

## **UCS@school**



### **Handbuch Import-Schnittstelle**

Version 4.4 v1  
Stand: 12. März 2019

Alle Rechte vorbehalten./ All rights reserved.  
(c) 2016-2019  
Univention GmbH  
Mary-Somerville-Straße 1  
28359 Bremen  
Deutschland  
feedback@univention.de

Jede aufgeführte Marke und jedes Warenzeichen steht im Eigentum ihrer jeweiligen eingetragenen Rechtsinhaber. Linux ist ein eingetragenes Warenzeichen von Linus Torvalds.

The mentioned brand names and registered trademarks are owned by the respective legal owners in each case. Linux is a registered trademark of Linus Torvalds.

## Inhaltsverzeichnis

1. Zielgruppe .....	5
2. Einführung .....	7
3. Ablauf des Importvorgangs .....	9
3.1. Datenformate .....	10
3.2. Zuordnung von Benutzern des Quellverzeichnis zu UCS@school-Benutzern .....	10
4. Konfiguration .....	13
4.1. Kommandozeilenparameter .....	14
4.2. JSON-Konfigurationsformat .....	15
4.2.1. Globale Konfiguration .....	15
4.2.2. Konfiguration des Benutzerimports .....	16
4.3. "default"-Schlüssel .....	20
4.4. Zuordnung von Eingabedaten zu Benutzerattributen .....	20
4.4.1. Sonderwerte .....	22
4.4.2. Eigene Erweiterungen hinzufügen .....	23
4.5. Formatierungsschema .....	23
4.6. Einmalige Benutzernamen und Email-Adressen .....	23
4.7. Benutzer löschen .....	24
5. Schulwechsel .....	27
6. Schuljahreswechsel .....	29
7. Schulübergreifende Benutzerkonten .....	31
7.1. Schulspezifisches <i>sambahome</i> .....	32
8. Erweiterung um neue Funktionalität .....	33
8.1. Die <code>ImportUser</code> Klasse .....	33
8.2. Hooks .....	34
8.2.1. <i>Format-Hooks</i> .....	36
8.3. <i>Subclassing</i> .....	37
8.3.1. <i>Abstract Factory</i> .....	38
8.3.2. Überschreiben einer Methode .....	38
8.3.3. Ersetzen durch eigene Klasse .....	39
8.4. <i>Prüfung der Konfiguration</i> .....	39
Literaturverzeichnis .....	41



# Kapitel 1. Zielgruppe

Dieses Handbuch richtet sich an Administratoren und Programmierer. Die Import Software ist im Auslieferungszustand stark konfigurierbar, kann aber zusätzlich programmatisch erweitert werden. In diesem Handbuch werden der Ablauf eines Importvorganges, Konfigurationsoptionen und Programmierschnittstellen beschrieben.



## Kapitel 2. Einführung

UCS@school bringt für viele regelmäßig wiederkehrende Verwaltungsaufgaben Werkzeuge und Schnittstellen mit. Die Übernahme von Benutzerdaten aus der Schulverwaltung ist eine dieser wiederkehrenden Aufgaben, die über die neue Importschnittstelle für Benutzer automatisiert erledigt werden kann.

Der UCS@school Import ermöglicht es Benutzerdaten aus einer Datei auszulesen, die Daten zu normieren, automatisch eindeutige Benutzernamen und E-Mail-Adressen zu generieren und notwendige Änderungen (Hinzufügen/Modifizieren/Löschen) automatisch zu erkennen. Er wurde so konzipiert, dass die Konten einer UCS Domäne automatisch mit dem Datenbestand eines vorhandenen Benutzerverzeichnisses abgeglichen werden können.

Die Importschnittstelle ist darauf ausgelegt, mit möglichst geringem Aufwand an die unterschiedlichen Gegebenheiten in Schulen angepasst zu werden. So ist die Basis der Importschnittstelle bereits vorbereitet, um unterschiedliche Dateiformate einlesen zu können. UCS@school bringt einen Importfilter für CSV-Dateien mit, der für unterschiedlichste CSV-Formate konfiguriert werden kann.

Über eigene Python-Plugins kann die Schnittstelle erheblich erweitert werden. Dies umfasst sowohl die Unterstützung für zusätzliche Dateiformate als auch die Implementierung von zusätzlichen Automatismen, die während des Imports greifen.

In den nachfolgenden Kapiteln werden der Ablauf eines Imports, die unterschiedlichen Konfigurationsmöglichkeiten sowie die Erweiterungsmöglichkeiten der Schnittstelle um neue Funktionalitäten beschrieben.





## Kapitel 3. Ablauf des Importvorgangs

3.1. Datenformate .....	10
3.2. Zuordnung von Benutzern des Quellverzeichnisses zu UCS@school-Benutzern .....	10

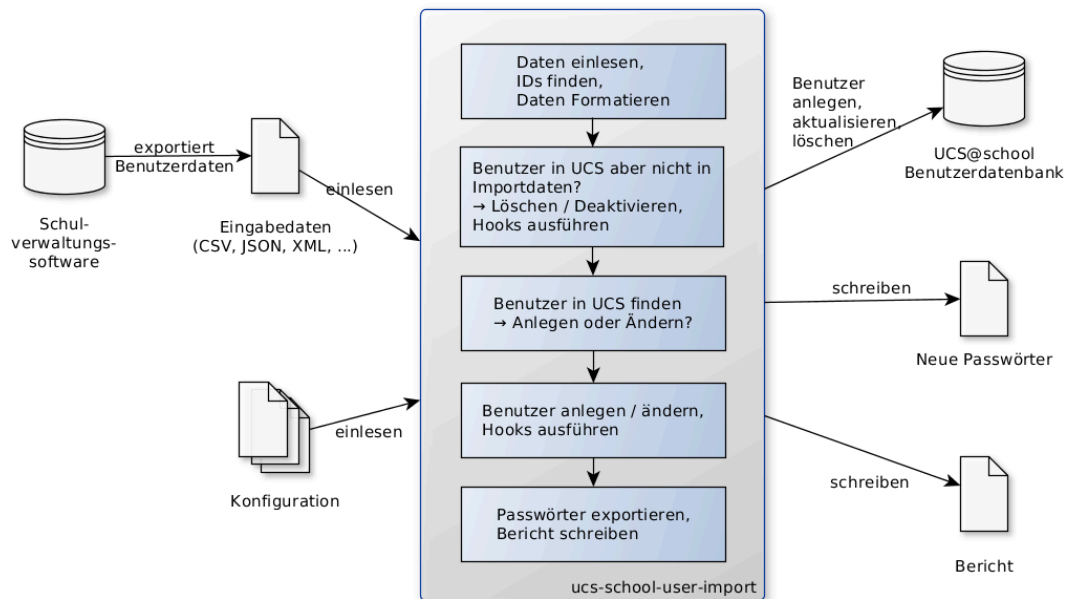
Die Importschnittstelle wurde als ein kommandozeilenbasiertes Tool umgesetzt, welches darauf ausgelegt ist, einen automatischen, nicht-interaktiven Import durchzuführen. Um dies zu erreichen, müssen bestimmte Voraussetzungen erfüllt werden, die in diesem Abschnitt anhand des Importablaufes erläutert werden.

Das Tool wurde in der Skriptsprache Python implementiert und ist in der Lage, zusätzliche/kundenspezifische Python-Dateien zu laden, die ein abweichendes Verhalten oder zusätzliche Funktionalitäten der Importschnittstelle hinzufügen.

Der Ablauf eines automatischen Imports wird über die nachfolgenden Schritte skizziert:


1. Die zu übernehmenden Benutzerdaten müssen automatisch oder manuell aus dem Quellverzeichnis exportiert und als Datei gespeichert werden.
2. Für den Import muss vorab einmalig eine Konfigurationsdatei erstellt werden, die bei jedem weiteren Import wiederverwendet werden kann. Die Konfigurationsdatei ermöglicht es dem Importtool `ucs-school-user-import`, die exportierten Daten einzulesen und die Eingabedatensätze (Benutzerdaten) konkreten UCS@school-Benutzern im LDAP-Verzeichnisdienst zuzuordnen.
3. Die Importschnittstelle unterstützt zwei unterschiedliche Modi, die festlegen, ob die Eingabedatensätze als neuer Soll-Zustand oder als inkrementelles Update zum Ist-Zustand interpretiert werden sollen.
  - Im ersten Fall werden die Eingabedatensätze als neuer Soll-Zustand verwendet. Dazu wird ein automatischer Abgleich zwischen den UCS@school-Benutzern im LDAP-Verzeichnisdienst und den übergebenen Eingabedatensätzen durchgeführt, um zu ermitteln, welche UCS@school-Benutzer im LDAP-Verzeichnisdienst angelegt, modifiziert oder gelöscht werden müssen, um den neuen Soll-Zustand zu erreichen.
  - Im zweiten Fall findet kein automatischer Abgleich statt. Stattdessen muss bei jedem Eingabedatensatz speziell vermerkt werden, ob der dazu passenden UCS@school-Benutzer im LDAP-Verzeichnis angelegt, modifiziert oder gelöscht werden. Dieser teilautomatische Ansatz erfordert erheblich mehr Prozesslogik bei der Bereitstellung der Eingabedatensätze, da die entsprechenden Teile in der Importschnittstelle deaktiviert sind.
4. Je nach Konfiguration werden die Werte des Eingabedatensatzes nach dem Einlesen automatisch geprüft, modifiziert und/oder erweitert. Darunter fällt z.B. auch die automatische Zuweisung eines eindeutigen Benutzernamens oder die Generierung einer E-Mail-Adresse.
5. Während des vollständigen Einlesens und Verarbeitens der Eingabedatensätze werden noch keine Änderungen am LDAP-Verzeichnis vorgenommen. Sind alle Eingabedatensätze verarbeitet, werden im Anschluss die notwendigen Änderungen am LDAP-Verzeichnis durchgeführt: UCS@school-Benutzer werden gelöscht, verändert oder angelegt.
6. Neuen UCS@school-Benutzern werden während des Imports automatisch sichere, zufällige Passwörter zugeteilt. Da Passwörter im LDAP-Verzeichnis grundsätzlich nicht im Klartext vorliegen und somit später nicht mehr ausgelesen werden können, werden die Passwörter für neue Benutzer, sofern explizit konfiguriert, in einer speziellen CSV-Datei abgelegt.
7. Nach den erfolgten Änderungen am LDAP-Verzeichnis wird abschließend ein Bericht in Form einer CSV-Datei erzeugt, mit dessen Hilfe sich schnell die Eingabedaten und die durch sie verursachten Änderungen sowie eventuell aufgetretene Probleme nachvollziehen lassen.

Abbildung 3.1. Ablauf eines Benutzerimports




Für die Analyse von Problemen wird während des gesamten Imports ein Protokoll über alle technischen Vorgänge in Logdateien mit unterschiedlichen Detailtiefen geführt. Wenn nicht anders konfiguriert, sind das die Dateien `/var/log/univention/ucs-school-import.log` und `/var/log/univention/ucs-school-import.info`.

### 3.1. Datenformate

Feedback 

Die Importsoftware kann gegenwärtig Daten nur aus *CSV-Dateien*<sup>1</sup> einlesen. Wie eine Unterstützung für weitere Dateiformate (z.B. JSON, XML etc.) hinzugefügt werden kann, kann dem Kapitel 8 entnommen werden.

