Dokumentation MT Watchfolder

Mit dem Watchfolder-Skript werden Studio-Rückpakete auf MT-Hinweise geprüft. Dieses Dokument unterstützt Entwickler bei der **Pflege und Anpassung des Codes für weitere Anwendungen**. Für eine Installationsanleitung siehe:

* README.md

Für eine Einführung in den Betrieb siehe:

* MT-Watchfolder - Tool-Präsentation.mp4
* Kurzanleitung MT Watchfolder.docx

# Übersicht

Das Script nutzt das watchdog-Paket und besteht aus drei Modulen:

|  |  |  |
| --- | --- | --- |
| Modul | Funktion | Anmerkungen |
| Watchfolder | * Lade Pfade zu Kundenordnern * Überwache Pfade auf OS-Notifications * Übergebe Ereignispfade an Processor zur Prüfung | Siehe watchdog observer |
| Processor | * Lade zusätzliche Filter und Blacklist * Lies Provider-Angaben in Arbeitsdateien und übergib an Logging * Prüfe Angaben anhand der Blacklist * Erstelle Warnhinweis und Log bei Match | Siehe watchdog event handler, zipfile, re |
| Provider-Logging | * Logge Start des Skripts * Logge Provider-Angaben * Logge MT-Warnungen * Erstelle tägliche neue Log-Datei | Siehe logging |

Die Einrichtung des Scripts auf einem Server erfolgt in zwei Schritten:

|  |  |  |
| --- | --- | --- |
| Schritt | Funktion | Anmerkungen |
| Ausführbare Datei | * Wandele das Python-Skript in eine ausführbare .EXE-Datei um | Siehe pyinstaller,  nicht hier dokumentiert |
| Service-Manager | * Richte die Anwendung als Dienst ein * Starte den Dienst automatisch beim Win-Startup * Starte den Dienst neu bei einer Ausnahme * Logge die Skript-Ausgabe auf die Konsole in einer externen Datei | Siehe nssm |

# Watchfolder

Die Observer-Instanz verwendet ein Verfahren, das als „synchron“ bezeichnet wird und auf Notifications der WinAPI in Form von ReadDirectoryChangesW zugreift[[1]](#endnote-1),[[2]](#endnote-2),[[3]](#endnote-3). Von den an den Processor übergebenen Ereignissen werden effektiv nur solche mit dem Flag: FILE\_NOTIFY\_CHANGE\_CREATION beachtet. Dafür wurde die Processor-Methode on\_created() überschrieben.

Das watchdog-Paket bietet außerdem einen OS-unabhängigen Polling-Observer als Fallback-Option. Polling, also regelmäßige DIR-Abfragen, wurde nicht getestet und sollte nur in absoluten Ausnahmefällen zum Einsatz kommen. Dagegen sprechen neben der hohen Latenz vor allem die zu erwartenden Folgen auf die Netzwerk-Performance.

Beim Starten des Skripts wird eine Config-Datei mit Kundenordnern geladen und als globale Variable gespeichert. Im Snippet in Abbildung 1 werden die Ordnerpfade mithilfe der schedule() Methode einzelnen Events zugewiesen und vom Observer rekursiv auf Notifications des OS überwacht. Mit unschedule() oder unschedule\_all() können Event-Watchs vom Observer wieder entfernt werden.

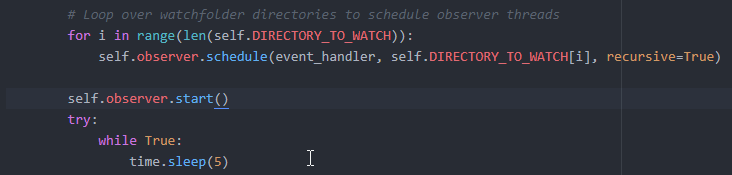


Abbildung 1: Überwachung mehrerer Ordner

## Bekannte Probleme

Es gibt Meldungen, dass ReadDirectoryChangesW nicht 100 % zuverlässig arbeitet. Insbesondere bei sehr umfangreiche Dateioperationen können Events „verlorengehen“. Um das zu prüfen, sollten die mit den Lieferungen innerhalb eines Zeitraums verglichen werden.

❗ Es ist dringend angeraten, die Anzahl der zu überwachenden Ordner gering zu halten, um Performance-Probleme zu vermeiden. Rekursionen sollten soweit möglich vermieden werden, beispielsweise indem wir:

* Zu überwachende Ordner explizit nennen
* Nicht relevante Ordner in Ordnerstrukturen filtern

# Processor

Bei dem Processor handelt es sich um einen Handler, der auf Dateiendungen achtet. Neben der Klasse PatternMatchingEventHandler bietet sich noch RegexMatchingEventHandler an. Mit letzterem lassen sich etwa Elemente im Dateipfad mit berücksichtigen.

