

Univention Admin API

Thema:	Erstellen eigener Module für Univention Admin	
Datum:	12.06.06	
Anzahl Seiten:	16	
Autoren:	Ingo Steuer Andreas Büsching	steuer@univention.de buesching@univention.de
Zusammenfassung:	Dokumentation des Aufbaus und der API von Univention Admin Modulen zur Verwendung mit dem Web-Frontend sowie der Kommandozeile.	
Verteiler:		

Inhaltsverzeichnis

1 Einführung.....	3
2 Module Univention Admin.....	3
2.1 Übersicht.....	3
2.2 Aufbau.....	3
3 Beispiel.....	7
3.1 Python Modul.....	8
3.2 LDAP Schema.....	14
4 Installation.....	15
4.1 Python Modul.....	15
4.2 LDAP Schema.....	16
4.3 Debian-Paket.....	16

1 Einführung

Univention Admin verwendet zur Abbildung der LDAP-Objekte eine flexible und erweiterbare Struktur aus Python-Modulen. Zusätzlich einzubindende Module werden nach Ablage im Dateisystem durch Univention Admin erkannt und zur Verwendung an Kommandozeile und Web-Administration angeboten.

An der Web-Schnittstelle können die zusätzlichen Module über den eingebauten Wizard, der z.B. Benutzer, Gruppen und Rechner verwaltet, oder über eigene Web-Module verwaltet werden. Die Entwicklung eigener Module für die Web-Administration erlaubt zusätzlich die Konfiguration von LDAP-Objekten oder Bereichen, die mit Admin-Modulen nicht abzubilden sind.

Unabhängig davon gibt es auch für Univention Console die Möglichkeit, serverbezogene Dienste durch ein eigenes Modul zu konfigurieren. Der Aufbau ist von den Web-Modulen abgeleitet und wird in der Technischen Dokumentation 'Univention Console API' im Detail beschrieben.

2 Module Univention Admin

2.1 Übersicht

Univention Admin verwendet zur Abbildung von LDAP-Objekten eine eigene Modulstruktur. Im Regelfall entspricht eines dieser Module einem LDAP-Objekt (z.B. einem Benutzer, einer Gruppe oder einem Container).

Die Module sind strukturiert nach Aufgabenbereichen im Verzeichnisbaum abgelegt und werden über diese Struktur vom Kommandozeilen-Frontend angeboten. Die Dateiablage findet sich unterhalb des Verzeichnisses

/usr/lib/python2.4/site-packages/univention/admin/handlers/

Die Module für die Verwaltung der verschiedenen Rechnerobjekte befinden sich unterhalb des Ordners ***computers***, z.B. ***computers/windows.pyo***. Dieses Objekt kann dann von der Kommandozeilenschnittstelle durch ***computers/windows*** angesprochen werden.

Eigene Module sollten nach Möglichkeit in einem eigenen Unterverzeichnis abgelegt werden, um Konflikte mit zukünftigen Standardmodulen zu vermeiden. Im Verzeichnis muss die Datei ***__init__.py*** existieren, damit die Module gefunden werden.

2.2 Aufbau

Ein Modul besteht aus der Definition der Modul-Attribute und der Einführung einer Klasse ***object***, die von ***univention.admin.simpleLdap*** abgeleitet wird. In der folgenden Dokumentation des Aufbaus eines Univention Admin Moduls wird mit einer ausführlichen Beschreibung der zu definierenden Variablen begonnen. Im darauf folgenden Abschnitt wird die Klasse ***object*** genauer betrachtet, d.h. es werden

notwendige Definitionen und Funktionen in der Klasse aufgelistet. Abschließend werden noch zwei optionale Funktionen definiert und erklärt.