### 3.2. Zuordnung von Benutzern des Quellverzeichnisses zu UCS@school-Benutzern

Feedback 

Das Tool `ucs-school-user-import` unterstützt den Import von Benutzerdaten aus mehreren Quellverzeichnissen. Um jeden UCS@school-Benutzer einem Benutzer in einer Quelldatenbank eindeutig zuzuordnen zu können, werden am UCS@school-Benutzerobjekt zwei zusätzliche Attribute gespeichert: `source_uid` und `record_uid`.

Die `source_uid` ist ein eindeutiger Bezeichner für die Quelldatenbank von der ein Benutzer importiert wurde. Der Bezeichner kann frei gewählt werden und muss für jede Quelldatenbank eindeutig sein. Er ist während des Imports auf der Kommandozeile bzw. in der Konfigurationsdatei für jede Quelldatenbank explizit mit anzugeben.

Die `record_uid` ist ein eindeutiger Bezeichner für den Benutzer in der Quelldatenbank. Als Bezeichner kann z.B. auf vorhandene Attribute innerhalb der Quelldatenbank, wie z.B. eine Schüler- oder Mitarbeiternummer, zurückgegriffen werden. Sollte kein eindeutig identifizierendes Attribut in der Quelldatenbank vorhanden sein, kann auch durch die Konkatenation von mehreren Attributen der Quelldatenbank ein eindeutiger Bezeichner generiert werden.

<sup>1</sup> [https://de.wikipedia.org/wiki/CSV\\_\(Dateiformat\)](https://de.wikipedia.org/wiki/CSV_(Dateiformat))

Durch die Kombination dieser beiden Bezeichner kann ein UCS@school-Benutzer genau einem Benutzer in einem bestimmten Quellverzeichnis zugeordnet werden.

## Achtung

*source\_uid* und *record\_uid* müssen eindeutig und unveränderlich sein, sonst werden UCS@school-Benutzer beim Abgleich mit den Eingabedaten nicht gefunden und ggf. gelöscht bzw. es werden die falschen UCS@school-Benutzerobjekte modifiziert.

Mit Hilfe der beiden Bezeichner *source\_uid* und *record\_uid* wird versucht, jeden Eingabedatensatz genau einem UCS@school-Benutzer zuzuordnen:

- Wurde kein UCS@school-Benutzer mit passenden Bezeichnern im LDAP-Verzeichnis gefunden, wird ein neuer UCS@school-Benutzer auf Basis des Eingabedatensatzes erstellt.
- Existiert ein passender UCS@school-Benutzer bereits, wird er von `ucs-school-user-import` modifiziert. Die Importsoftware gleicht die Eingabedaten mit dem LDAP-Verzeichnisdienst ab und passt den UCS@school-Benutzer entsprechend dem Eingabedatensatz an.
- Während des Abgleichs wird auch geprüft, ob im LDAP-Verzeichnis UCS@school-Benutzer der betreffenden Quelldatenbank vorhanden sind, die in den Eingabedatensätzen nicht mehr vorhanden sind. Die betroffenen UCS@school-Benutzer werden dann automatisch gelöscht.

## Achtung

Wird vom früheren Import Skript zum Neuen migriert muss beachtet werden, dass je nachdem welche Version zuvor benutzt wurde, an den Benutzerobjekten entweder *keine source\_uid* gespeichert wurde, oder der Wert `LegacyDB`.

Beispiele:

- Die Schulen eines Schulträgers verwenden voneinander unabhängige Verwaltungssoftware. Die Software exportiert für jede Schule separate CSV-Dateien für den Import. Es wird je eine Datei für Schüler, Lehrer und Mitarbeiter erzeugt. Für den Import der CSV-Dateien wird pro Schule und Benutzerrolle eine separate Konfiguration mit individueller *source\_uid* benötigt. Sind die Konfigurationen hinreichend ähnlich, können die gleichen Konfigurationsdateien verwendet werden und die sie unterscheidenden Optionen an der Kommandozeile gesetzt werden. Sollten sich nur *source\_uid* und Benutzerrolle unterscheiden, so würde der Import mit den entsprechenden Optionen z.B. so aufgerufen:

```
# /usr/share/ucs-school-import/scripts/ucs-school-user-import \  
--conffile <gemeinsame Konfigurationsdatei> \  
--source_uid <Schulname>-<Benutzerrolle> \  
--user_role <Benutzerrolle> \  
--infile <CSV-Datei>
```

Durch die Verwendung von *Schulname-Benutzerrolle* (z.B. `GSMitte-student`) als *source\_uid* wird ein eindeutiger Bezeichner pro Schule und Benutzerrolle sicher gestellt.

Ab UCS@school Version 4.2 v4 wird nur eine *source\_uid* pro Schule benötigt (`-<Benutzerrolle>` kann weggelassen werden), sofern mit `--user_role` die Benutzerrolle angegeben wird.

In diesem Szenario ist die Verwendung schulübergreifender Benutzerkonten *nicht* möglich, weil dafür eine gemeinsame *source\_uid* der beteiligten Schulen benötigt wird.

- Ein Schulträger verwendet eine Software für die Verwaltung aller seiner Schulen. Die Software exportiert für jede Benutzerrolle eine CSV-Datei. In diesen Dateien sind alle Benutzer aller Schulen (von der jeweili-

## Zuordnung von Benutzern des Quellverzeichnisses zu UCS@school-Benutzern

ligen Rolle) enthalten. Für den Import der CSV-Dateien wird nur pro Benutzerrolle eine separate Konfiguration mit individueller *source\_uid* benötigt, bzw. die gleiche Konfigurationsdatei und an der Kommandozeile wird gesetzt: `--source_uid <Benutzerrolle>`.

Ab UCS@school Version 4.2 v4 wird keine separate *source\_uid* mehr pro Benutzerrolle benötigt. Es reicht dann eine Konfigurationsdatei mit einer darin eingespeicherten *source\_uid* für alle Importvorgänge, sofern mit `--user_role` die Benutzerrolle angegeben wird.

In diesem Szenario ist die Verwendung schulübergreifender Benutzerkonten möglich, weil Benutzer mit der gleichen Rolle an allen Schulen die gleiche *source\_uid* haben.

- Ein Schulträger verwendet eine Software für die Verwaltung aller seiner Schulen. Die Software exportiert alle Benutzer in eine CSV-Datei. In dieser Datei sind Benutzer aller Rollen und aller Schulen enthalten. In der CSV-Datei gibt es eine Spalte in der steht welche Rolle der jeweilige Benutzer hat. Für den Import der CSV-Dateien wird nur eine Konfigurationsdatei mit einer darin eingespeicherten *source\_uid* benötigt. Um die Benutzerrolle auszulesen, wird der entsprechenden Spalte der Sonderwert `__role` zugeordnet (siehe als Beispiel die Konfigurationsdatei `/usr/share/ucs-school-import/configs/ucs-school-testuser-import.json`).

In diesem Szenario ist die Verwendung schulübergreifender Benutzerkonten möglich, weil alle Benutzer die gleiche *source\_uid* haben.

# Kapitel 4. Konfiguration

4.1. Kommandozeilenparameter .....	14
4.2. JSON-Konfigurationsformat .....	15
4.2.1. Globale Konfiguration .....	15
4.2.2. Konfiguration des Benutzerimports .....	16
4.3. "default"-Schlüssel .....	20
4.4. Zuordnung von Eingabedaten zu Benutzerattributen .....	20
4.4.1. Sonderwerte .....	22
4.4.2. Eigene Erweiterungen hinzufügen .....	23
4.5. Formatierungsschema .....	23
4.6. Einmalige Benutzernamen und Email-Adressen .....	23
4.7. Benutzer löschen .....	24

Die Konfiguration des Imports wird über Dateien im *JSON-Format*<sup>1</sup> und Kommandozeilenparameter gesteuert. Alle Kommandozeilenparameter können als Variablen in den Konfigurationsdateien verwendet werden, so dass ein Ausführen der Importsoftware ohne Kommandozeilenparameter möglich ist. Es ist aber auch möglich, über Kommandozeilenparameter sämtliche Variablen, die in Konfigurationsdateien stehen, zu überschreiben.

Beim Start von `ucs-school-user-import` werden nacheinander mehrere Konfigurationsdateien eingelesen. Jede Datei fügt der Konfiguration neue Konfigurationsvariablen hinzu oder überschreibt bereits existierende Konfigurationsvariablen von vorher eingelesenen Konfigurationsdateien.

Die Konfiguration des Imports wird in der folgenden Reihenfolge (mit aufsteigender Priorität) über folgende Konfigurationsdateien und Kommandozeilenparameter eingelesen:

1. `/usr/share/ucs-school-import/configs/global_defaults.json` (Diese Datei sollte nicht manuell editiert werden!)
2. `/var/lib/ucs-school-import/configs/global.json` (Diese Datei kann manuell angepasst werden – siehe unten)
3. `/usr/share/ucs-school-import/configs/user_import_defaults.json` (Diese Datei sollte nicht manuell editiert werden!)
4. `/var/lib/ucs-school-import/configs/user_import.json` (Diese Datei kann manuell angepasst werden – siehe unten)
5. Eine JSON-Datei, die mit dem Parameter `-c` an der Kommandozeile angegeben wurde.
6. Variablen, die über den Kommandozeilenparameter `--set` gesetzt wurden.

Die Konfigurationsdateien unterhalb von `/usr/share/ucs-school-import/configs/` sollten nicht editiert werden. Sie sind Teil der UCS@school-Installation und werden u.U. von Updates überschrieben.

Die Dateien unter `/var/lib/ucs-school-import/configs/` werden automatisch bei der Installation angelegt und sind eigens dafür vorgesehen, eigene Einstellungen bzw. Konfigurationen vorzuhalten. Die Dateien bleiben bei Updates unangetastet.

Folgendes Verfahren wird empfohlen:


1. Um Variablen zu überschreiben, die in `/usr/share/ucs-school-import/configs/global_defaults.json` gesetzt werden, können eigene Werte in die Datei `/var/lib/ucs-school-import/configs/global.json` eingetragen werden.

<sup>1</sup> [https://de.wikipedia.org/wiki/JavaScript\\_Object\\_Notation](https://de.wikipedia.org/wiki/JavaScript_Object_Notation)

2. Um Variablen zu überschreiben, die in `/usr/share/ucs-school-import/configs/user_import_defaults.json` gesetzt werden, können eigene Werte in die Datei `/var/lib/ucs-school-import/configs/user_import.json` eingetragen werden. Falls regelmäßig aus mehreren Quellverzeichnissen Benutzer importiert werden, sollten in dieser Datei auch Variablen gesetzt werden, die für alle Datenquellen gleichermaßen gelten.
3. Pro Quellverzeichnis sollte eine Konfigurationsdatei unterhalb von `/var/lib/ucs-school-import/configs/` abgelegt werden, welche schließlich an der Kommandozeile mit `-c` angegeben wird. Diese Datei enthält Konfigurationseinstellungen, die auf die Spezifika der jeweiligen Schule bzw. Schulverwaltungssoftware einzugehen, und die jeweilige `source_uid` passend zum Quellverzeichnis.

