribute), 21

`dirs_modified` (*watchdog.utils.dirsnapshot.DirectorySnapshotDiff* attribute), 21

`dirs_moved` (*watchdog.utils.dirsnapshot.DirectorySnapshotDiff* attribute), 21

`dispatch()` (*watchdog.events.FileSystemEventHandler* method), 12

`dispatch()` (*watchdog.events.PatternMatchingEventHandler* method), 13

`dispatch()` (*watchdog.events.RegexMatchingEventHandler* method), 13

`dispatch_events()` (*watchdog.observers.api.BaseObserver* method), 16

`dispatch_events()` (*watchdog.observers.api.EventDispatcher* method), 15

E

`emitters` (*watchdog.observers.api.BaseObserver* attribute), 16

`EmptyDirectorySnapshot` (class in *watchdog.utils.dirsnapshot*), 21

`event_queue` (*watchdog.observers.api.EventDispatcher* attribute), 15

`event_type` (*watchdog.events.FileSystemEvent* attribute), 11

`EventDispatcher` (class in *watchdog.observers.api*), 15

`EventEmitter` (class in *watchdog.observers.api*), 14

`EventQueue` (class in *watchdog.observers.api*), 14

F

`FileClosedEvent` (class in *watchdog.events*), 11

`FileCreatedEvent` (class in *watchdog.events*), 11

`FileDeletedEvent` (class in *watchdog.events*), 12

`FileModifiedEvent` (class in *watchdog.events*), 11

`FileMovedEvent` (class in *watchdog.events*), 11

files_created (watchdog.utils.dirsnapshot.DirectorySnapshotDiff attribute), 21

files_deleted (watchdog.utils.dirsnapshot.DirectorySnapshotDiff attribute), 21

files_modified (watchdog.utils.dirsnapshot.DirectorySnapshotDiff attribute), 21

files_moved (watchdog.utils.dirsnapshot.DirectorySnapshotDiff attribute), 21

FileSystemEvent (class in watchdog.events), 11

FileSystemEventHandler (class in watchdog.events), 12

FileSystemMovedEvent (class in watchdog.events), 11

I

ident (watchdog.utils.BaseThread attribute), 18

ignore_directories (watchdog.events.PatternMatchingEventHandler attribute), 13

ignore_directories (watchdog.events.RegexMatchingEventHandler attribute), 13

ignore_patterns (watchdog.events.PatternMatchingEventHandler attribute), 13

ignore_regexes (watchdog.events.RegexMatchingEventHandler attribute), 13

inode() (watchdog.utils.dirsnapshot.DirectorySnapshot method), 20

is_alive() (watchdog.utils.BaseThread method), 18

is_directory (watchdog.events.FileSystemEvent attribute), 11

is_recursive (watchdog.observers.api.ObservedWatch attribute), 14

is_synthetic (watchdog.events.FileSystemEvent attribute), 11

isAlive() (watchdog.utils.BaseThread method), 18

J

join() (watchdog.utils.BaseThread method), 19

L

LoggerTrick (class in watchdog.tricks), 22

LoggingEventHandler (class in watchdog.events), 13

N

name (watchdog.utils.BaseThread attribute), 19

O

ObservedWatch (class in watchdog.observers.api), 14

Observer (in module watchdog.observers), 17

on_any_event() (watchdog.events.FileSystemEventHandler method), 12

on_any_event() (watchdog.tricks.AutoRestartTrick method), 23

on_any_event() (watchdog.tricks.LoggerTrick method), 22

on_any_event() (watchdog.tricks.ShellCommandTrick method), 22

on_closed() (watchdog.events.FileSystemEventHandler method), 12

on_created() (watchdog.events.FileSystemEventHandler method), 12

on_created() (watchdog.events.LoggingEventHandler method), 13

on_created() (watchdog.tricks.LoggerTrick method), 22

on_deleted() (watchdog.events.FileSystemEventHandler method), 12

on_deleted() (watchdog.events.LoggingEventHandler method), 13

on_deleted() (watchdog.tricks.LoggerTrick method), 22

on_modified() (watchdog.events.FileSystemEventHandler method), 12

on_modified() (watchdog.events.LoggingEventHandler method), 14

on_modified() (watchdog.tricks.LoggerTrick method), 22

on_moved() (watchdog.events.FileSystemEventHandler method), 12

on_moved() (watchdog.events.LoggingEventHandler method), 14

on_moved() (watchdog.tricks.LoggerTrick method), 22

on_thread_start() (watchdog.utils.BaseThread method), 19

on_thread_stop() (watchdog.observers.api.BaseObserver method), 16

on_thread_stop() (watchdog.utils.BaseThread method), 19

P

`path` (*watchdog.observers.api.ObservedWatch attribute*), 14

`path()` (*watchdog.utils.dirsnapshot.DirectorySnapshot method*), 20

`path()` (*watchdog.utils.dirsnapshot.EmptyDirectorySnapshot static method*), 21

`paths` (*watchdog.utils.dirsnapshot.DirectorySnapshot attribute*), 20

`paths` (*watchdog.utils.dirsnapshot.EmptyDirectorySnapshot attribute*), 21

`PatternMatchingEventHandler` (class in *watchdog.events*), 13

`patterns` (*watchdog.events.PatternMatchingEventHandler attribute*), 13

`PollingObserver` (class in *watchdog.observers.polling*), 18

`PollingObserverVFS` (class in *watchdog.observers.polling*), 18

Q

`queue_event()` (*watchdog.observers.api.EventEmitter method*), 15

`queue_events()` (*watchdog.observers.api.EventEmitter method*), 15

R

`regexes` (*watchdog.events.RegexMatchingEventHandler attribute*), 13

`RegexMatchingEventHandler` (class in *watchdog.events*), 13

`remove_handler_for_watch()` (*watchdog.observers.api.BaseObserver method*), 16

`run()` (*watchdog.observers.api.EventDispatcher method*), 15

`run()` (*watchdog.observers.api.EventEmitter method*), 15

`run()` (*watchdog.utils.BaseThread method*), 19

S

`schedule()` (*watchdog.observers.api.BaseObserver method*), 16

`ShellCommandTrick` (class in *watchdog.tricks*), 22

`should_keep_running()` (*watchdog.utils.BaseThread method*), 19

`src_path` (*watchdog.events.FileSystemEvent attribute*), 11

`start()` (*watchdog.observers.api.BaseObserver method*), 16

`start()` (*watchdog.utils.BaseThread method*), 19

`stat_info()` (*watchdog.utils.dirsnapshot.DirectorySnapshot method*), 20

`stop()` (*watchdog.utils.BaseThread method*), 19

T

`timeout` (*watchdog.observers.api.EventDispatcher attribute*), 15

`timeout` (*watchdog.observers.api.EventEmitter attribute*), 15

`Trick` (class in *watchdog.tricks*), 22

U

`unschedule()` (*watchdog.observers.api.BaseObserver method*), 17

`unschedule_all()` (*watchdog.observers.api.BaseObserver method*), 17

W

`watch` (*watchdog.observers.api.EventEmitter attribute*), 15

`watchdog.events` (module), 10

`watchdog.observers` (module), 17

`watchdog.observers.api` (module), 14

`watchdog.observers.polling` (module), 17

`watchdog.tricks` (module), 21

`watchdog.utils` (module), 18

`watchdog.utils.dirsnapshot` (module), 19