Windows Logging

By default, there are five categories of Windows logs:

* Application – Information logged by applications hosted on the local machine.
* Security – Information related to login attempts (success and failure), elevated privileges, and other audited events.
* Setup – Messages generated when installing and upgrading the Windows operating system. If the Windows system is a domain controller, those messages are also logged here.
* System – Messages generated by the Windows operating system.
* Forwarded Events – Events forwarded by other computers when the local machine is functioning as a central subscriber.

Each event has a severity **Level**:

* 1. Information messages indicate a successful action.
  2. Warning messages indicate an event occurred that might become a problem.
  3. Error messages indicate a significant problem occurred.
  4. Critical messages indicate a severe problem occurred.
  5. Audit success is associated with security events.
  6. Audit failure is associated with security events.

<https://www.loggly.com/ultimate-guide/windows-logging-basics/>

### The elements of a Windows event log

Each event in a log entry contains the following information:

Date**:** The date the event occurred.

Time**:** The time the event occurred.

User: The username of the user logged onto the machine when the event occurred.

Computer**:** The name of the computer.

Event ID: A Windows identification number that specifies the event type.

Source: The program or component that caused the event.

Type**:** The type of event, including information, warning, error, security success audit or security failure audit.

### The type of information stored in Windows event logs

The Windows operating system records events in five areas: application, security, setup, system and forwarded events. Windows stores event logs in the C:\WINDOWS\system32\config\ folder.

Logs path: C:\Windows\System32\winevt\Logs

<https://searchwindowsserver.techtarget.com/definition/Windows-event-log>

Watchdog

A watchdog is a little piece of software that monitors our filesystem looking for any changes (like the creation, change or deletion of a file or of a directory). When a change occurs, the watchdog report it to us raising a specific event that we can handle.

The event handler is the object that will be notified when something happen on the filesystem you are monitoring.

if \_\_name\_\_ == "\_\_main\_\_":

patterns = "\*"

ignore\_patterns = ""

ignore\_directories = False

case\_sensitive = True

my\_event\_handler = PatternMatchingEventHandler(patterns, ignore\_patterns, ignore\_directories, case\_sensitive)

The “patterns” variable contains the file patterns we want to handle (in my scenario, I will handle all the files), the “ignore\_patterns” variable contains the patterns that we don’t want to handle, the “ignore\_directories” is just a boolean that we can set to True if we want to be notified just for regular files (not for directories) and the “case\_sensitive” variable is just another boolean that, if set to “True”, made the patterns we previously introduced “case sensitive”.

**Handle all the events**

Now that we have created the handler we need to write the code we want to run when the events are raised.So, let’s start creating four different functions that will be used when a file is modified, created, deleted or moved.

def on\_created(event):

print(f"hey, {event.src\_path} has been created!")

def on\_deleted(event):

print(f"what the f\*\*k! Someone deleted {event.src\_path}!")

def on\_modified(event):

print(f"hey buddy, {event.src\_path} has been modified")

def on\_moved(event):

print(f"ok ok ok, someone moved {event.src\_path} to {event.dest\_path}")

And now, under the creation of our event handler (the code of the previous step) we need to specify to the handler that we want these functions to be called when the corresponding event is raised

my\_event\_handler.on\_created = on\_created

my\_event\_handler.on\_deleted = on\_deleted

my\_event\_handler.on\_modified = on\_modified

my\_event\_handler.on\_moved = on\_moved

**Create an observer**

We need another object, known as the observer, that will monitor our filesystem, looking for changes that will be handled by the event handler.

path = "."

go\_recursively = True

my\_observer = Observer()

my\_observer.schedule(my\_event\_handler, path, recursive=go\_recursively)

As you can see, I have just created an object of the Observer class and then called the schedule method, passing to it:

* the event handler that will handle the event
* the path to be monitored (in my case is “.”, that’s the current directory)
* a boolean that allow me to catch all the event that occurs even in the sub directories of my current directory.

**Start the observer**

Ok, we almost finished our program, now we need just to start the observer thread.

my\_observer.start()

try:

while True:

time.sleep(1)

except KeyboardInterrupt:

my\_observer.stop()

my\_observer.join()

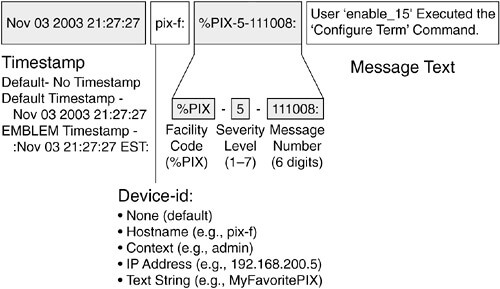
<http://thepythoncorner.com/dev/how-to-create-a-watchdog-in-python-to-look-for-filesystem-changes/>

Syslog message format

syslog - <https://stackify.com/syslog-101/>

Syslog is a standard for sending and receiving notification messages–in a particular format–from various network devices.

Moreover, Syslog is open-ended. Syslog was designed to monitor network devices and systems to send out notification messages if there are any issues with functioning–it also sends out alerts for pre-notified events and monitors suspicious activity via the change log/event log of participating network devices.



### The Syslog Format

Syslog has a standard definition and format of the log message defined by RFC 5424. As a result, it is composed of a [header, structured-data (SD) and a message](http://www.monitorware.com/common/en/articles/syslog-described.php). Within the header, you will see a description of the type such as:

* Priority
* Version
* Timestamp
* Hostname
* Application
* Process id
* Message id

Then, you will see structured-data which have data blocks in the “key=value” format within square brackets. After the SD, you will see the detailed log message, which is encoded in UTF-8.

For example, the [following message](https://www.balabit.com/documents/syslog-ng-ose-latest-guides/en/syslog-ng-ose-guide-admin/html/concepts-message-ietfsyslog.html):

* <34>1 2003-10-11T22:14:15.003Z mymachine.example.com su - ID47 - BOM'su root' failed for lonvick on /dev/pts/8

Corresponds to the following format:

* <priority>VERSION ISOTIMESTAMP HOSTNAME APPLICATION PID MESSAGEID STRUCTURED-DATA MSG

Example of a system error message:

May 11 10:40:48 scrooge disk-health-nurse[26783]: [ID 702911 user.error] m:SY-mon-full-500 c:H : partition health measures for /var did not suffice - still using 96% of partition space

The message can be split in to the following columns:

Column 1 = "May 11 10:40:48"               > Timestamp  
Column 2 = "scrooge"                       > Loghost  
Column 3 = "disk-health-nurse[26783]:"     > Application/Process  
Column 4 = "[ID 702911 user.error]"        > Syslog facility.level  
Column 5 = "m:SY-mon-full-500"             > Message ID  
Column 6 = "c:H : partition health..."     > Message [possibly including rid, sid, ip]

<https://techzone.ergon.ch/syslog-message-format>

win32 <https://www.accadius.com/using-python-read-windows-event-logs-multiple-servers/>