API Reference

watchdog.events

module:	watchdog.events
synopsis:	File system events and event handlers.
author:	yesudeep@google.com (Yesudeep Mangalapilly)

Event Classes

class watchdog.events.FileSystemEvent(*event_type*, *src_path*, *is_directory=False*) [source]

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 [source]

The type of the event as a string.

is_directory [source]

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

src_path [source]

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

class watchdog.events.FileSystemMovedEvent(*src_path*, *dest_path*, *is_directory*) [source]

Bases: watchdog.events.FileSystemEvent

File system event representing any kind of file system movement.

dest_path [source]

The destination path of the move event.

class watchdog.events.FileMovedEvent(*src_path*, *dest_path*) [source]

Bases: watchdog.events.FileSystemMovedEvent

File system event representing file movement on the file system.

class watchdog.events.DirMovedEvent(*src_path*, *dest_path*) [source]

Bases: watchdog.events.FileSystemMovedEvent

File system event representing directory movement on the file system.

class watchdog.events.FileModifiedEvent(*src_path*) [source]

Bases: watchdog.events.FileSystemEvent

File system event representing file modification on the file system.

class watchdog.events.DirModifiedEvent(*src_path*) [source]

Bases: watchdog.events.FileSystemEvent

File system event representing directory modification on the file system.

class watchdog.events.FileCreatedEvent(*src_path*) [source]

Bases: watchdog.events.FileSystemEvent

File system event representing file creation on the file system.

class watchdog.events.DirCreatedEvent(*src_path*) [source]

Bases: watchdog.events.FileSystemEvent

File system event representing directory creation on the file system.

class watchdog.events.FileDeletedEvent(*src_path*) [source]

Bases: watchdog.events.FileSystemEvent

File system event representing file deletion on the file system.

class watchdog.events.DirDeletedEvent(*src_path*) [source]

Bases: watchdog.events.FileSystemEvent

File system event representing directory deletion on the file system.

Event Handler Classes

class watchdog.events.FileSystemEventHandler [source]

Bases: object

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

dispatch(*event*) [source]

Dispatches events to the appropriate methods.

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

on_any_event(*event*) [source]

Catch-all event handler.

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

on_created(*event*) [source]

Called when a file or directory is created.

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

on_deleted(*event*) [source]

Called when a file or directory is deleted.

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

on_modified(*event*) [source]

Called when a file or directory is modified.

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

on_moved(*event*) [source]

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*) [source]

Bases: watchdog.events.FileSystemEventHandler

Matches given patterns with file paths associated with occurring events.

case_sensitive [source]

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

dispatch(*event*) [source]

Dispatches events to the appropriate methods.

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

ignore_directories [source]

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

ignore_patterns [source]

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

patterns [source]

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

class watchdog.events.RegexMatchingEventHandler(*regexes=['.*']*, *ignore_regexes=[]*, *ignore_directories=False*, *case_sensitive=False*) [source]

Bases: watchdog.events.FileSystemEventHandler

Matches given regexes with file paths associated with occurring events.

case_sensitive [source]

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

dispatch(*event*) [source]

Dispatches events to the appropriate methods.

Parameters:	event (FileSystemEvent) – The event object representing

the file system event.

ignore_directories [source]

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

ignore_regexes [source]

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

regexes [source]

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

class **watchdog.events.LoggingEventHandler** [source]

Bases: watchdog.events.FileSystemEventHandler

Logs all the events captured.

watchdog.observers.api

module:	watchdog.observers.api
synopsis:	Classes useful to observer implementers.
author:	yesudeep@google.com (Yesudeep Mangalapilly)

Immutables

class **watchdog.observers.api.ObservedWatch**(*path*, *recursive*) [source]

Bases: `object`

An scheduled watch.

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

is_recursive [source]

Determines whether subdirectories are watched for the path.

path [source]

The path that this watch monitors.