Die resultierende Konfiguration, die sich aus eingelesenen Konfigurationsdateien sowie verwendete Kommandozeilenparameter zusammenstellt, wird am Anfang jedes Importlaufes angezeigt und in alle Protokolldateien geschrieben. Um eine Konfiguration zu testen, kann ein Probelauf mit `--dry-run` gestartet (und jederzeit gefahrlos abgebrochen) werden. Der Parameter simuliert einen Import, ohne dass Änderungen an UCS@school-Benutzern vorgenommen werden.

## 4.1. Kommandozeilenparameter

 Feedback 

```
# /usr/share/ucs-school-import/scripts/ucs-school-user-import --help
usage: ucs-school-user-import [-h] [-c CONFFILE] [-i INFILE]
                             [-l LOGFILE] [-m] [-n] [-s SCHOOL]
                             [--set [KEY=VALUE [KEY=VALUE ...]]]
                             [--source_uid source_uid] [-u USER_ROLE] [-v]

optional arguments:
  -h, --help                show this help message and exit
  -c CONFFILE, --conffile CONFFILE
                            Configuration file to use (see
                            /usr/share/doc/ucs-school-import for an
                            explanation on configuration file stacking).
  -i INFILE, --infile INFILE
                            CSV file with users to import (shortcut for
                            --set input:filename=...).
  -l LOGFILE, --logfile LOGFILE
                            Write to additional logfile (shortcut for
                            --set logfile=...).
  --set [KEY=VALUE [KEY=VALUE ...]]
                            Overwrite setting(s) from the configuration file. Use ':' in
                            key to set nested values (e.g. 'scheme:email=...').
  -m, --no-delete           Only add/modify given user objects. User objects not
                            mentioned within input files are not deleted/deactivated
                            (shortcut for --set no_delete=...) [default: False].
  -n, --dry-run             Dry run - don't actually commit changes to LDAP (shortcut
                            for --set dry_run=...) [default: False].
  --source_uid source_uid   The ID of the source database (shortcut for
                            --set source_uid=...) [mandatory either here or in the
                            configuration file].
  -s SCHOOL, --school SCHOOL
                            Name of school. Set only, if the source data does not
                            contain the name of the school and all users are from one
                            school (shortcut for --set school=...) [default: None].
  -u USER_ROLE, --user_role USER_ROLE
```

```

    Set this, if the source data contains users with only one
    role <none|student|staff|teacher|teacher_and_staff>
    (shortcut for --set user_role=...) [default: None].
-v, --verbose
    Enable debugging output on the console [default: False].
  
```

Nahezu alle Kommandozeilenparameter können auch in den Konfigurationsdateien angegeben werden. Um Variablen aus Konfigurationsdateien an der Kommandozeile zu setzen, kann `--set` verwendet werden. Verschachtelte Konfigurationsvariablen können mit dem Doppelpunkt angegeben werden.

Um z.B. "2" als die Anzahl der Kopfzeilen in einer CSV-Datei anzugeben, kann entweder in einer Konfigurationsdatei stehen:

```

{
  "csv": {
    "header_lines": 2
  }
}
  
```


oder es kann an der Kommandozeile der folgende Parameter verwendet werden:

```
--set csv:header_lines=2
```

Alle Zuweisungen für Variablen können leerzeichengetrennt hinter dem Kommandozeilenparameter `--set` aufgelistet werden. Dabei ist zu beachten, dass nur ein `--set`-Parameter an der Kommandozeile ausgewertet wird.

```
--set csv:header_lines=2 maildomain=univention.de no_delete=True
```

## 4.2. JSON-Konfigurationsformat

 Feedback 

Das JSON-Format erlaubt Daten in verschachtelten Strukturen zu speichern, und ist sowohl von Computern als auch Menschen zuverlässig zu lesen und zu schreiben. Nach dem Editieren einer JSON-Datei kann ihre syntaktische Korrektheit mit Hilfe einer Webseite zur JSON Validierung oder eines Kommandozeilenprogramms überprüft werden:

```
python -m json.tool < my_config.json
```


Im Folgenden werden alle Konfigurationsschlüssel und ihre möglichen Werte und Typen beschrieben.

Schlüssel sind immer als Zeichenketten (`string`) zu behandeln und müssen in doppelten Anführungszeichen stehen. Als Datentypen werden folgende Typen unterstützt: Wahrheitswerte (`boolean`: `true` / `false`), Ganzzahlen (`int`), Gleitkommazahlen (`float`), Listen (`list`: Werte die in `[` und `]` eingeschlossen sind) und Objekte (`object`: neue Verschachtelungsebene die in `{` und `}` eingeschlossen wird). Die Verschachtelungstiefe wird in den Schlüsseln wie oben beschrieben mit Doppelpunkten angezeigt.

### Anmerkung

Eine Kurzreferenz aller Konfigurationsschlüssel findet sich auf dem Domänencontroller Master im Verzeichnis `/usr/share/doc/ucs-school-import/`.

### 4.2.1. Globale Konfiguration

 Feedback 

#### Achtung


In den folgenden Tabellen kommt es layoutbedingt zu Leerzeichen und Umbrüchen in Variablen- und Schlüsselnamen. Diese müssen in den Konfigurationsdateien bzw. an der Kommandozeile entfernt werden. *Keines* der Schlüsselwörter enthält ein Leerzeichen oder einen Schrägstrich.



**Tabelle 4.1. Globale Konfigurationsoptionen**

Schlüssel / Kommandozeilenparameter	Beschreibung
dry_run -n --dry-run	Ob ein Testlauf gestartet werden soll. Es werden keine Änderungen vorgenommen. Standard (boolean): false
logfile -l --logfile	Datei in die das ausführliche Protokoll geschrieben werden soll. Es wird außerdem eine Datei die auf <code>.info</code> endet, mit weniger technischen Details, angelegt. Standard (string): <code>"/var/log/univention/ucs-school-import.log"</code>
verbose -v --verbose	Ob ein ausführliches Protokoll auf die Kommandozeile geschrieben werden soll. Standard (boolean): true

## 4.2.2. Konfiguration des Benutzerimports

Feedback 

**Tabelle 4.2. Konfigurationsoptionen für den Benutzerimport**

Schlüssel	Beschreibung
classes	Die Methoden der Klasse <code>DefaultUserImportFactory</code> können überschrieben werden ohne die Klasse selbst zu ändern. Die Namen der überschriebenen Methoden ohne vorangestelltes <code>make_</code> sind die Schlüssel, der volle Python-Pfad der Wert. Standardmäßig ist das Objekt leer und die Klasse <code>DefaultUserImportFactory</code> wird unverändert verwendet. Ein Beispiel findet sich in Abschnitt 8.3.2. Standard (object): <code>{}</code>
factory	Voller Python-Pfad zu einer Klasse die von <code>DefaultUserImportFactory</code> abgeleitet ist. Wenn gesetzt, wird sie an ihrer Stelle verwendet (siehe Abschnitt 8.3.3). Standard (string): <code>"ucsschool.importer.default_user_import_factory.DefaultUserImportFactory"</code>
input	Objekt, welches Informationen über die Eingabedaten enthält. Standard (object)
input:type	Datenformat der angegebenen Eingabedatei. UCS@school unterstützt derzeit nur "csv" als Datenformat. Standard (string): <code>"csv"</code>
input:filename -i --infile	Einzulesende Datei. Standard (string): <code>"/var/lib/ucs-school-import/new-format-userimport.csv"</code>
activate_new \_users	Objekt, welches Konfigurationmöglichkeiten zur Benutzeraktivierung enthält. Standardmäßig ist im Objekt nur der Schlüssel <code>default</code> gesetzt. Weitere Schlüssel ( <code>student</code> , <code>staff</code> , <code>teacher</code> , <code>teacher_and_staff</code> ) sind möglich (siehe Abschnitt 4.3). Standard (object): <code>{"default": ..}</code>
activate_new \_users: \_default	Diese Variable definiert, ob ein neuer Benutzer automatisch aktiviert werden soll. Ist <code>false</code> eingestellt, wird das Benutzerkonto beim Anlegen automatisch deaktiviert. Standard (boolean): true



Schlüssel	Beschreibung
csv	Dieses Objekt enthält Informationen darüber, wie CSV-Eingabedaten interpretiert werden sollen.
	Standard (object): {"header_lines": .., "incell-delimiter": .., "mapping": ..}
csv:delimiter	Diese Variable definiert das Trennzeichen zwischen zwei Spalten. Als Wert wird üblicherweise ein Komma, Semikolon oder Tabulator verwendet. Die Importschnittstelle versucht das Trennzeichen automatisch zu erkennen, wenn diese Variable nicht gesetzt ist.
	Standard (string): nicht gesetzt
csv:header_lines	Diese Variable definiert, wie viele Zeilen der Eingabedaten übersprungen werden sollen, bevor die eigentlichen Benutzerdaten anfangen. Wird der Wert "1" (Kopfdatensatz) verwendet, wird der Inhalt der ersten Zeile als Namen der einzelnen Spalten interpretiert. Die dort verwendeten Namen können dann in csv:mapping als Schlüssel verwendet werden.
	Standard (int): 1
csv:incell-delimiter	Dieses Objekt enthält Informationen darüber, welches Zeichen <i>innerhalb</i> einer Zelle zwei Daten trennt und kann z.B. bei der Angabe von mehreren Telefonnummern verwendet werden. Es kann ein Standard (default) und pro Univention Directory Manager-Attribut eine Konfiguration (mit dem Namen des Schlüssels in csv:mapping) definiert werden.
	Standard (object): {"default": ..}
csv:incell-delimiter:default	Standard-Trennzeichen <i>innerhalb</i> einer Zelle, wenn kein spezieller Schlüssel für die Spalte existiert.
	Standard (string): ", "
csv:mapping	Enthält Informationen über die Zuordnung von CSV-Spalten zum Benutzerobjekt. Ist standardmäßig leer. Siehe Abschnitt 4.4.
	Standard (object): {}
deletion_grace_period	Dieses Objekt enthält Einstellungen zum Löschen von Benutzern.
	Standard (object): {"deactivation": .., "deletion": ..}
deletion_grace_period:deactivation	Definiert in wie vielen Tagen ein Benutzer, der nicht mehr in den Eingabedaten enthalten ist, deaktiviert (nicht gelöscht) werden soll. Wenn 0 gesetzt ist, wird das betroffene UCS@school-Benutzerkonto sofort deaktiviert. Wenn deletion_grace_period:deletion auf einen kleineren oder den gleichen Wert gesetzt ist, wird das Benutzerobjekt gelöscht statt deaktiviert.
	Standard (int): 0
deletion_grace_period:deletion	Definiert die Anzahl der Tage, die nach dem Import vergehen sollen, bevor der Benutzer gelöscht aus dem Verzeichnisdienst wird. Bei einem Wert von 0 wird der Benutzer sofort gelöscht. Bei größeren Zahlen wird das geplante Löschedatum im Univention Directory Manager-Attribut ucsschoolPurgeTimestamp gesetzt. Ein Cron Job löscht automatisch Benutzer, deren geplanter Löschezitpunkt erreicht ist.
	Standard (int): 0
normalize:firstname	Definiert, ob der in der CSV-Datei angegebene Wert für den Vornamen (i.d.R. UTF-8-kodiert) auf die Kodierung ASCII normalisiert wird. Umlaute und Sonderzeichen werden dabei ersetzt ("ä" wird zu "ae") oder entfernt.
	Standard (bool): false