Globale Variablen

Im folgende werden die globalen Variablen definiert, die in einem Modul für den Univention Admin besondere Bedeutungen haben. Dabei wird zwischen zwingend notwendig und optionalen Variablen unterschieden. Die folgende Liste enthält die erforderlichen Variablen, die jedes Modul definieren muss:

- *module*
String, der dem Namen des UCS-Moduls entsprechen muss, z.B. *computers/computer*
- *operations*
Liste von Strings, enthält alle mit diesem Objekt erlaubten LDAP-Operationen, die vollständige Liste ist: ['add','edit','remove','search','move']
- *short_description*
Diese Beschreibung wird im Web-Frontend des Univention Admin als Name eingesetzt. Im Bereich 'Navigation' wird dieser Text in der Auswahlliste für mögliche Objekttypen angezeigt.
- *childs*
Gibt an ob es sich bei diesem LDAP-Objekt um einen Container handelt. In diesem Fall wird diese Variable auf den Wert 1 gesetzt und andernfalls auf 0.
- *long_description*
Eine ausführliche Beschreibung des Moduls
- *options*
Ein Dictionary, das Optionen definiert, die genutzt werden können um einzelne Attribute (*properties*) dieses Moduls auszublenden. In diesem Dictionary werden die Optionen (als *univention.admin.option* Objekte) einer eindeutigen Zeichenkette zugeordnet, die später zur Referenzierung verwendet wird (siehe *property-descriptions*).
Beispiel:

```
options = { 'opt1' : univention.admin.option(  
  short_description = 'short description',  
  default = 1 )  
}
```


Jede Option bekommt zwei Parameter:
 - *short_description*: eine kurze Beschreibung der Option, die beispielsweise im Web-Frontend des Univention Admin als beschreibender Text zu den Eingabefeldern genutzt wird.
 - *default*: Definiert ob diese Option standardmäßig aktiviert sein soll. Eine 1 steht dabei für aktiv und eine 0 für inaktiv.

- *property-descriptions*

Dieses Dictionary enthält alle möglichen Attribute, die dieses Modul zur Verfügung stellt. Dabei werden die Attribute (als *univention.admin.property* Objekte), wie schon die Optionen, über eine eindeutige Zeichenkette als Schlüssel referenziert. Ein solches Modul-Attribut entspricht i.d.R. einem LDAP-Attribut.

Beispiel:

```
property_descriptions = { 'prop1' : univention.admin.property(  
    short_description = 'name'  
    long_description = 'long description',  
    syntax = univention.admin.syntax.string,  
    multivalue = 0,  
    required = 1,  
    may_change = 1,  
    identifies = 0,  
    dontsearch = 0,  
    show_in_lists = 0,  
    one_only = 0,  
    options = [ 'opt1' ] )
```

Die Parameter haben folgende Bedeutung:

- *short_description*: Eine kurze Beschreibung, die beispielsweise im Web-Frontend des Univention Admin als beschreibender Text zu den Eingabefeldern genutzt wird.
- *long_description*: Eine ausführlichere Beschreibung, die im Web-Frontend des Univention Admin für die Tooltips genutzt wird.
- *syntax*: Dieser Parameter definiert den Typ eines Attributs. Die folgende Liste enthält die wichtigsten Beispiele für mögliche Werte. All diesen Werten ist die Prefix *univention.admin.syntax.* voranzustellen:
 - *string*: eine beliebige Zeichenkette
 - *integer*: ein ganzzahliger positiver Wert
 - *boolean*: kann nur eine 0 oder eine 1 enthalten
 - *filesize*: ein ganzzahliger positiver Wert mit einer optional folgenden Angabe der Einheit. Unterstützt werden die Einheiten: B, KB, MB, GB (die Groß- und Kleinschreibung ist irrelevant)
 - *phone*: Telefonnummern
 - *uid, uid_unlauts*: ein Benutzername; wahlweise mit (uid_umlauts) oder ohne Umlaute (uid)
 - *gid*: ein Gruppenname
 - *sharePath*: ein Pfad für eine Freigabe