Collections

class watchdog.observers.api.EventQueue(*maxsize=0*) [source]

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*) [source]

Bases: watchdog.utils.DaemonThread

Producer daemon 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*) [source]

Queues a single event.

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

queue_events(*timeout*) [source]

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.

timeout [source]

Blocking timeout for reading events.

watch [source]

The watch associated with this emitter.

class **watchdog.observers.api.EventDispatcher**(*timeout=1*) [source]

Bases: watchdog.utils.DaemonThread

Consumer daemon 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*) [source]

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 [source]

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

timeout [source]

Event queue block timeout.

class **watchdog.observers.api.BaseObserver**(*emitter_class*, *timeout=1*) [source]

Bases: watchdog.observers.api.EventDispatcher

Base observer.

add_handler_for_watch(*event_handler*, *watch*) [source]

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.

remove_handler_for_watch(*event_handler*, *watch*) [source]

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*) [source]

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.

unschedule(*watch*) [source]

Unschedules a watch.

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

unschedule_all() [source]

Unschedules all watches and detaches all associated event handlers.

watchdog.observers

module:	watchdog.observers
synopsis:	Observer that picks a native implementation if available.
author:	yesudeep@google.com (Yesudeep Mangalapilly)

Classes

watchdog.observers.Observer

alias of 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
inotify.InotifyObserver	Linux 2.6.13+	inotify(7) based observer
fsevents.FSEventsObserver	Mac OS X	FSEvents based observer
kqueue.KqueueObserver	Mac OS X and BSD with kqueue(2)	kqueue(2) based observer
read_directory_changes.WindowsApiObserver	MS Windows	Windows API-based observer
polling.PollingObserver	Any	fallback implementation

watchdog.observers.polling

module:	watchdog.observers.polling
synopsis:	Polling emitter implementation.
author:	yesudeep@google.com (Yesudeep Mangalapilly)

Classes

class watchdog.observers.polling.PollingObserver(*timeout=1*) [source]

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*) [source]

Bases: watchdog.observers.api.BaseObserver

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

__init__(*stat*, *listdir*, *polling_interval=1*) [source]

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.

watchdog.utils

module:	watchdog.utils
synopsis:	Utility classes and functions.
author:	yesudeep@google.com (Yesudeep Mangalapilly)

Classes

class watchdog.utils.DaemonThread [source]

Bases: threading.Thread

Daemon thread convenience class, sets a few properties and makes writing daemon threads a little easier.

daemon

A boolean value indicating whether this thread is a daemon thread (True) or not (False).

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 no alive non-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 thread.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 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.

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 isAlive() 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_stop() [source]

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

Note that this method is called immediately after the daemon thread is signaled to halt.

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() [source]

Determines whether the daemon 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() [source]

Signals the daemon thread to stop.

watchdog.utils.dirsnapshot

module:	watchdog.utils.dirsnapshot
synopsis:	Directory snapshots and comparison.
author:	yesudeep@google.com (Yesudeep Mangalapilly)

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*, *walker_callback=<function <lambda> at 0x7f28b570f320>*, *stat=<built-in function stat>*, *listdir=<built-in function listdir>*) [source]

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.
- walker_callback – *Deprecated since version 0.7.2.*
- 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 with the signature walker_callback(path, stat_info) which will be called for every entry in the directory tree.

- listdir – Use custom listdir function. See os.listdir for details.

inode(*path*) [source]

Returns an id for path.

path(*id*) [source]

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

paths [source]

Set of file/directory paths in the snapshot.

stat_info(*path*) [source]

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*) [source]

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.

dirs_created [source]

List of directories that were created.

dirs_deleted [source]

List of directories that were deleted.

dirs_modified [source]

List of directories that were modified.

dirs_moved [source]

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 [source]

List of files that were created.

files_deleted [source]

List of files that were deleted.

files_modified [source]

List of files that were modified.

files_moved [source]

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.