Die Verarbeitungsroutine unterscheidet zwischen Einzeldateien und Paketen. Bei Paketen muss darauf geachtet werden, dass beim Packen zwei Ereignisse entstehen:

1. Start-Ereignis: Ein leerer Zip-Container erstellt und dieser sukzessive mit Dateien befüllt wird. Das Start-Ereignis führt zu einem FileNotFoundError und die Prüfung wird beendet.
2. Stop-Ereignis: Das Paket ist gepackt und die enthaltenen Dateien lassen sich lesen. Die eigentliche Prüfung besteht aus zwei Stufen:
   * get\_providers(): Regex-Matches werden von einer Liste mit Tuples („origin“, „origin-system“) in ein Dictionary-Format origin: {origin-system: count} überführt.
   * check\_against\_blacklist(): Provider-Angaben werden mit bekannten MT-Providern verglichen

# Provider-Logging

Das Logging-Modul dient in erster Linie dem Logging von Provider-Daten und wird beim Ausführen mit watchfolder.py als main aktiviert.

Eine Vorlage wird mit einem Formatter erstellt[[4]](#endnote-4). Das gewählte Format entspricht dem einer \*.TAB-Datei.

Wie in Zeile 63 im Snippet in Abbildung 2 ersichtlich, wird jedem Eintrag ein Level zugewiesen. Die Reihenfolge der Level lautet (NOTSET, DEBUG, INFO, WARNING, ERROR, CRITICAL)[[5]](#endnote-5). Mit basicConfig() lässt sich festlegen, ab welcher Stufen das Logging anspringt.

Mit der Klasse TimedRotationFileHandler wurde ein Intervall von einem Tag festgelegt. Ein Beispiel: Wenn am Dienstagmorgen ein neuer Eintrag erstellt wird, werden die Einträge vom Montag in eine Datei mit entsprechender Datumsangabe transferiert. Wenn am Montag kein neuer Eintrag erstellt wurde, werden die Einträge vom letzten Vortag mit Einträgen (bspw. Freitag) in eine neue Datei transferiert.

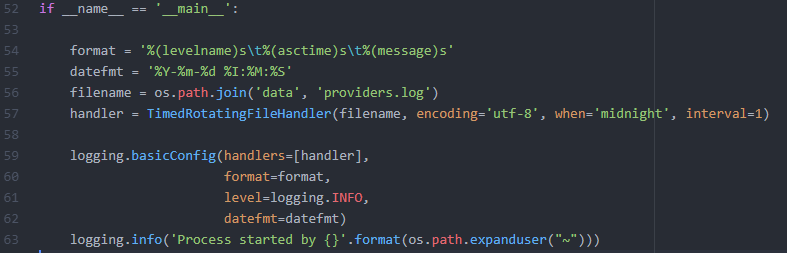


Abbildung 2: Logging-Format und Level

# Service-Bereitstellung

Die App wird als Service auf dem Server <app1> eingerichtet. Die Einrichtung erfolgt mit dem Service-Manager nssm[[6]](#endnote-6) + watchfolder.exe. Die alternative Einrichtung mit Service-Manager + watchfolder.py + python.exe wurde nicht getestet. Die wichtigsten Vorteile eines Services sind, dass das Skript:

* unabhängig von einem Benutzer läuft
* bei einer Ausnahme vom Service-Manager automatisch neu gestartet wird.

## Einrichtung

1. Für Hinweise zur Benutzung siehe nssm-README[[7]](#endnote-7) und nssm-Cheatsheet[[8]](#endnote-8).
2. Die Datei nssm.exe lokal ablegen und mit der Kommandozeile zum Pfad navigieren.
3. nssm service editor GUI starten mit:

>>>nssm install <Name des Dienstes>

1. Anschließend Pfad und Verzeichnis der ausführenden Datei angeben.
2. Unter Details den Startup Type auf Automatic (Delayed Start) setzen. Die Verzögerung beim Windows-Startup beträgt 1 Minute.
3. Mit Install Service Einrichtung abschließen

Hinweis: Alle Dienst-Parameter werden in der Registry abgelegt unter:

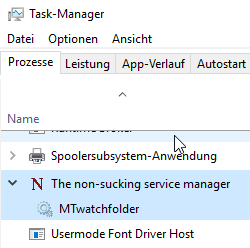
HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\<Name des Dienstes>

## Manuelles Starten und Beenden, Anpassen und Entfernen

Der Dienst kann über den Taskmanager > Dienste manuell gestartet und beendet werden. Im Taskmanager wird der Prozess unter der nssm Host-Anwendung angezeigt (siehe Abbildung 3: Dienst-Zugriff).