- *passwd, userPasswd*: zwei Varianten von Passwörtern. Während Passwörter vom Typ *passwd* eine Mindestlänge von 8 Zeichen haben müssen reichen für den Typ *userPasswd* Passwörter einer beliebigen Länge > 0 .
- *hostName*: ein Rechnername nach der Definition aus RFC 952
- *ipAddress*: eine IP Adresse (Version 4)
- *hostOrIP*: ein Rechnername oder eine IP Adresse (Version 4)
- *netmask*: eine Netzmaske
- *absolutePath*: ein absoluter Pfad nach UNIX-Syntax, d.h. der Wert muss mit einem '/' anfangen
- *emailAddress, emailAddressTemplate*: eine E-Mail Adresse mit (*emailAddressTemplate*) oder ohne (*emailAddress*) Nutzernamen
- *date*: ein Datum in verschiedenen Kurzschreibweisen: TT.MM.JJ oder TT.MM.JJJJ oder JJJJ-MM-TT

Anhand dieser Typdefinition kann der Univention Admin die angegebenen Werte für das Attribut überprüfen und bei ungültigen Werten eine detaillierte Fehlermeldung liefern. Zusätzlich zu den vordefinierten Typdefinitionen können auch eigene Typen erstellt werden. Dies sollte im Normalfall nicht notwendig sein.

- *multivalue*: Kann die Werte 1 oder 0 annehmen. Ist dieser Parameter auf 1 gesetzt handelt es sich bei dem Wert des Attributs um eine Liste. Der Parameter *syntax* gibt in diesem Fall den Typ der Elemente dieser Liste an.
- *may_change*: Ist dieser Parameter auf 1 gesetzt kann der Wert dieses Attributs auch nach der Erstellung des Objektes verändert werden.
- *options*: Eine Liste von Schlüsselwörtern, die Optionen identifizieren mit denen dieses Attribut ein- bzw. ausgeblendet werden kann.
- *identifies*: diese Option sollte bei genau einem Attribut eines Moduls gesetzt sein. Sie wird verwendet um anhand des zugehörigen Attributs einen eindeutigen Namen zu bilden.
- *layout*
Die Attribute eines Objektes können in Gruppen angeordnet werden. Diese werden beispielsweise im Univention Admin als Reiter dargestellt. In der Variable *layout* wird diese Struktur definiert.
- *mapping*
Ist vom Typ *univention.admin.mapping* und definiert die Abbildung der Attribute des Univention Admin Moduls auf LDAP-Attribute (siehe Beispiel in Kapitel 3).

Folgende Angaben sind optional und müssen nur definiert werden wenn das Modul diese speziellen Attribute besitzt:

- *virtual*
Module, die diese Variable auf 1 setzen sind eine Art Hilfsmodul für andere Module,

die keine zugehörigen LDAP-Objekte haben. Ein Beispiel hierfür ist das Modul *computers/computer*, das als Hilfsmodul für alle Rechnertypen dient.

- *template*
Ein Modul, das diese Variable auf 1 setzt, bietet die Möglichkeit Vorgabewerte für die Attribute von anderen Modulen zu definieren. Ein Beispiel hierfür ist die Benutzer-Vorlage (Module *settings/usertemplate*). Im Web-Frontend des Univention Admin können im Bereich Univention/templates neue Vorlagen erstellt werden. Eine solche Vorlage kann dann beispielsweise beim Anlegen eines Benutzers ausgewählt werden und die darin definierten Werte werden als Vorgaben in die Eingabemasken übernommen.

Die Klasse *object*

Die Klasse *object* eines Moduls ist die Schnittstelle zum Univention Admin bei LDAP Operationen wie *create*, *modify* und *remove*. Diese Klasse unterstützt den Univention Admin bei der Abbildung der definierten Attribute des Moduls auf LDAP-Objekte und Attribute. Dafür ist eine vordefinierte API der Klasse einzuhalten.

Die Basisklasse *univention.admin.handlers.simpleLdap* bietet für einfache LDAP-Objekte die wesentliche Funktionalität, so dass hierfür nur noch wenige Anpassungen notwendig sind. Um Anpassungen vornehmen zu können, bietet die *simpleLdap* Klasse die Möglichkeit vor und nach der LDAP-Operation in den Verlauf einzugreifen in dem vordefinierte funktionen aufgerufen werden. Beispielsweise wird vor dem Anlegen eines LDAP-Objekte die Funktion *_ldap_pre_create* aufgerufen und nach dem Vorgang die Funktion *_ldap_post_create*. Diese *pre*- und *post*-Funktionen gibt es zusätzlich noch für die Operationen *modify*, *move* und *delete* (siehe Beispiel im Kapitel 3).