Schlüssel	Beschreibung
normalize: lastname	<p>Definiert, ob der in der CSV-Datei angegebene Wert für den Nachnamen (i.d.R. UTF-8-kodiert) auf die Kodierung ASCII normalisiert wird. Umlaute und Sonderzeichen werden dabei ersetzt ("ä" wird zu "ae") oder entfernt.</p> <p style="text-align: right;">Standard (bool): false</p>
scheme	<p>Enthält Informationen über die Erzeugung von Werten aus anderen Werten und Regeln.</p> <p>Es können Ersetzungen wie in den <i>UCS-Benutzervorlagen</i><sup>2</sup> verwendet werden sowie alle Schlüssel aus <code>csv:mapping</code>. Neben Formatvorlagen für <code>email</code>, <code>record_uid</code> und <code>username</code> können Konfigurationen für beliebige Univention Directory Manager-Attribute hinterlegt werden. [ALWAYS COUNTER] und [COUNTER2] werden <i>nur</i> in <code>scheme:email</code> und <code>scheme:username</code> verarbeitet.</p> <p style="text-align: right;">Standard (object): {"email": .., "record_uid": .., "username": {...}}</p>
scheme:email	<p>Schema, aus dem die Email-Adresse erzeugt werden soll. Zusätzlich zu den in Abschnitt 4.5 beschriebenen Ersetzungen kommen noch zwei weitere hinzu: [ALWAYS COUNTER] und [COUNTER2] (siehe Abschnitt 4.6). Für die Verwendung des <code>email</code>-Schemas ist es erforderlich, dass <code>maildomain</code> oder die UCR-Variable <code>mail/hosteddomains</code> gesetzt ist. Anderenfalls wird keine Mailadresse generiert.</p> <p style="text-align: right;">Standard (string): "&lt;firstname&gt;[0].&lt;lastname&gt;@&lt;maildomain&gt;"</p>
scheme: record_uid	<p>Schema aus dem die eindeutige ID des Benutzers in der Quelldatenbank (Schulverwaltungssoftware) erzeugt werden soll.</p> <p style="text-align: right;">Standard (string): "&lt;email&gt;"</p>
scheme: username	<p>Enthält Informationen über die Erzeugung von Benutzernamen. Standardmäßig enthält das Objekt nur den Schlüssel <code>default</code>. Weitere Schlüssel (<code>student</code>, <code>staff</code>, <code>teacher</code>, <code>teacher_and_staff</code>) sind möglich (siehe Abschnitt 4.3). Zusätzlich zu den in Abschnitt 4.5 beschriebenen Ersetzungen kommen noch zwei weitere hinzu: [ALWAYS COUNTER] und [COUNTER2] (siehe Abschnitt 4.6).</p> <p style="text-align: right;">Standard (object): {"default": ..}</p>
scheme: username: default	<p>Schema aus dem der Benutzername erzeugt werden soll, wenn kein Schema speziell für diesen Benutzertyp (<code>scheme:username:teacher</code> etc) existiert.</p> <p style="text-align: right;">Standard (string): "&lt;:umlauts&gt;&lt;firstname&gt;[0].&lt;lastname&gt;[COUNTER2]"</p>
scheme:<udm attribute name>	<p>Univention Directory Manager-Attribute, die aus einem Schema erzeugt werden sollen. Der Schlüssel braucht nicht in <code>csv:mapping</code> vorzukommen.</p> <p style="text-align: right;">Standard (string): nicht gesetzt</p>
maildomain	<p>Der Wert dieser Variable wird beim Formatieren mit einem Schema in die Variable <code>&lt;maildomain&gt;</code> eingesetzt. Wenn nicht gesetzt, wird versucht <code>&lt;maildomain&gt;</code> durch Daten aus dem System zu füllen. Dafür wird die UCR-Variable <code>mail/hosteddomains</code> herangezogen. Sind <code>maildomain</code> und <code>mail/hosteddomains</code> nicht gesetzt, werden keine Mailadressen automatisch generiert.</p> <p style="text-align: right;">Standard (string): nicht gesetzt</p>
mandatory_ attributes	<p>Liste von Univention Directory Manager Attributen, die an jedem Benutzer gesetzt sein müssen.</p>

<sup>2</sup> <http://docs.software-univention.de/handbuch-4.4.html#users:templates>


Schlüssel	Beschreibung
	Standard (list): ["firstname", "lastname", "name", "record_uid", "school", "source_uid"]
no_delete -m --no-delete	Wenn auf true gesetzt, werden keine Benutzer gelöscht, oder nur solche für die es in den Eingabedaten <i>explizit</i> vermerkt ist. Dies kann genutzt werden, um eine Änderung an UCS@school-Benutzern vorzunehmen, ohne einen vollständigen Soll-Zustand zu übergeben oder um neue Benutzer hinzuzufügen..
	Standard (bool): false
output	Dieses Objekt enthält Informationen über zu produzierende Dokumente.
	Standard (object): {"import_summary": ..}
output:new_ user_pass- words	Diese Variable definiert den Pfad zur CSV-Datei, in die Passwörter neuer Benutzer geschrieben werden. Auf den Dateinamen wird die Python-Funktion <code>datetime.datetime.strptime()</code> angewandt. Wenn ein <i>Python-Format-String</i> <sup>3</sup> in ihm vorkommt, wird dieser umgewandelt (siehe Beispiel <code>output:user_import_summary</code> ).
	Standard (string): nicht gesetzt
output:user_ import_summa- ry	Diese Variable definiert den Pfad zur CSV-Datei, in die eine Zusammenfassung des Import-Vorganges geschrieben wird. Auf den Dateinamen wird, wie bei <code>output:new_user_passwords</code> , die Python-Funktion <code>datetime.datetime.strptime()</code> angewandt.
	Standard (string): "/var/lib/ucs-school-import/summary/%Y/%m/user_import_summary_%Y-%m-%d_%H:%M:%S.csv"
password_ length	Definiert die Länge des zufälligen Passwortes, das für neue Benutzer erzeugt wird.
	Standard (int): 15
school -s --school	Schulkürzel/OU-Name der Schule, für die der Import sein soll. Dieser Wert gilt für alle Benutzer in den Eingabedaten. Sollte nur gesetzt werden, wenn die Schule nicht über die Eingabedaten gesetzt wird.
	Standard (string): nicht gesetzt
source_uid --source_uid	Eindeutige und unveränderliche Kennzeichnung der Datenquelle. Muss zwingend entweder in einer Konfigurationsdatei oder an der Kommandozeile gesetzt werden.
	Standard (string): nicht gesetzt
tolerate_ errors	Definiert die Anzahl an (für die Import-Software) nicht-kritischen Fehlern, die toleriert werden sollen, bevor der Import abgebrochen wird. Wird der Wert -1 verwendet, bricht der Import nicht ab und fährt mit dem nächsten Eingabedatensatz fort.
	Standard (int): 0
user_deletion	Veraltet. Bitte <code>deletion_grace_period</code> verwenden.
	Standard (object): nicht gesetzt
user_role -u --user_role	Definiert die Benutzerrolle für alle Eingabedatensätze. Diese Variable sollte nur gesetzt werden, wenn die Benutzerrolle nicht in den Eingabedaten enthalten ist und die Eingabedatensätze homogen alle die gleiche Benutzerrolle verwenden sollen. Erlaubte Werte sind <code>student</code> , <code>staff</code> , <code>teacher</code> und <code>teacher_and_staff</code> .
	Standard (string): nicht gesetzt
username	Enthält Informationen über die Erzeugung von Benutzernamen.

<sup>3</sup> <https://docs.python.org/2/library/datetime.html#strftime-and-strptime-behavior>

"default"-Schlüssel

Schlüssel	Beschreibung
	Standard (object): {"max_length": {...}}
username: max_length	\ Enthält Informationen über die Länge von Benutzernamen. Standard (object): {"default": .., "student": ..}
username: max_length: default	\ Länge eines Benutzernamens, wenn keine Konfiguration speziell für diesen Benutzer- typ (username:max_length:staff etc) existiert. Standard (int): 20

### 4.3. "default"-Schlüssel


Feedback 

Einige Einstellungen erlauben das Setzen von verschiedenen Werten, je nach Rolle des Benutzers, der gerade importiert wird. In einem solchen Fall gibt es immer den Schlüssel `default`, der automatisch verwendet wird, wenn es keinen Schlüssel in der Konfiguration für die betroffene Benutzerrolle gibt. Erlaubte Werte für die Benutzerrollen-Schlüssel sind `student`, `staff`, `teacher` und `teacher_and_staff`.

Es müssen nicht zwangsläufig Schlüssel für alle Benutzerrollen angegeben werden. Gilt für eine Einstellung z.B. das gleiche für Mitarbeiter und Lehrer und weicht nur der Wert für die Schüler-Benutzerrolle ab, so reicht es aus, `default` und `student` zu konfigurieren. In den Fällen `staff`, `teacher` und `teacher_and_staff` wird in Abwesenheit einer spezifischen Konfiguration automatisch auf `default` zurückgefallen:

```
{
  "scheme": {
    "username": {
      "default": "<:umlauts><firstname>[0].<lastname>[COUNTER2]",
      "student":
"<:umlauts><firstname>.<lastname><:lower>[ALWAYS COUNTER]"
    }
  }
}
```

### 4.4. Zuordnung von Eingabedaten zu Benutzerattributen

Feedback 

Während des Imports aus einer CSV-Datei müssen die Daten einer Zeile, Attributen des anzulegenden bzw. zu ändernden Benutzerobjekts zugeordnet werden. Diese Zuordnung geschieht im Konfigurationsobjekt `csv:mapping`. In ihm stehen Schlüssel-Wert-Paare: CSV-Spalte → Benutzerattribut.

Folgendes Beispiel zeigt wie der Import von drei Schülern an zwei Schulen konfiguriert werden kann. Die Schulverwaltungssoftware hat folgendes CSV produziert:

```
"Schulen", "Vorname", "Nachname", "Klassen", "Mailadresse", "phone"
"schule1,schule2", "Anton", "Meyer", "schule1-1A,schule2-2B", "anton@schule.local", ""
"schule1,schule2", "Bea", "Schmidt", "schule1-2B,schule2-1A", "bea@schule.local", "0421"
"schule2", "Daniel", "Krause", "schule2-1A", "daniel@schule.local", ""
```

Als erstes fällt auf, dass ein Schüler an zwei Schulen gleichzeitig eingeschrieben ist. Schulübergreifende Benutzerkonten wurden mit UCS@school 4.1R2 eingeführt (siehe *UCS@school 4.1 R2 v1 Release Notes*<sup>4</sup>) und werden von der Importsoftware unterstützt. Entsprechend sind die Namen der Klassen so kodiert, dass sie eindeutig einer Schule zugeordnet werden können. Anton geht also in die Klasse 1A der Schule `Schule1` und in die Klasse 2B der Schule `Schule2`.