Die entsprechenden Befehle auf der nssm-Kommandozeile lauten:

>>>nssm start <Name des Dienstes>

>>>nssm stop <Name des Dienstes>

Bei einem Fehler erfolgt der Shutdown der Anwendung stufenweise: Zuerst wird die Eingabe von CTRL+C simuliert, anschließend das Schließen des Anwendungsfensters. Ansonsten werden alle Threads beendet und bei Nichterfolg ein Kill-Signal gesendet.

Weitere nützliche Befehle sind:

>>>nssm status <Name des Dienstes>

>>>nssm edit <Name des Dienstes>

>>>nssm remove <Name des Dienstes>

## Logging und Troubleshooting

Abbildung 3: Dienst-Zugriff

Die Logs des Service-Managers lassen sich über die Windows-Ereignisanzeige verfolgen. Typische Probleme sind:

* Pfad wird nicht erkannt 🡪 UNC-Netzwerkpfade verwenden
* Skript wird sofort wieder beendet 🡪 data-Unterordner mit config.ini fehlt

Um den Betrieb des Skriptes auch ohne Anmeldung auf <app1> verfolgen zu können, sollte die Konsolenausgabe des Skriptes an eine Log-Datei auf dem Netzwerkordner weitergeleitet werden (siehe Abbildung 4: Ausgabe an Log-Datei).

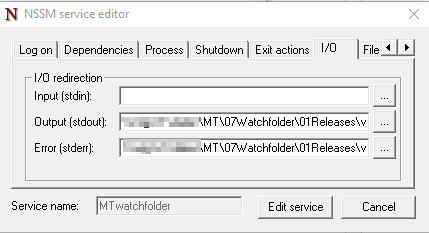


Abbildung 4: Ausgabe an Log-Datei

Hinweis: Von einer Online-Ausgabe, d. h. der Ausgabe bei laufendem Betrieb wird wegen der Komplexität des Zugriffs abgeraten.

Um zu vermeiden, dass das Log beim Neustart des Skripts überschrieben wird, sollte File-Rotation aktiviert sein. Die Option generiert Log-Dateien in definierten Intervallen. Sie ist vergleichbar mit der Klasse TimedRotationFileHandler beim Provider-Logging.

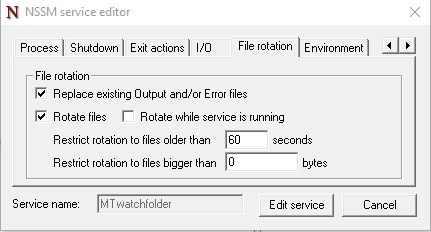


Abbildung 5: Log-Generator

# Tests

Funktionstests- und Langzeittests: Der Testzeitraum für das Skript betrug 14 Tage. Dabei wurden ca. 8000 Ordner überwacht. Es war ursprünglich vorgesehen, den Observer ca. 30.000 Ordner überwachen zu lassen.

Das Skript zum Testen der Skript-Funktionen liegt im tests-Unterordner und nutzt pytest[[9]](#endnote-9). Befehl zum Starten des Tests:

>>> pytest watchfolder.py

1. Referenz des Watchdog-Pakets <https://pythonhosted.org/watchdog/installation.html#supported-platforms-and-caveats> [↑](#endnote-ref-1)
2. Observer-Implementierungen: <http://timgolden.me.uk/python/win32_how_do_i/watch_directory_for_changes.html> [↑](#endnote-ref-2)
3. winapi im watchdog Source-Code: <https://github.com/gorakhargosh/watchdog/blob/master/src/watchdog/observers/winapi.py> [↑](#endnote-ref-3)
4. <https://docs.python.org/3/library/logging.html#logging.Formatter.formatTime> [↑](#endnote-ref-4)
5. Logging-Modul: <https://docs.python.org/3/library/logging.html> [↑](#endnote-ref-5)
6. <https://nssm.cc/> [↑](#endnote-ref-6)
7. NSSM-Readme: <https://github.com/kirillkovalenko/nssm/blob/master/README.txt> [↑](#endnote-ref-7)
8. NSSM-Cheatsheet: <https://gist.github.com/magnetikonline/2217fd95cf15a0324696> [↑](#endnote-ref-8)
9. <https://docs.pytest.org/en/latest/> [↑](#endnote-ref-9)