Die Funktionen *identify* und *lookup*

Diese Funktionen werden genutzt um einerseits bei Suchanfragen aus dem Web-Frontend des Univention Admin die entsprechenden Objekte zu finden (*lookup*) und um LDAP-Objekte einem Univention Admin Module zuzuordnen. In Kapitel 3 werden Beispielimplementierungen für diese zwei Funktionen vorgestellt, die für einfache LDAP-Objekte nur leicht modifiziert werden müssen.

3 Beispiel

In diesem Kapitel wird ein Beispielmodul für den Univention Admin vorgestellt, das auch als Debian Paket „univention-admin-module-example“ für UCS unter <http://download.univention.de/download/addons/documentation/> verfügbar ist.

Ein Univention Admin Modul bestimmt in der Regel immer aus zwei Teilen: Einmal dem Python Modul, das die Implementierung der Schnittstelle zum Univention Admin enthält und einem LDAP Schema, das das zu verwaltende LDAP-Objekt definiert. Im folgenden werden beide Teile beschrieben, wobei der Schwerpunkt bei der Erstellung des Python Moduls und der Installation beider Teile liegt.

3.1 Python Modul

Das Beispielmodul für den Univention Admin demonstriert die rudimentäre Verwaltung von IP Telefonen. Dabei wird versucht mit einem einfach gehaltenen Beispiel möglichst viele der Fähigkeiten eines solchen Univention Admin Moduls aufzuzeigen.

Bevor der eigentliche Quellcode des Moduls kommt müssen einige Univention Python Module importiert werden, die auf jeden Fall notwendig sind:

```
import univention.admin.filter
import univention.admin.handlers
import univention.admin.syntax
```

Diese Liste von Python Modulen kann natürlich noch erweitert werden.

Wie in Abschnitt 2.2 beschrieben werden in einem Univention Admin Modul zu Beginn einige notwendige globale Variablen definiert, die eine Beschreibung des Moduls liefern. Dazu gehört der eindeutige Name und Beschreibungstexte sowie die Variable *childs*, die definiert ob das Objekt noch Kindobjekte haben kann und die Variable *operations*, die festlegt welche LDAP-Operationen mit diesem Modul möglich sind.

```
module = 'test/ip-phone'
childs = 0
short_description = u'IP-Telefon'
long_description = u'Ein Beispiel-Modul für den Univention Admin zur
Verwaltung von IP-Telefonen'
operations=['add', 'edit', 'remove', 'search', 'move']
```

Eine weitere wichtige globale Variable für das Univention Admin Web-Frontend ist *layout*. Damit wird die Anordnung der einzelnen Attribute des Objektes auf die Reiter verteilt und innerhalb der Reiter strukturiert. Die Liste besteht aus Elementen vom Typ *univention.admin.tab*, die jeweils den Inhalt eines Reiters bestimmen. In diesem Fall gibt es einen Reiter Allgemein und einen weiteren Reiter Erweiterungen.

```
layout=[
    univention.admin.tab( u'Allgemein', u'Grundeinstellungen',
        [ [ univention.admin.field( "name" ), univention.admin.field
( "active" ) ],
        [ univention.admin.field( "ip" ), univention.admin.field
( "protocol" ) ],
        [ univention.admin.field( "priuser" ) ] ] ),
    univention.admin.tab( u'Erweiterungen', u'Erweiterte Einstellungen',
        [ [ univention.admin.field( "users" ) ] ] ) ]
```

Hinweis:

Es können nicht mehr als zwei Felder nebeneinander angegeben werden, d.h. ein Listenelement darf nicht aus mehr als zwei Objekten vom Typ *univention.admin.field* bestehen.

