

# Univention LDAP Listener API

Thema:	Beschreibung der Univention LDAP Listener API	
Datum:	11. April 2007	
Seitenzahl:	7	
Versionsnummer:	639	
Autoren:	Janis Meybohm Stefan Gohmann	meybohm@univention.de gohmann@univention.de

## Inhaltsverzeichnis

<b>1</b>	<b>Einführung</b>	<b>3</b>
<b>2</b>	<b>Modul-Aufbau</b>	<b>3</b>
2.1	Verzeichnisse . . . . .	3
2.2	Modul Definition . . . . .	3
2.3	Funktionen der Listener-Klasse . . . . .	4
2.4	Funktionen der Debug-Klasse . . . . .	5
2.5	Funktionen des Moduls . . . . .	5
<b>3</b>	<b>Beispiel-Modul - PrintUsers</b>	<b>6</b>

## 1 Einführung

Der Listener ist Teil des Univention Notifier-Listener Mechanismus. Er erhält von den Notifier Systemen in der Domäne die Adressen der geänderten LDAP-Objekte und kann anschließend die Objekte im LDAP des jeweiligen Notifier Systems abfragen. Über eine interne Datenbank kann zusätzlich auf die vorherigen Werte des jeweiligen Objektes zugegriffen werden.

Der Listener kann über eine Plugin-Architektur erweitert werden, so dass bei Änderungen bestimmter Objekte im LDAP verschiedene Aktionen ausgeführt werden können. (Z.B. Neuschreiben von Konfigurationsdateien, Zwischenspeicherung globaler LDAP-Einstellungen in lokalen Variablen oder Neustarts von Diensten.)

Module für die Plugin-Architektur müssen in Python implementiert werden.

Treten Änderungen auf, legt der Listener durch definierte Filter und Attribute in den Modulen fest, ob das jeweilige Modul für die anstehende Änderung zuständig ist. Es können auch mehrere Module bei bestimmten Änderungen verwendet werden.

Die Module werden unterhalb von `/usr/lib/univention-ldap-listener/system/` gespeichert.

Alle Module legen nach ihrer Initialisierung so genannte Handler-Dateien in `/var/lib/univention-ldap-listener/handlers/` ab. Diese geben den Status ihres Moduls in einem Zahlenwert an, eine 3 bedeutet beispielsweise, dass das Modul erfolgreich initialisiert wurde. Ist die Handler-Datei beim Start des Listeners nicht vorhanden, wird das Modul neu initialisiert. Bei der Initialisierung des Moduls wird das LDAP des Notifier Systems komplett mit dem definierten Filter des Moduls durchsucht und für jedes gefundene LDAP-Objekt das Modul aufgerufen.

## 2 Modul-Aufbau

### 2.1 Verzeichnisse

Die Listener-Module müssen im Verzeichnis `/usr/lib/univention-ldap-listener/system/` abgelegt werden. Beim Starten lädt der Listener alle Module aus diesem Verzeichnis.

### 2.2 Modul Definition

Im Header eines jeden Moduls werden die folgenden Informationen festgelegt:

```
name='Ihr Modulname '  
description='Ihre Beschreibung '  
filter='(objectClass=posixAccount) '  
attributes=['uid', 'uidNumber', 'cn']
```

Variable	Erklärung
name	Der Name des Moduls.
description	Eine kurze Beschreibung des Moduls.
filter	Definiert den LDAP-Filter, bei dessen Übereinstimmung das Modul verwendet wird. Es ist zu beachten, dass dieser Filter - im Gegensatz zu <code>ldapsearch - case-sensitive</code> ist. <b>NICHT</b> -Klauseln wie <code>!(uid=0)</code> oder <code>(uid!=0)</code> werden zur Zeit noch nicht unterstützt.
attributes	Eine Liste von Attributen bei deren Änderung das Modul aufgerufen wird. Ist die Liste leer, wird das Modul bei Änderungen an jedem Attribut gestartet.

Nach der Definition der Header folgen die zu importierenden Python-Module. Mit `import listener` importieren Sie die Listener-Klasse um Zugriff auf die Baseconfig- und Listener-Funktionalitäten zu haben. Mit `listener.baseConfig['<Variable>']` kann auf alle Baseconfig-Variablen zugegriffen werden. Sie haben auch die Möglichkeit über die Klasse `univention.debug` zusätzliche Debug Ausgaben zu erzeugen, die dann in der `listener.log` erscheinen.