Die Namen der Schulen bzw. Klassen sind ohne Leerzeichen und durch Komma getrennt, aufgelistet. Als Trennzeichen innerhalb einer CSV-Zelle wird das Komma verwendet, da dies implizit aus der Standard-

<sup>4</sup> <https://docs.software-univention.de/release-notes-ucsschool-4.1R2v1-de.html>

einstellung `csv:inccell-delimiter:default=","` aus `/usr/share/ucs-school-import/configs/user_import_defaults.json` übernommen wurde.

Folgende Konfiguration nutzt implizit die Standardeinstellung `csv:header_lines=1` aus `/usr/share/ucs-school-import/configs/user_import_defaults.json` und verwendet damit die Spaltennamen aus der CSV-Kopfzeile als Schlüssel.

```
{
  "csv": {
    "mapping": {
      "Schulen": "schools",
      "Vorname": "firstname",
      "Nachname": "lastname",
      "Klassen": "school_classes",
      "Mailadresse": "email",
      "phone": "phone"
    }
  }
}
```

Um die Konfiguration zu überprüfen kann ein Testlauf mit `--dry-run` gestartet werden. Anschließend steht in `/var/log/univention/ucs-school-import.log` ein Protokoll das Debug-Ausgaben enthält. Hier findet sich:

```
2016-06-28 17:47:25 INFO user_import.read_input:81 ----- Starting to
read users from input data... -----
[..]
2016-06-28 17:47:25 DEBUG base_reader.next:73 Input 3: ['schule1',
'Bea', 'Schmidt', 'schule1-2B,schule2-1A',
'bea@schule.local', 'Sch\xc3\xbcclerin mit Telefon', '0421-1234567890'] -
> {u'Schulen': u'schule1',
u'Vorname': u'Bea', u'phone': u'0421-1234567890', u'Nachname':
u'Schmidt', u'Klassen': u'schule1-2B,schule2-1A',
u'Mailadresse': u'bea@schule.local'}
```

Ab der zweiten Zeile ist dies folgendermaßen zu lesen:

- `Input 3`: dritte Zeile der Eingabedatei, die Kopfzeile mitgerechnet.
- `['schule1', 'Bea', 'Schmidt', 'schule1-2B,schule2-1A', 'bea@schule.local', '0421-1234567890']`: Die Eingabezeile mit bereits getrennten Spalten.
- `{u'Schulen': u'schule1', u'Vorname': u'Bea', u'phone': u'0421-1234567890', u'Nachname': u'Schmidt', u'Klassen': u'schule1-2B,schule2-1A', u'Mailadresse': u'bea@schule.local'}`: Die Zuordnung von Daten zu den Schlüsseln aus der CSV-Kopfzeile.

Das Einlesen aus der CSV-Datei ist gelungen. Die Daten wurden den Schlüsseln aus der CSV-Kopfzeile zugeordnet. Da diese in `csv:mapping` verwendet werden, kann nun weiter unten, beim Anlegen der Benutzer, die Zuordnung der Daten zu Benutzerattributen beobachtet werden:

```
2016-06-28 17:47:25 INFO user_import.create_and_modify_users:107
----- Creating / modifying users... -----
[..]
2016-06-28 17:47:25 INFO user_import.create_and_modify_users:128
Adding ImportStudent(name='B.Schmidt',
```

```

school='schule1',
dn='uid=B.Schmidt,cn=schueler,cn=users,ou=schule1,dc=uni,dc=dtr',
old_dn=None) (source_uid:NewDB
record_uid:bea@schule.local) attributes={'$dn$':
'uid=B.Schmidt,cn=schueler,cn=users,ou=schule1,dc=uni,dc=dtr',
'display_name': 'Bea Schmidt', 'record_uid': u'bea@schule.local',
'firstname': 'Bea',
'lastname': 'Schmidt', 'type_name': 'Student', 'school':
'schule1', 'name': 'B.Schmidt',
'disabled': '0', 'email': u'bea@schule.local', 'birthday': None, 'type':
'importStudent', 'schools': ['schule1'],
'password': 'xxxxxxxxxx', 'source_uid': u'NewDB', 'school_classes':
{'schule1': ['schule1-2B'],
'schule2': ['schule2-1A']}, 'objectType': 'users/
user'} udm_properties={u'phone': [u'0421-1234567890'],
'overridePWHistory': '1', 'overridePWLength': '1'}...

```

Hier ist nun zu sehen, dass Daten umgewandelt und Attributen zugeordnet wurden, sowie dass einige Attribute aus anderen Daten generiert wurden:

- *school\_classes* ist von einer kommaseparierten Liste zu einer Datenstruktur geworden.
- *name* und *record\_uid* sind aus den konfigurierten Schemata `scheme:username` und `scheme:record_uid` erzeugt worden.
- *phone* wurde in einem *udm\_properties* genannten Objekt gespeichert.

## Anmerkung

In *udm\_properties* werden Daten am Benutzerobjekt gespeichert, die nicht zu den Attributen der `ImportUser` Klasse gehören (siehe Abschnitt 8.1). Die Schlüssel entsprechen der Ausgabe des Kommandos:

```
udm users/user
```

Bei der obigen, langen Ausgabe handelt es sich um die Beschreibung eines `ImportUser` Objektes. Dieses zu kennen wird wichtig für die Programmierung von Hooks (siehe Abschnitt 8.2), mit denen vor und nach dem Anlegen, Ändern oder Löschen von Benutzern noch Aktionen ausgeführt werden können.

### 4.4.1. Sonderwerte

 Feedback 

Es existieren "Sonderwerte", die in der Konfiguration der Zuordnung (`mapping`) verwendet werden können:


- `__action`: Steht in einer CSV-Spalte immer die auf einen eingelesenen Benutzer anzuwendende Aktion als Buchstabe kodiert, so wird die Import-Software keine eigene Entscheidung darüber fällen, sondern dieser Anweisung folgen.

Anlegen (*add*): A, Ändern (*modify*): M oder Löschen (*delete*): D.

- `__ignore`: Der Inhalt dieser Spalte wird ignoriert. Sie kann z.B. verwendet werden, wenn die CSV-Datei leere Spalten, oder solche mit nicht zu importierenden Daten, enthält.
- `__role`: Der Inhalt dieser Spalte wird verwendet um die Rolle des Benutzers zu bestimmen. Gültige Werte sind `student`, `staff`, `teacher` und `teacher_and_staff`. Wenn die Rolle der zu importierenden Benutzer in einer Spalte angegeben wird, darf die Option `user_role` nicht (oder nur auf `null`) gesetzt werden.




## 4.4.2. Eigene Erweiterungen hinzufügen

 Feedback 

Weitere, eigene Interpretationen von Eingabewerten können in einer von `ucsschool.importer.reader.csv_reader.CsvReader` abgeleiteten Klasse (siehe Abschnitt 8.3) in der Methode `handle_input()` erzeugt werden. Als Beispiel kann `handle_input()` in `ucsschool.importer.legacy.legacy_csv_reader.LegacyCsvReader` dienen, welches für den Import des alten CSV-Dateiformats den Sonderwert `__activate` (neue Benutzer de/aktivieren) hinzufügt.

Um Unterstützung für den Import von anderen Dateiformaten als CSV (JSON, XML etc) hinzuzufügen, kann von `ucsschool.importer.reader.base_reader.BaseReader` abgeleitet werden (siehe Abschnitt 8.3).

## 4.5. Formatierungsschema

 Feedback 

Es kann wünschenswert – oder wie im Fall von Benutzername und E-Mail-Adresse notwendig – sein, Attribute aus den Werten anderer Attribute zu erzeugen. Zum Beispiel speichern und exportieren Schulverwaltungssoftwares häufig keine Benutzernamen und E-Mail-Adressen, die zur eingesetzten Infrastruktur passen.


Aus diesem Grund unterstützt die Importsoftware die Erzeugung von Attributen mit Hilfe von konfigurierbaren Schemata. Das Format ist das gleiche wie das bei den *UCS-Benutzervorlagen*<sup>5</sup> eingesetzte. Es existieren dedizierte Konfigurationsschlüssel für die Attribute `email`, `record_uid` und `username`. Darüber hinaus können Schemata für beliebige Univention Directory Manager Attribute (mit dem Namen des Attributs als Schlüssel) hinterlegt werden.

Im folgenden Beispiel würde allen aus den Eingabedaten übernommenen Telefonnummern die Ländervorwahl vorangestellt (und die führende 0 der Städtevorwahl entfernt), sowie die E-Mail-Adresse aus Vor- und Nachname berechnet:

```
{
  "scheme": {
    "email": "<firstname>[0].<lastname>@<maildomain>",
    "record_uid": "<email>",
    "phone": "+49-<phone>[1:]"
  }
}
```

Sollen Attribute in modifizierter Form für die Erzeugung weiterer Attribute verwendet werden (z.B. nur der erste Teil eines Doppelnamens für eine E-Mail-Adresse), so kann dies mit Hilfe von *Format-Hooks* geschehen. Ihre Erstellung und Verwendung wird in Abschnitt 8.2.1 beschrieben.

## 4.6. Einmalige Benutzernamen und Email-Adressen

 Feedback 

Benutzernamen und Email-Adressen müssen in der gesamten Domäne, nicht nur an einer Schule, einmalig sein. Darüber hinaus kann es die Anforderung geben, dass Benutzernamen und Email-Adressen auch "historisch einmalig" sind, also auch dann nicht wiederverwendet werden, wenn die vorherigen Konten bereits lange gelöscht sind.

Aus diesem Grund können zur Erzeugung von Benutzernamen und Email-Adressen, über die üblichen Variablen in Formatierungsschema (siehe Abschnitt 4.5) hinaus, spezielle Zählervariablen verwendet werden. Diese Variablen werden bei ihrer Verwendung automatisch hochgezählt. Ihr Wert wird pro Benutzername bzw. Email-Adresse gespeichert. Es existieren zwei Variablen, die sich darin unterscheiden wie die ersten Benutzer mit gleichem Benutzernamen bzw. Email-Adresse, benannt werden:

<sup>5</sup> <http://docs.software-univention.de/handbuch-4.4.html#users:templates>

- [ALWAYS\_COUNTER] ist ein Zähler der bei seiner ersten Verwendung eine 1 einsetzt. Benutzernamen für anton wären: anton1, anton2, anton3... Analog für anton@dom.ain: anton1@dom.ain, anton2@dom.ain, anton3@dom.ain...
- [COUNTER2] ist ein Zähler der bei seiner ersten Verwendung keine Zahl einsetzt, erst bei seiner zweiten. Benutzernamen für anton wären: anton, anton2, anton3... Analog für anton@dom.ain: anton@dom.ain, anton2@dom.ain, anton3@dom.ain...

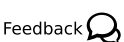
Im folgenden Beispiel würden für Bea Schmidt die Benutzernamen b.schmidt, b.schmidt2, b.schmidt3 sowie Email-Adressen bea.schmidt1@dom.ain, bea.schmidt2@dom.ain, bea.schmidt3@dom.ain erzeugt werden:

```
{
  "scheme": {
    "username": {
      "default":
"<:umlauts><firstname>[0].<lastname><:lower>[COUNTER2]"
    },
    "email": "<firstname>.<lastname>[ALWAYS_COUNTER]@<maildomain>"
  },
  "maildomain": "dom.ain",
}
```

Die [0] im Beispiel bedeutet, dass nur das erste Zeichen des davor stehenden Attributes genommen wird. Es ist auch möglich Bereiche anzugeben. Weitere Informationen dazu finden sich im Kapitel *Benutzervorlagen*<sup>6</sup> des UCS Handbuchs.