Als nächstes sollten die Optionen und Attribute des Moduls definiert werden. In diesem Fall wird eine Option *extended* angelegt, deren Aufgabe später noch erläutert wird. Als Parameter bekommt das Object *univention.admin.option short_description* für eine kurze Beschreibung und *default* mit dem der vordefinierte Wert der Option festgelegt wird. Wenn *default* auf 1 gesetzt wird ist die Option aktiviert und bei 0 deaktiviert.

```
options={
  'extended' : univention.admin.option(
    short_description= u'Erweiterte Einstellungen',
    default=1
  )
}
```

Nach den Optionen werden die Attribute des Moduls festgelegt. Dabei werden die Attribute durch textuelle Beschreibungen, Syntaxdefinitionen und Anweisungen für das Web-Frontend des Univention Admins definiert.

```
property_descriptions={
```

Das Attribut *name* definiert den Rechnernamen, den das IP-Telefon haben soll. Mit dem Parameter *syntax* wird dem Univention Admin mitgeteilt, dass gültige Werte für dieses Attribut der Syntax eines Rechnernamen entsprechen müssen. Weitere vordefinierte Syntaxdefinitionen sind in dem Abschnitt 2.3 zu finden. Eine Besonderheit dieses Attributs ist der Parameter *may_change*, der in diesem Fall auf 0 gesetzt ist. Dadurch ist festgelegt, dass dieses Attribut nach dem Anlegen des Objektes nicht mehr geändert werden kann.

```
'name': univention.admin.property(
  short_description= u'Name',
  long_description= u'Name des Telefons',
  syntax=univention.admin.syntax.hostName,
  multivalue=0,
  options=[],
  required=1,
  may_change=0,
  identifies=1
),
```

active ist ein Beispiel für ein bool'sches Attribut, die nur die Werte '1' oder '0' annehmen kann. In diesem Beispiel bedeutet dies eine Freischaltung oder Sperrung des IP-Telefons. Mit dem Parameter *default='1'* wird das Telefon initial freigeschaltet.

```
'active': univention.admin.property(
  short_description= u'freigeschaltet',
  long_description= u'Ein IP-Telefon kann gesperrt werden',
  syntax=univention.admin.syntax.boolean,
```

```
        multivalue=0,  
        options=[],  
        required=0,  
        default='1',  
        may_change=1,  
        identifies=0  
    ),
```

Das Attribut *protocol* legt fest, welches VoIP Protokoll von dem Telefon unterstützt wird. Dabei wird für dieses Attribut keine vordefinierte Syntaxdefinition angegeben, sondern eine eigens dafür deklarierte Klasse *SynVoIP_Protocols*. Der Quellcode dieser Klasse folgt nach der Auflistung der Attribute. Die Syntax Klasse definiert eine Auswahlliste mit einer vordefinierten Menge an Möglichkeiten. Durch den Parameter *default* wird der Wert mit dem Schlüssel 'sip' vorausgewählt.

```
    'protocol': univention.admin.property(  
        short_description= u'Protokoll',  
        long_description= u'Welches VoIP Protokoll wird von dem Telefon  
        unterstützt',  
        syntax=SynVoIP_Protocols,  
        multivalue=0,  
        options=[],  
        required=0,  
        default='sip',  
        may_change=1,  
        identifies=0  
    ),
```

Das Attribut *ip* legt die IP-Adresse des Telefons fest. Als Syntaxdefinition wird dafür die vordefinierte Klasse *univention.admin.syntax.ipAddress* angegeben. Zusätzlich wird mit dem Parameter *required* erzwungen, dass diesem Attribut ein valider Wert zugewiesen wird.

```
    'ip': univention.admin.property(  
        short_description = u'IP-Adresse',  
        long_description = u'',  
        syntax=univention.admin.syntax.ipAddress,  
        multivalue=0,  
        options=[],  
        required=1,  
        may_change=1,  
        identifies=0  
    ),
```