## 2.3 Funktionen der Listener-Klasse

Für einige Operationen - etwa Dateioperationen in Systemverzeichnissen oder Neustarts von Diensten - kann es nötig sein, das Modul kurzzeitig mit **root**-Rechten zu betreiben. Wenn die **listener**-Klasse importiert wurde, kann dies durch `listener.setuid(0)` bzw. `listener.unsetuid()` durchgeführt werden.

- `listener.setuid(<uid>)`: Mit `setuid()` sind Sie in der Lage Ihr Modul für kurze Zeit mit der UserID von **root** laufen zu lassen.
- `listener.unsetuid()`: Setzt die UserID wieder zurück.
- `listener.run(<Programm>, <Argumente>, <uid>, <Warten>)`: Mit `run` haben Sie die Möglichkeit Dienste oder Programme direkt aus dem Modul heraus zu starten.
  - `<Programm>`: Gibt das zu startende Programm an. (Für Dienste z.B. **initscript**)
  - `<Argumente>`: Eine Liste von Argumenten die dem zu startendem Programm übergeben werden.
  - `<uid>`: Die UserID unter der das Programm gestartet werden soll. Wird es nicht übergeben, wird das Programm mit der UID von `root` gestartet.
  - `<Warten>`: Gibt als Booleschen Wert (**true**, **false**) an, ob das Modul warten soll bis das auszuführende Programm terminiert ist oder der Ablauf sofort fortgesetzt wird.

## 2.4 Funktionen der Debug-Klasse

- `univention.debug.debug(<id>, <level>, <string>)`
  - `<id>`: Beschreibt den Dienst in dessen Log-Datei die Ausgaben erscheinen sollen, in diesem Fall würde `univention.debug.LISTENER` in `/var/log/univention/listener.log` schreiben.
  - `<level>`: Gibt den zu verwendenden Debug-Level an, zur Auswahl stehen (z.B.: `univention.debug.ERROR`):
    - \* `INFO`: Für grundsätzliche Ausgaben, wird erst ab Log-Level 4 ausgegeben.
    - \* `WARN`: Warnmeldungen, werden ab Log-Level 1 ausgegeben.
    - \* `ERROR`: Fehlermeldungen, werden immer in die Log-Datei und zusätzlich per **syslog** geschrieben.
  - `<string>`: Der auszugebende Text als Zeichenkette.

## 2.5 Funktionen des Moduls

- `def handler(dn, new, old)`: Diese Funktion wird aufgerufen wenn der Filter zutrifft. Wird nur `new` übergeben, wurde das Objekt neu angelegt. Wird nur `old` übergeben, wird es gelöscht. Werden `new` und `old` übergeben wird das zu behandelnde Objekt verändert.
- `def postrun()`: Wird 15 Sekunden nachdem der Handler fertig ist aufgerufen um z.B. Dienste neu zu starten.
- `def clean()`: Wird vor einem Resync bzw. vor dem Entfernen eines Moduls aufgerufen.
- `def initialize()`: Wird bei der Initialisierung eines Moduls aufgerufen.

Die Module werden beim Starten des LDAP-Listeners nicht neu initialisiert. Wollen Sie ein Modul neu initialisieren verwenden Sie den Befehl:

```
univention-ldap-listener-ctrl resync <Modulname>
```

Der Listener wird angehalten (dies kann ein paar Sekunden dauern), die Handlers-Datei des Moduls wird gelöscht und anschließend das Modul neu geladen. Wenn Sie einmal alle Module neu initialisieren wollen, kann dies durch Stop des Listener, Löschen der bestehenden Handler-Dateien unterhalb von `/var/lib/univention-ldap-listener/handlers/` und anschließendem Listener-Start geschehen.

In `/var/log/univention/listener.log` sind Debug-Ausgaben des Moduls zu finden. Detailliertere Ausgaben können durch Erhöhen des Loglevels in Univention Baseconfig erzielt werden:

```
univention-baseconfig set ldap/debug/level=4
```

Der Log-Level sollte zeitnah auf 1 zurückgesetzt werden, da sonst die Log-Datei sehr schnell anwachsen kann.