Um neue Zählervariablen hinzuzufügen, muss von der Klasse `ucsschool.importer.utils.username_handler.UsernameHandler` abgeleitet und die Methode `counter_variable_to_function()` überschrieben werden (siehe Abschnitt 8.3). Um diese neuen Zählervariablen auch für Email-Adressen zu verwenden, muss `ucsschool.importer.utils.username_handler.EmailHandler` von der neuen, abgeleiteten `UsernameHandler` Klasse *sowie* von `ucsschool.importer.utils.username_handler.EmailHandler` abgeleitet werden. Um Zählervariablen *nur* für Email-Adressen hinzuzufügen muss nur von der Klasse `ucsschool.importer.utils.username_handler.EmailHandler` abgeleitet, und oben beschriebene Methoden überschrieben werden.

## 4.7. Benutzer löschen



Das Löschen von Benutzern kann in zwei Varianten konfiguriert werden:

- Das Benutzerkonto wird sofort oder später gelöscht, nicht deaktiviert.

Dies entspricht dem Löschen eines Kontos im Univention Management Console-Modul **Benutzer** (siehe [ucs-handbuch]), zum definierten Zeitpunkt.

Diese Variante wird ausgewählt durch das Setzen von `deletion_grace_period:deletion` auf einen Wert kleiner oder gleich dem von `deletion_grace_period:deactivation`. Ist der Wert von `deletion_grace_period:deletion=0`, wird sofort (während des Imports) gelöscht. Ist der Wert größer als 0, wird ein Verfallsdatum im Univention Directory Manager-Attribut `ucsschoolPurgeTimestamp` gespeichert. Der Benutzer wird erst an diesem Tag durch einen Cron Job gelöscht.

- Das Benutzerkonto wird erst deaktiviert und später gelöscht.

Dies entspricht dem Deaktivieren oder Setzen eines Kontoablaufdatums im Univention Management Console-Modul **Benutzer** und späteren Löschens in selbigem. Der Benutzer wird zuerst deaktiviert (kann sich

<sup>6</sup> <http://docs.software-univention.de/handbuch-4.4.html#users:templates>



nicht mehr einloggen), aber erst zu dem gesetzten Datum gelöscht. Bis zum finalen Löschen kann das Benutzerkonto noch reaktiviert werden, sollte es durch einen Import wieder "angelegt" werden.

Diese Variante wird ausgewählt durch das Setzen von `deletion_grace_period:deletion` auf einen Wert größer dem von `deletion_grace_period:deactivation`. Ist der Wert von `deletion_grace_period:deactivation=0`, wird der Account sofort (während des Imports) deaktiviert. Ist der Wert größer als 0, wird ein Verfallsdatum im Univention Directory Manager-Attribut `user_expiry` gespeichert. Der Benutzer kann sich ab diesem Tag nicht mehr anmelden. Der Wert von `deletion_grace_period:deletion` wird wie in der ersten Variante beschrieben im Univention Directory Manager-Attribut `ucsschoolPurgeTimestamp` gespeichert und die Löschung später durch einen Cron Job durchgeführt.

Um eine der Löschvarianten zu ändern oder neue hinzuzufügen, muss von der Klasse `ucsschool.importer.mass_import.user_import.UserImport` abgeleitet und die Methode `do_delete()` überschrieben werden (siehe Abschnitt 8.3).



## Kapitel 5. Schulwechsel

Bei einem Schulwechsel verlässt ein Schüler oder Lehrer seine ursprüngliche Schule A und wird an einer anderen Schule B aufgenommen. Hierbei sind drei Szenarien denkbar:

- Schule A und Schule B werden vom gleichen Quellverzeichnis abgedeckt und gemeinsam verwaltet. D.h. die für den Benutzer hinterlegte Schule ändert sich in einem Schritt von Schule A auf Schule B. Die Importsoftware kann das Benutzerobjekt verschieben, ohne dass sich Daten wie Benutzername, User-ID, Telefonnummer oder Passwort ändern.
- Schule A und Schule B werden vom gleichen Quellverzeichnis abgedeckt, die beiden Schulverwaltungen pflegen die Daten ihrer Schüler oder Lehrer jedoch unabhängig voneinander. Der Schulwechsel findet also in zwei Schritten statt. Es können zwei Szenarien auftreten:
  - Der Benutzer wird an Schule A entfernt und erst später an Schule B neu aufgenommen. Wurde das Benutzerkonto gelöscht und nicht deaktiviert, verliert der Benutzer alle Benutzerdaten und erhält ein komplett neues Benutzerkonto inkl. Benutzernamen, User-ID, Passwort etc.
  - Der Benutzer wird an Schule B aufgenommen, bevor er an Schule A entfernt wird. Das Benutzerkonto wird kurzfristig an zwei Schulen repliziert, die Daten bleiben während der gesamten Zeit (auch nach Entfernen von Schule A) erhalten.
- Schule A und Schule B werden von unterschiedlichen Quellverzeichnissen abgedeckt. Der Benutzer wird in Schule A entfernt und vorher oder später in Schule B neu angelegt. Der Benutzer erhält dann einen neuen Benutzernamen, User-ID, Passwort etc. Das Übernehmen des Benutzerkontos ist nicht ohne weiteres möglich.



## Kapitel 6. Schuljahreswechsel

Der Schuljahreswechsel erfolgt in drei Schritten:

1. In der Schulverwaltungssoftware findet der Schuljahreswechsel statt. Anschließend wird ein Export der Benutzer aus der Schulverwaltungssoftware gemacht.
2. Die Klassen der Schüler die die Schule verlassen, werden im Univention Management Console-Modul **Klassen** gelöscht.
3. Durchführung des Imports der zuvor Exportierten Daten.



# Kapitel 7. Schulübergreifende Benutzerkonten

## 7.1. Schulspezifisches *sambahome* ..... 32

Seit UCS@school 4.1 R2 werden schulübergreifende Benutzerkonten unterstützt. Ein Benutzerobjekt existiert im LDAP-Verzeichnis nur einmal: an seiner primären Schule (Attribut *school*). An weitere, festgelegte Schulen (Attribut *schools*) wird nur ein Ausschnitt des LDAP-Verzeichnisses seiner primären Schule repliziert: sein Benutzerobjekt und die Standardgruppen. Verlässt der Benutzer die Schule (durch Entfernen aus dem Attribut *schools*), so wird sein Benutzerobjekt dort gelöscht und nicht mehr dorthin repliziert. Bei der Verwendung von schulübergreifenden Benutzerkonten gilt es einige Dinge zu beachten.

Der Klassenarbeitsmodus und die Materialverteilung arbeiten grundsätzlich so, dass sie auf dem Schulserver an dem sie veranlasst wurden, die hochgeladenen Dateien in die Heimatverzeichnisse der betroffenen Benutzer kopieren. Befindet sich ein Heimatverzeichnis nicht auf dem Server, scheitert dies.

Windows-Clients verwenden das LDAP-Attribut *homeDirectory* (LDAP-Attribut *sambaHomePath* bzw. Univention Directory Manager-Attribut *sambahome*) um beim Einloggen das Netzwerklaufwerk mit den Dokumenten des Benutzers einzubinden. Wenn die primäre Schule eines Benutzers eine andere ist, als die, an der er gerade eine Klassenarbeit schreiben soll, so existiert sein Heimatverzeichnis dort unter Umständen nicht.

Es existieren drei Varianten des Umgangs mit der Univention Directory Manager-Attribut *sambahome*, mit folgenden Vor- und Nachteilen:

1. *sambahome* wird regulär durch das Import-Skript gesetzt und nicht manuell verändert. *sambahome* ist dann immer ein Verzeichnis auf dem Schulserver der primären Schule des jeweiligen Benutzers.
  - pro: Es existiert genau ein Heimatverzeichnis auf einem Server für alle Clients der Domäne (egal an welcher Schule).
  - contra: Klassenarbeitsmodus und Materialverteilung funktionieren nicht an anderen Schulen als der primären. Beim regulären Arbeiten gibt es ein hohes Datenaufkommen zwischen den Schulen.
2. *sambahome* wird durch das Import-Skript für alle auf einen (per Univention Configuration Registry) festgesetzten, zentralen, Server gesetzt.
  - pro: wie bei 1.
  - contra: wie bei 1.
3. *sambahome* wird durch das Import-Skript auf einen Server mit einem Alias-Namen gesetzt. Je nach dem an welcher Schule sich der Benutzer befindet, bekommt er vom DNS-Server eine andere IP-Adresse für den gleichen Servernamen geliefert.
  - pro: Klassenarbeitsmodus und Materialverteilung funktionieren, an der jeweiligen Schule an der sie stattfinden, für alle Benutzer - egal ob es ihre primäre Schule ist oder nicht. Kein Datenverkehr zwischen Schulen.
  - contra: Es wird an jeder Schule eines Benutzers ein eigenes Heimatverzeichnis für ihn angelegt. Einmaliger Installationsaufwand: An jeder Schule müssen ein paar Univention Configuration Registry-Variablen eingestellt werden.

Es folgt eine Anleitung zur Einrichtung der dritten Variante.

## 7.1. Schulspezifisches sambahome

Die folgenden Befehle müssen, mit angepassten Hostnamen und IP-Adressen, auf jedem Schulserver ausgeführt werden:

```
# UCR Variablen verfügbar machen
eval "$(ucr shell)"

# Name (Alias) des Servers auf dem das Heimatverzeichnis liegt
ucr set ucsschool/import/set/sambahome=schoolserver

# DNS-Eintrag schoolserver.$domainname -> IP-Adresse des *jeweiligen*
# Schulservers
ucr set "connector/s4/mapping/dns/host_record/schoolserver.$domainname/
static/ipv4 "=172.16.3.12

# DNS-Eintrag aktivieren
systemctl restart univention-s4-connector.service
```

Folgendes muss auf dem DC Master ausgeführt werden:

```
# Name (Alias) des Servers auf dem das Heimatverzeichnis liegt (wird vom
# Import ausgewertet)
ucr set ucsschool/import/set/sambahome=schoolserver

# DNS Forward-Eintrag einrichten (school.local durch korrekte Domäne
# ersetzen,
# IP-Adresse eines zentralen Server, z.B. des DC Master, verwenden)
udm dns/host_record create \
  --superordinate "zoneName=school.local,cn=dns,$ldap_base" \
  --set name=schoolserver \
  --set a=172.16.1.1
```

Der Befehl `host schoolserver` sollte nun auf allen Schulservern die IP-Adresse des jeweiligen Schulservers liefern. Mit `nslookup schoolserver` können die gleiche DNS-Anfrage komfortabel an verschiedene DNS-Server geschickt werden.