Das Attribut *priuser* legt den primären Benutzer für ein IP-Telefon fest. Die Besonderheit dieses Attributs liegt in der Syntaxdefinition, die genauso wie *SynVoIP_Protocols* nicht zu den vordefinierten gehört. In diesem Fall handelt es sich allerdings nicht um eine Auswahlliste, sondern um eine Klasse, die einen regulären Ausdruck definiert, der das Muster für valide Werte spezifiziert (Quellcode folgt nach der Liste der Attribute).

```
'priuser': univention.admin.property(  
    short_description = u'primärer Benutzer',  
    long_description = u'Der primäre Benutzer dieses Telefons',  
    syntax=SynVoIP_Address,  
    multivalue=0,  
    options=[],  
    required=1,  
    may_change=1,  
    identifies=0  
),
```

Das Attribut *users* zeigt die Verwendung und den Zweck von Optionen. mit dem Parameter `options=['extended']` wird festgelegt, dass dieses Attribut nur zur Verfügung steht, wenn mindestens eine der aufgelisteten Optionen aktiviert ist. Ist die Liste der angegebenen Optionen leer (vordefinierter Wert), wird das Attribut immer angezeigt.

```
'users': univention.admin.property(  
    short_description = u'weitere Benutzer',  
    long_description = u'Benutzer, die an diesem Telefon registriert  
sein dürfen',  
    syntax=SynVoIP_Address,  
    multivalue=1,  
    options=['extended'],  
    required=0,  
    may_change=1,  
    identifies=0  
)  
}
```

Die folgenden zwei Klassen sind die Syntaxdefinitionen, die für die Attribute *protocols*, *priuser* und *users* verwendet wurden. *SynVoIP_Protocols* basiert auf der vordefinierten Klasse *univention.admin.syntax.select*, die die Basisfunktionalität für Auswahllisten zur Verfügung stellt. Abgeleitete Klassen müssen, wie in der folgenden Klasse zu sehen ist, nur einen Namen und die Liste der Auswahlmöglichkeiten definieren.

Das Attribut *choices* ist eine Liste von Paaren, die aus einem eindeutigen Schlüssel für das Element und dem Text, der angezeigt werden soll, bestehen.

```
class SynVoIP_Protocols(univention.admin.syntax.select):
```

```
name=u'VoIP_Protocol'
choices=[ ( 'sip', u'SIP'), ( 'h323', u'H.323' ), ( 'skype', u'Skype' ) ]
```

Die andere Syntaxdefinition *SynVoIP_Address* basiert auf der Klasse *univention.admin.syntax.simple*, die eine Basisfunktionalität für Syntaxdefinitionen bietet, die mit regulären Ausdrücken arbeiten. Wie bei der anderen Definition auch muß Name vergeben werden. Zusätzlich sind noch die Attribute *min_length* und *max_length* anzugeben, die eine minimale und eine maximale Länge für valide Werte festlegen. Wird eines dieser Attribute auf 0 gesetzt, entspricht das einer nicht existenten Grenze in die jeweilige Richtung. Außer den genannten Attributen muß noch die Funktion *parse* definiert werden, die als Parameter den zu prüfenden Wert übergeben bekommt. Mittels des Python Moduls *re* wird in diesem Fall geprüft, ob der Wert dem Muster einer VoIP Adresse entspricht, z.B. sip:hans@mustermann.de.

```
class SynVoIP_Address(univention.admin.syntax.simple):
    name='VoIP_Address'
    min_length=4
    max_length=256
    _re = re.compile('((^(sip|h323|skype):)?([a-zA-Z])[a-zA-Z0-9._-]+@[a-zA-Z0-9._-]+)$')

    def parse(self, text):
        if self._re.match(text) != None:
            return text
        raise univention.admin.uexceptions.valueError,
            u'Keine gültige VoIP Adresse'
```