### 3 Beispiel-Modul - PrintUsers

Dieses Beispiel-Modul wartet auf Änderungen an Nutzer-Objekten und speichert diese in einer Datei UserList.txt in /tmp.

```
name='printusers'
description='print all names/users/uidNumbers into a file'
filter='(&(|(&(objectClass=posixAccount)(objectClass=shadowAccount)) \
        (objectClass=univentionMail) (objectClass=sambaSamAccount) \
        (&(objectClass=person)(objectClass=organizationalPerson) \
        (objectClass=inetOrgPerson))))'
attributes=['uid', 'uidNumber', 'cn']

import listener
import os
import sys
import univention.debug
import univention.utf8
import types

sys.setdefaultencoding('iso8859-15')
file='/tmp/UserList.txt'

def handler(dn, new, old):

# ----- Funktion um in eine Textdatei zu schreiben, mit setuid(0) da
# der Handler als root ausgeführt wird:
def writeit(cn, uid, uidnr, com):
    listener.setuid(0)
    try:
        o=open(file,'a')
        try:
            if com and com != 'indent':
                o.write('Name: "%s"\t\tUser: "%s"\t\tUID: "%s"\t%s\n'%
                        (univention.utf8.decode(cn), uid, uidnr, com))
            elif com == 'indent':
                o.write('\tName: "%s"\t\tUser: "%s"\t\tUID: "%s"\n'%
                        (univention.utf8.decode(cn), uid, uidnr))
            else:
                o.write('Name: "%s"\t\tUser: "%s"\t\tUID: "%s"\n'%
                        (univention.utf8.decode(cn), uid, uidnr))
        except Exception, e:
            univention.debug.debug(univention.debug.LISTENER, \
                                   univention.debug.ERROR, 'Failed to write to file \
                                   "%s": %s'%(file, str(e)))
    except Exception, e:
        univention.debug.debug(univention.debug.LISTENER, \
                               univention.debug.ERROR, 'Failed to open "%s": \
                               %s'%(file, str(e)))
    o.close()
    listener.unsetuid()

# ----- Wenn ein Objekt veraendert wird:
```

```
if new and old:
    # Schließt Computerobjekte aus:
    if not new['uid'][0][-1:] == '$' and not old['uid'][0][-1:] == '$':
        univention.debug.debug(univention.debug.LISTENER, \
            univention.debug.INFO, 'Edited User "%s"'%old['uid'][0])
        if old['uid'][0] == 'root' or old['uid'][0] == 'spam':
            writeit(old['cn'][0], old['uid'][0], '****', 'edited. \
                Is now:')
            writeit(new['cn'][0], new['uid'][0], '****', 'indent')
        else:
            writeit(old['cn'][0], old['uid'][0], old['uidNumber'][0], \
                'edited. Is now:')
            writeit(new['cn'][0], new['uid'][0], new['uidNumber'][0], \
                'indent')

# ----- Wenn ein neues Objekt erzeugt wird:
# (Nach einer Initialisierung sind alle Benutzer "neu")
if new and not old:
    if not new['uid'][0][-1:] == '$':
        univention.debug.debug(univention.debug.LISTENER, \
            univention.debug.INFO, Added User "%s"'%new['uid'][0])
        if new['uid'][0] == 'root' or new['uid'][0] == 'spam':
            writeit(new['cn'][0], new['uid'][0], '****', 'added.')
        else:
            writeit(new['cn'][0], new['uid'][0], new['uidNumber'][0], 'added')

# ----- Wenn ein Objekt geloescht wird:
if old and not new:
    if not old['uid'][0][-1:] == '$':
        univention.debug.debug(univention.debug.LISTENER, \
            univention.debug.INFO, Removed User "%s"'%old['uid'][0])
        if old['uid'][0] == 'root' or old['uid'][0] == 'spam':
            writeit(old['cn'][0], old['uid'][0], '****', 'removed')
        else:
            writeit(old['cn'][0], old['uid'][0], old['uidNumber'][0], 'removed')

# ----- Initialisierungsfunktion, die Output-Datei wird
# ----- geloescht, falls vorhanden.
def initialize():
    if os.path.exists(file):
        try:
            listener.setuid(0)
            os.remove(file)
            listener.unsetuid()
            univention.debug.debug(univention.debug.LISTENER, \
                univention.debug.INFO, 'Successfully deleted: "%s"'%file)
        except Exception, e:
            univention.debug.debug(univention.debug.LISTENER, \
                univention.debug.WARN, 'Failed to delete file: "%s": \
                    %s'% (file, str(e)))
    else:
        univention.debug.debug(univention.debug.LISTENER, \
            univention.debug.INFO, 'File: "%s" does not exist, going to \
                create it!'%file)
```