# Kapitel 8. Erweiterung um neue Funktionalität


8.1. Die <code>ImportUser</code> Klasse .....	33
8.2. Hooks .....	34
8.2.1. <i>Format-Hooks</i> .....	36
8.3. <i>Subclassing</i> .....	37
8.3.1. <i>Abstract Factory</i> .....	38
8.3.2. Überschreiben einer Methode .....	38
8.3.3. Ersetzen durch eigene Klasse .....	39
8.4. <i>Prüfung der Konfiguration</i> .....	39

Die UCS@school Importsoftware ist so geschrieben worden, dass ihre Funktionalität möglichst einfach und gleichzeitig umfangreich veränderbar und erweiterbar ist. Dazu stehen zwei Methoden zur Verfügung:

- Das Ausführen von Aktionen zu bestimmten Zeitpunkten mit der Hilfe von Python-Hooks.
- Die Veränderung der Importsoftware durch das Überschreiben von Teilen des Programmcodes.

Setzt eine Erweiterung eine bestimmte Konfiguration voraus, können zusätzliche Prüfungen registriert werden, die vor dem Start des Importvorganges gestartet werden.

## 8.1. Die `ImportUser` Klasse

Feedback 

Die Klasse `ImportUser` wird verwendet um Daten von eingelesenen oder zu ändernden Benutzern zu speichern. An Objekten der `ImportUser` Klasse können folgende Attribute gesetzt werden:

**Tabelle 8.1. Attribute der `ImportUser` Klasse**

Attribut	Typ	Beschreibung
<code>name</code>	string	Benutzername
<code>school</code>	string	Primäre Schule des Benutzers (Position des Objektes im LDAP)
<code>schools</code>	string / liste	Alle Schulen des Benutzers inkl. der primären Schule, als ein kommaseparierter String oder als Liste von Strings.
<code>firstname</code>	string	Vorname
<code>lastname</code>	string	Nachname
<code>birthday</code>	string	Geburtsdatum im Format JJJJ-MM-TT
<code>email</code>	string	E-Mail-Adresse
<code>password</code>	string	Passwort (wird für neue Benutzer automatisch erzeugt, wenn nicht in den Eingabedaten vorhanden).
<code>disabled</code>	boolean	Ob ein neuer Benutzer deaktiviert erzeugt werden soll.
<code>school_classes</code>	string / object	Klassen in denen der Benutzer ist.  Als String im Format <code>schule1-1A</code> , <code>schule1-2B</code> , <code>schule2-1A</code> oder als Python <i>dictionary</i> : <code>{"schule1": ["1A", "2B"], "schule2": ["1A"]}</code> .  Es können Klassen aus mehreren Schulen aufgelistet werden; diese Schulen müssen alle in <code>schools</code> auftauchen. Falls die Klassenzuordnung nicht


Attribut	Typ	Beschreibung
		per Import stattfinden soll, weil z.B. die Klassenzuordnungen der Lehrkräfte über die UMC-Module <b>Klassen zuordnen</b> bzw. <b>Lehrer zuordnen</b> stattfinden sollen, kann in der Konfiguration <code>school_classes_keep_if_empty</code> auf <code>true</code> gesetzt werden. Der Import verhält sich dann so, dass wenn der Wert für <code>school_classes</code> leer ist, bestehende Klassenzuordnungen nicht verändert werden.
<code>source_uid</code>	string	Kennzeichnung der Datenquelle
<code>record_uid</code>	string	ID des Benutzers in der Datenquelle
<code>udm_properties</code>	object	Alle anderen Univention Directory Manager Attribute die in den Eingabedaten enthalten waren, werden in dieses Python <i>dictionary</i> gespeichert.  Oben stehende Attribute und ihre <i>Univention Directory Manager-Pendants</i> ( <code>name</code> → <code>username</code> , <code>email</code> → <code>mailPrimaryAddress</code> ) sind hier nicht erlaubt.

Weitere interessante Attribute, die jedoch nur gelesen und nicht modifiziert werden sollten, sind:

**Tabelle 8.2. Attribute der `ImportUser` Klasse (nur lesen)**

Attribut	Typ	Beschreibung
<code>dn</code>	string	DN des Benutzer-Objekts im LDAP, wenn es jetzt gespeichert werden würde.
<code>entry_count</code>	int	Zeile in CSV-Datei, aus der Daten des Benutzers stammen. Ist 0, wenn dies nicht zutrifft.
<code>input_data</code>	list	Unveränderte Eingabedaten aus der CSV-Datei, bereits zu Elementen einer Liste aufgeteilt.
<code>ucr</code>	object	Eine Univention Configuration Registry-Instanz zum Auslesen von Univention Configuration Registry-Einstellungen.

## 8.2. Hooks

Feedback 

*Hooks*<sup>1</sup> sind Stellen im Programmcode, an die zusätzlicher Code "gehängt" werden kann. Für den Benutzerimport sind acht Stellen vorgesehen: jeweils vor und nach dem Anlegen, Ändern, Löschen oder Verschieben von Benutzern. Zusätzlich gibt es *Format-Hooks* die vor dem Erstellen eines Attributes aus anderen Attributen (siehe Abschnitt 4.5) ausgeführt werden. Diese werden weiter unten in Abschnitt 8.2.1 separat behandelt.

Zur Nutzung der Hook-Funktionalität muss eine eigene Python-Klasse erstellt werden, die von `ucsschool.importer.utils.user_pyhook.UserPyHook` ableitet. In der Klasse können Methoden `pre_create()`, `post_create()`, etc. definiert werden, welche zum jeweiligen Zeitpunkt ausgeführt werden. Der Name der Datei mit der eigenen Klasse muss auf `.py` enden und im Verzeichnis `/usr/share/ucsschool-importer/pyhooks` abgespeichert werden.

### Anmerkung

Der Quellcode der Klasse `UserPyHook` ist zu finden in `/usr/lib/python2.7/ucsschool/importer/utils/user_pyhook.py`. In ihm sind alle Methoden und Signaturen dokumentiert.

<sup>1</sup> [https://de.wikipedia.org/wiki/Hook\\_\(Informatik\)](https://de.wikipedia.org/wiki/Hook_(Informatik))

Die Methoden der Hook-Klasse bekommen als Argument das Benutzerobjekt übergeben, das aus dem LDAP geladen wurde bzw. im LDAP gespeichert werden soll. Veränderungen an diesem Objekt werden bei dessen Abspeicherung direkt ins LDAP übernommen.

Die Klasse definiert ein *dictionary* *priority* mit dessen Hilfe eine Reihenfolge definiert werden kann, sollten mehrere Hook-Klassen mit zum Einsatz kommen, die die gleichen Methoden definieren. Die Namen der Methoden die ausgeführt werden sollen sind die Schlüssel, Methoden mit höheren Zahlen werden zu erst ausgeführt. Ist der Wert *None*, wird die Methode deaktiviert.

Zur Erstellung einer eigenen Hook-Klasse kann das Beispiel in `/usr/share/doc/ucs-school-import/hook_example.py` kopiert und angepasst werden. Alle Funktionen die nicht ausgeführt werden sollen, sollten entweder gelöscht oder deaktiviert werden (indem ihr Wert in *priority* auf *None* gesetzt wird). Das könnte Beispielsweise so aussehen:

```
import datetime
import shutil

from ucsschool.importer.utils.user_pyhook import UserPyHook

class MyHook(UserPyHook):
    supports_dry_run = True # Hook Klasse wird auch während eines
                           # dry-runs ausgeführt

    priority = {
        "pre_create": 1,
        "post_create": None, # Funktion ist deaktiviert
        "pre_remove": 1
    }

    def pre_create(self, user):
        if user.birthday:
            bday = datetime.datetime.strptime(user.birthday,
                                              "%Y-%m-%d").date()

            if bday == datetime.date.today():
                self.logger.info("%s has birthday.", user)
                user.udm_properties["description"] = "Herzlichen \
                    Glückwunsch"

    def post_create(self, user):
        # Diese Funktion ist deaktiviert.
        self.logger.info("Running a post_create hook for %s.", user)

    def pre_remove(self, user):
        # backup users home directory
        self.logger.info("Backing up home directory of %s.", user)
        user_udm = user.get_udm_object(self.lo)
        homedir = user_udm["unixhome"]
        target = "/var/backup/{%s}".format(user.name)
        if self.dry_run:
            self.logger.info("Dry-run: would copy %r to %r.", homedir,
                             target)
        else:
            shutil.copy2(homedir, target)
```

- Da die Klassenvariable `supports_dry_run = True` gesetzt ist, wird der Hook auch während eines `dry-run` ausgeführt.

- In `pre_create()` wird bei einem neuen Benutzer ein Gruß am Benutzerobjekt gespeichert, wenn er Geburtstag hat.
- Die `post_create()` Funktion ist durch das `None` in `priority` deaktiviert.
- In `pre_remove()` wird, wenn nicht während eines `dry-run` ausgeführt, ein Backup des Heimatverzeichnisses des Benutzers gemacht, bevor er gelöscht wird.

In `pre_create()` wird in `udm_properties` an den Schlüssel `description` der Wert Herzlichen Glückwunsch geschrieben. Das explizite Abspeichern des `user` Objektes ist in dieser Funktion nicht nötig, da dies ja beim auf den Hook folgenden `create` geschieht.

In der Funktion wird außerdem mit `self.logger.info()` ein Text zu Protokoll gegeben. Es handelt sich bei `self.logger` um eine Instanz eines *Python logging*<sup>2</sup> Objekts.


In `pre_remove()` wird das Heimatverzeichnis des Benutzers benötigt. Da dies nicht eines der direkt am Objekt stehenden Daten ist (siehe Abschnitt 8.1), muss zuerst das gesamte Benutzerobjekt aus dem LDAP geladen werden. Dies tut `user.get_udm_object()`, welches als Argument ein LDAP-Verbindungsobjekt erwartet. Dieses ist im *Hook-Objekt* an `self.lo` gespeichert.

### Achtung

Falls das Benutzerobjekt in einem *post-Hook* geändert werden soll, so ist es möglich `user.modify_without_hooks()` auszuführen, aber generell sollte ein erneutes Modifizieren *nach* dem Speichern vermieden werden.

Die Methoden `create()`, `modify()` und `remove()` des Benutzerobjekts sollten von Hook-Methoden nicht ausgeführt werden, da dies zu einer Rekursion führen kann.

## 8.2.1. Format-Hooks

 Feedback 

*Format-Hooks* erlauben es Attribute, nur für den Zeitraum ihrer Verwendung als Daten eines Formatierungsschemas, zu modifizieren. Der häufigste Anwendungsfall ist die Kürzung von Vor- und Nachnamen während der Erzeugung von E-Mail-Adressen und Benutzernamen.

Die eckigen Klammern im Formatierungsschema erlauben es zwar die Länge von Attributen statisch einzuschränken, aber sie erlauben z.B. kein Trennen an bestimmten Zeichen. Ein Beispiel für einen *Format-Hook* bei der Erzeugung der Attribute `username` und `email`, die Attribute `firstname` und `lastname` an Freizeichen und Bindestrichen trennt, ist in `/usr/share/doc/ucs-school-import/format_hook_example.py` zu finden.

Ein *Format-Hook* ist eine Klasse, die von `ucsschool.importer.utils.format_pyhook.FormatPyHook` abgeleitet ist. Der Name der Datei mit der eigenen Klasse muss, wie bei den regulären PyHooks, auf `.py` enden und im Verzeichnis `/usr/share/ucs-school-import/pyhooks` abgespeichert werden.