Anschließend wird die Abbildung von den Modul-Attributen auf die Attribute des zu erzeugenden LDAP-Objektes definiert. Dabei wird die Klasse *univention.admin.mapping.mapping* verwendet, die mit der Funktion *register* eine einfache Möglichkeit bietet, für die einzelnen Attribute Abbildungen zu registrieren. Das erste Argument der Funktion ist der Name des Modul-Attributs und das zweite der Name des LDAP-Attribute. Mit den folgenden zwei Argument der Funktion *register* können Abbildungsfunktionen für die Konvertierung von der Modul-Attributen zum LDAP-Attribute und vice versa angegeben werden.

```
mapping=univention.admin.mapping.mapping()
mapping.register('name', 'cn', None, univention.admin.mapping.ListToString)
mapping.register('active', 'testPhoneActive', boolToString, stringToBool)
mapping.register('protocol', 'testPhoneProtocol', None,
univention.admin.mapping.ListToString)
mapping.register('ip', 'testPhoneIP', None,
univention.admin.mapping.ListToString)
mapping.register('priuser', 'testPhonePrimaryUser', None,
univention.admin.mapping.ListToString)
```

```
mapping.register('users', 'testPhoneUsers')
```

Abschließend muss für das Modul noch eine Klasse *object* definiert werden, die den in Abschnitt 2.2 definierten Vorgaben entspricht. Für das IP-Telefon würde die Klasse folgendermaßen aussehen:

```
class object(univention.admin.handlers.simpleLdap):
    module=module

    def __init__(self, co, lo, position, dn='', superordinate=None,
                 arg=None):
        global mapping
        global property_descriptions
        self.co=co
        self.lo=lo
        self.dn=dn
        self.position=position
        self._exists=0
        self.mapping=mapping
        self.descriptions=property_descriptions
        univention.admin.handlers.simpleLdap.__init__(self, co, lo, position,
            dn, superordinate)

    def exists(self):
        return self._exists

    def open(self):
        univention.admin.handlers.simpleLdap.open(self)
        self.save()

    def _ldap_pre_create(self):
        self.dn='%s=%s,%s' % (mapping.mapName('name'), mapping.mapValue
            ('name', self.info['name']), self.position.getDn())

    def _ldap_addlist(self):
        return [ ('objectClass', [ 'top', 'testPhone' ] ) ]
```

Damit auch nach Objekten, die dieses Modul verwaltet, gesucht werden kann gibt es noch zwei Funktionen *lookup* und *identify*. Die hier vorgegebenen Funktionen sollten für einfache LDAP-Objekte, die durch eine einzelne *objectClass* identifiziert werden können, ausreichen. Für eigene LDAP-Objekte müsste die Objektklasse *testPhone* ersetzt werden.

```
def lookup(co, lo, filter_s, base='', superordinate=None, scope='sub',
          unique=0, required=0, timeout=-1, sizelimit=0):
    filter=univention.admin.filter.conjunction('&', [
        univention.admin.filter.expression('objectClass', 'testPhone'),
    ])

    if filter_s:
        filter_p=univention.admin.filter.parse(filter_s)
        univention.admin.filter.walk(filter_p,
            univention.admin.mapping.mapRewrite, arg=mapping)
        filter.expressions.append(filter_p)

    res=[]
    for dn in lo.searchDn(unicode(filter), base, scope, unique, required,
        timeout, sizelimit):
        res.append(object(co, lo, None, dn))
    return res
```

Diese Funktion prüft, ob das übergebene LDAP-Objekt zu der von diesem Modul verwalteten Menge gehört.

```
def identify(dn, attr, canonical=0):
    return 'testPhone' in attr.get('objectClass', [])
```

3.2 LDAP Schema

Bevor das entwickelte Modul für den Univention Admin genutzt werden kann, muß dem LDAP-Server erst noch die neue Objektklasse, in diesem Fall *testPhone*, zusammen mit ihren Attributen bekannt gemacht werden. Solche Objektdefinitionen werden bei LDAP über sogenannte Schemata definiert, die in Dateien beschrieben werden, die wie die folgende aussehen:

```
attributetype ( 1.3.6.1.4.1.10176.9999.1.1 NAME 'testPhoneActive'
    DESC 'state of the IP phone'
    EQUALITY caseIgnoreIA5Match
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 SINGLE-VALUE )

attributetype ( 1.3.6.1.4.1.10176.9999.1.2 NAME 'testPhoneProtocol'
    DESC 'The supported VoIP protocol'
    EQUALITY caseExactIA5Match
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 SINGLE-VALUE )

attributetype ( 1.3.6.1.4.1.10176.9999.1.3 NAME 'testPhoneIP'
```

```
DESC 'The IP address of the phone'
EQUALITY caseExactIA5Match
SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 SINGLE-VALUE )

attributetype ( 1.3.6.1.4.1.10176.9999.1.4 NAME 'testPhonePrimaryUser'
DESC 'The primary user of the phone'
EQUALITY caseIgnoreIA5Match
SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 SINGLE-VALUE )

attributetype ( 1.3.6.1.4.1.10176.9999.1.5 NAME 'testPhoneUsers'
DESC 'A list of other users allowed to use the phone'
EQUALITY caseIgnoreIA5Match
SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )

objectclass ( 1.3.6.1.4.1.10176.9999.2.1 NAME 'testPhone'
DESC 'IP Phone'
SUP top STRUCTURAL
MUST ( cn $ testPhoneActive $ testPhoneProtocol $ testPhoneIP $
testPhonePrimaryUser )
MAY ( testPhoneUsers )
)
```

Eine ausführliche Dokumentation zur Erstellung von LDAP Schema-Dateien ist auf der Webseite des OpenLDAP Projektes (<http://www.openldap.org/>) zu finden und ist nicht Schwerpunkt dieser Dokumentation.

4 Installation

Als letzter Schritt müssen das Python Modul und das LDAP Schema installiert werden. Im folgenden werden diese beiden Schritte dokumentiert.

4.1 Python Modul

Das Python Modul muss in ein bestimmtes Verzeichnis kopiert werden, damit der Univention Admin es findet. Zusätzlich muss die Python Quellcodedatei kompiliert werden, da der Univention Admin nur nach **.pyo* Dateien sucht. Dafür sind folgende Schritte zu tun:

- In das Verzeichnis */usr/lib/python2.4/site-packages/univention/admin/handlers/* wechseln. Die **2.4** Im Verzeichnisnamen hängt von der Python Version ab, die auf dem System genutzt wird. Sollten mehrere Python Versionen installiert sein, muß überprüft werden, in welchem der *site-packages* Verzeichnisse sich die Unterverzeichnisstruktur *univention/admin/handlers* befindet.

- In diesem Verzeichnis ist ein Unterverzeichnis anzulegen. Es sollte dem ersten Teil des Modulnamens entsprechen. Wenn der Name des Moduls beispielsweise **test/ip-phone** ist, dann sollte das Verzeichnis **test** heißen.
- Die Datei **ip-phone.py** in das neue Verzeichnis kopieren.
- Den Python Quellcode mit folgendem Befehl kompilieren:

```
python -OO -c "import py_compile; \
    py_compile.compile( 'test/ip-phone.py' )"
```

Um die Installation des neuen Moduls zu testen kann folgendes getan werden:

- Sicherstellen, dass keine Instanz von *univention-cli-server* mehr läuft
- Mit der Kommandozeilen Variante des Univention Admin alle vorhandenen Module auflisten lassen:

```
univention-admin modules list
```

In der ausgegebenen Liste sollte das neue Modul auftauchen.

4.2 LDAP Schema

Die Datei, die das LDAP-Schema enthält, kann im Prinzip in ein beliebiges Verzeichnis kopiert werden. Die Schema-Definitionen von Univention werden beispielsweise in dem Verzeichnis **/usr/share/univention-ldap/schema/** abgelegt. Damit der LDAP-Server dieses Schema findet muß es in der Konfigurationsdatei **/etc/ldap/slapd.conf** eingebunden werden. Da diese Datei unter der Kontrolle von Univention Baseconfig steht, sollte nicht direkt die Datei editiert werden, sondern ein Univention Baseconfig Template erstellt werden (Hierzu existiert eine Technische Dokumentation 'Univention Baseconfig Templates').

4.3 Debian-Paket

Die Informationen, wie aus einem Univention Admin Modul ein Debian-Paket erstellt werden kann, ist der Technischen Dokumentation 'Paketerstellung' zu entnehmen, welche in einer kommenden Version um die Behandlung von Baseconfig-Templates ergänzt wird.