*Format-Hooks* haben die Methoden `patch_fields_staff()`, `patch_fields_student()`, `patch_fields_teacher()` und `patch_fields_teacher_and_staff()` von der immer nur diejenige ausgerufen wird, die zu der Rolle des zu erzeugenden / bearbeitenden Benutzers passt. `priority` hat die gleiche Funktion wie bei den regulären PyHooks. Das Klassenattribut `properties` enthält eine Liste von Attributnamen. Der *Format-Hook* wird nur für diese Attribute ausgeführt. Das Beispiel würde nur bei der Erzeugung von `username` und `email` ausgeführt und bei `birthday`, `firstname`, `school_classes`, usw. nicht. Hier können auch Univention Directory Manager-Attribute aus `udm_properties` aufgeführt werden. Aus Gründen der Performance ist es wichtig hier nur die Attribute aufzuführen, die tatsächlich geändert werden sollen.

<sup>2</sup> <https://docs.python.org/2/library/logging.html>


Den Methoden werden die Argumente `property_name` und `fields` übergeben. `property_name` enthält den Namen des Benutzerattributs, das gerade erzeugt werden soll und `fields` ist ein *dictionary*, welches alle Attribute und Werte des Benutzerobjekts zu diesem Zeitpunkt enthält, aus denen besagtes Attribut berechnet werden soll. Durch das Ändern von Werten in `fields` wird Einfluss genommen auf das Ergebnis des darauf folgenden Formatierens.

Im Beispiel werden bei `staff` und `teacher` Benutzern die Vor- und Nachnamen getrennt, wenn das `username` Attribut erzeugt wird, und bei `student` und `teacher_and_staff` bei der Erzeugung von `email`.

Stünde in der Konfiguration z.B. `csv:mapping:email=<firstname><lastname>@<maildomain>` so würde bei der Erzeugung des `email` Attributs eines `students` ein Vorname Hans-Otto gekürzt zu Hans. Mit einem Nachnamen Mayer und einer Domäne `univention.de` würde daraus die E-Mail-Adresse `hans.mayer@univention.de` erzeugt.

Die Modifikationen eines *Format-Hooks* sind nur während der Erzeugung *eines* Attributs gültig. Sie haben weder direkte Auswirkung auf das Benutzerobjekt noch auf die Erzeugung anderer Attribute. Existieren mehrere *Format-Hooks* für das *gleiche* Attribut, so werden sie nacheinander ausgeführt und das von einem *Format-Hook* modifizierte *fields dictionary* dem nächsten *Format-Hook* übergeben.

## 8.3. Subclassing

Feedback 

Hooks erlauben das Ausführen von neuem Code zu bestimmten Zeitpunkten. Sie erlauben aber nicht bestehenden Code zu verändern. In einer objektorientierten Sprache wie Python wird dies üblicherweise getan, indem eine Klasse modifiziert wird. Soll für einen bestimmten Fall nur ein Teil der Klasse verändert werden, wird von ihr abgeleitet und nur dieser Teil verändert, der unveränderte Teil wird geerbt.

Folgendes Beispiel zeigt, wie der Klasse, welche die historisch einmaligen Benutzernamen erzeugt, eine weitere Variable hinzugefügt werden kann. Ein weiteres Beispiel ist in `/usr/share/doc/ucs-school-import/subclassing_example.py` zu finden.

```
from ucsschool.importer.utils.username_handler import UsernameHandler

class MyUsernameHandler(UsernameHandler):
    @property
    def counter_variable_to_function(self):
        name_function_mapping = super(MyUsernameHandler,
self).counter_variable_to_function
        name_function_mapping["[ALWAYSWITHZEROS]"] =
self.always_counter_with_zeros
        return name_function_mapping

    def always_counter_with_zeros(self, name_base):
        number_str = self.always_counter(name_base)
        number_int = int(number_str)
        new_number_str = "{:04}".format(number_int)
        return new_number_str
```

In `counter_variable_to_function()` wird den existierenden beiden Variablen eine weitere hinzugefügt und auf die neue Funktion verwiesen. `always_counter_with_zeros()` verwendet `always_counter()` zur Erzeugung der nächsten freien Zahl, schreibt diese aber dann so um, dass sie immer vier Stellen lang ist (der Anfang wird mit Nullen aufgefüllt).

Wird die Klasse unter `/usr/local/lib/python2.7/dist-packages/usernames_with_zeros.py` abgespeichert, so kann sie unter Python als `usernames_with_zeros.MyUsernameHandler` verwendet werden. Zuvor muss das Verzeichnis angelegt werden:

```
mkdir -p /usr/local/lib/python2.7/dist-packages
```

Ob Python die Klasse findet, lässt sich testen mit:

```
python -c 'from usernames_with_zeros import MyUsernameHandler'
```


(Es sollte keine Ausgabe geben.)

Die neue Funktionalität lässt sich testen mit:

```
# python
>>> from usernames_with_zeros import MyUsernameHandler
>>> print MyUsernameHandler(15).format_username("Anton[ALWAYS_COUNTER]")
Anton1
>>> print
MyUsernameHandler(15).format_username("Anton[ALWAYS_WITH_ZEROS]")
Anton0002
>>> print
MyUsernameHandler(15).format_username("Anton[ALWAYS_WITH_ZEROS]")
Anton0003
>>> exit()
```

Es gibt jetzt eine neue Klasse mit der neuen Funktionalität. Die Importsoftware muss nun noch dazu gebracht werden, diese neue, ihr nicht bekannte Klasse zu verwenden.

### 8.3.1. Abstract Factory

Feedback 


Die Architektur der Importsoftware ist als *Abstrakte Fabrik (Abstract Factory)*<sup>3</sup> implementiert. In ihr wird die Erzeugung von Objekten zentralisiert. Sie zeichnet sich u.a. dadurch aus, dass sie erlaubt das Austauschen mehrerer Komponenten einer Software konsistent zu halten. Im Fall der Importsoftware ist die *abstractfactory* jedoch nicht Abstrakt, alle Methoden wurden implementiert.

An allen Stellen der Importsoftware die z.B. mit dem Einlesen von CSV-Dateien zu tun haben, wird nicht die Klasse `ucsschool.importer.reader.csv_reader.CsvReader` direkt instantiiert, sondern es wird von der eingesetzten *factory* eine Instanz verlangt (`factory.make_reader()`) und verwendet. Welche Klasse dem verwendeten Objekt zugrunde liegt ist nicht bekannt, sie muss nur die Methoden der ersetzten Klasse mit der gleichen Signatur implementieren. Auf diese Art könnte z.B. der CSV-Reader durch einen JSON-Reader ersetzt werden. Alles was dann zu tun bleibt, ist, die *factory* zu verändern. Dies kann auf zwei Arten geschehen:

- Überschreiben einzelner Methoden der `DefaultUserImportFactory` Klasse.
- Ersetzen von `DefaultUserImportFactory` durch eine eigene Klasse.

Welche Methode gewählt wird, hängt davon ab ob die Anpassungen nur punktuell sind, oder ob es sich um ein größeres Umschreiben der Importsoftware handelt.

### 8.3.2. Überschreiben einer Methode

Feedback 


Es ist möglich die Methoden der `DefaultUserImportFactory` Klasse einzeln zu überschreiben, ohne ihren Code zu ändern. Damit die *factory* Objekte der `MyUsernameHandler` Klasse aus dem obigen Beispiel beim Aufruf von `make_username_handler()` liefert, muss in die Konfiguration folgendes eingetragen werden (siehe Konfigurationsoption `classes`):

```
{
```

<sup>3</sup> [https://de.wikipedia.org/wiki/Abstrakte\\_Fabrik](https://de.wikipedia.org/wiki/Abstrakte_Fabrik)

```
"classes": {
    "username_handler": "usernames_with_zeros.MyUsernameHandler"
  }
}
```

### 8.3.3. Ersetzen durch eigene Klasse

 Feedback 

Sollen umfangreichere Änderungen an der Importsoftware durchgeführt werden, kann von `ucsschool.importer.default_user_import_factory.DefaultUserImportFactory` abgeleitet und ihre Methoden ersetzt werden. In der Konfigurationsdatei kann die zu nutzende *factory*-Klasse über den Schlüssel `factory` als voller Python-Pfad angegeben werden.

Obiges Beispiel lässt sich anstatt in der Konfiguration `classes:username_handler` zu setzen auch so lösen:

```
from ucsschool.importer.default_user_import_factory import
    DefaultUserImportFactory
from usernames_with_zeros import MyUsernameHandler


class MyUserImportFactory(DefaultUserImportFactory):
    def make_username_handler(self, max_length):
        return MyUsernameHandler(max_length)
```

Wird diese Datei nun als `/usr/local/lib/python2.7/dist-packages/my_userimport_factory.py` abgespeichert, so kann sie in der Konfiguration zur Verwendung als *factory* für die Importsoftware folgendermaßen aktiviert werden:

```
{
    "factory": "my_userimport_factory.MyUserImportFactory"
}
```

Der nächste Importvorgang lädt nun anstelle der `DefaultUserImportFactory` die `MyUserImportFactory` und wenn in der Importsoftware ein Objekt zur Erzeugung von Benutzernamen angefordert wird, so wird die neue Klasse entscheiden, das eines vom Typ `MyUsernameHandler` geliefert wird.

## 8.4. Prüfung der Konfiguration

 Feedback 

Nach dem Einlesen der Konfigurationsdateien und vor dem eigentlichen Start des Importvorgangs, laufen Tests, die die Korrektheit und Konsistenz der Konfiguration prüfen. Der Code für die Tests wird aus Python-Modulen im Verzeichnis `/usr/share/ucs-school-import/checks/` geladen. Damit ein Modul aus diesem Verzeichnis ausgeführt wird, muss sein Name (ohne `.py`) in der JSON-Konfigurationsdatei in der Liste unter dem Schlüssel `configuration_checks` vorkommen:

```
{
    "configuration_checks": ["defaults", "mychecks"]
}
```

Das Modul `defaults` führt die Standardprüfungen durch. Es sollte normalerweise Teil der Liste sein.

Um eigene Prüfungen hinzuzufügen, muss eine Klasse geschrieben werden, die von `ucsschool.importer.utils.configuration_checks.ConfigurationChecks` abgeleitet wurde. Alle Methoden, deren Namen mit `test_` anfangen, werden in alphanumerischer Reihenfolge ausgeführt. Beispiel (zu speichern in `/usr/share/ucs-school-import/checks/mychecks.py`):

```
from ucsschool.importer.exceptions import InitialisationError
```

*Prüfung der Konfiguration*

```
from ucsschool.importer.utils.configuration_checks import
    ConfigurationChecks

class MyConfigurationChecks(ConfigurationChecks):
    def test_nonzero_deactivation_grace(self):
        if self.config.get('deletion_grace_period', {}).get('deactivation', 0)
            == 0:
            raise InitialisationError('Value of
            "deletion_grace_period:deactivation" must not be zero.')
```



# Literaturverzeichnis

[ucs-handbuch] Univention GmbH. 2019. *Univention Corporate Server - Handbuch für Benutzer und Administratoren*.  
<https://docs.software-univention.de/handbuch-4.4.html>.

