sr solutions ag
Farbweg 9
CH-3400 Burgdorf

https://sr.solutions
info@sr.solutions

Synchronisationen mit HUB2

Dokumentation

Datum 26.11.21
Version 37
Autor:in Fabian Schmid

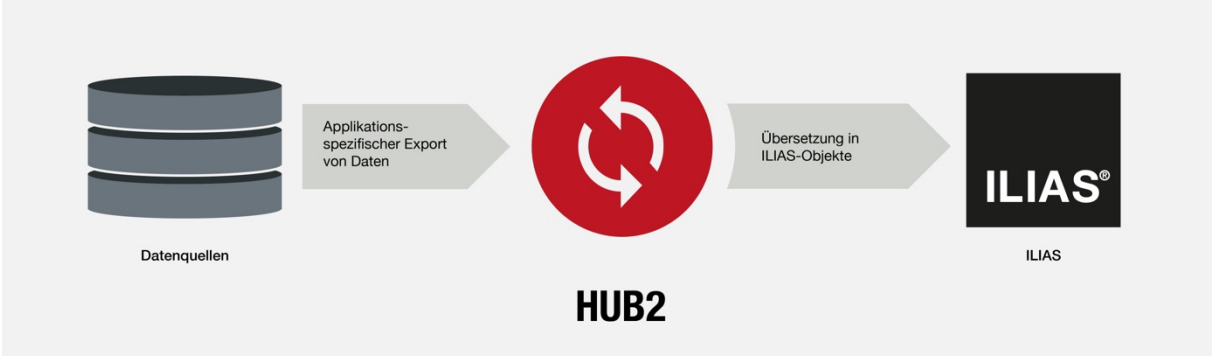
1 Inhalt

2 Einleitung

Viele Institutionen pflegen bereits Benutzerdaten oder Veranstaltungsanmeldungen in externen Systemen und möchten diese mit ILIAS abgleichen. ILIAS selber bietet bspw. für die Authentifizierung bereits eine Vielzahl von Standardlösungen wie ActiveDirectory, openID oder Shibboleth. Daneben können über die bestehende SOAP-Schnittstelle bspw. auch Kurse angelegt und aktualisiert werden, dies aber oft mit grösserem Aufwand für die Institution, welche die nötigen Abläufe und Scripts dafür selber implementieren muss.

Mit HUB2 bietet die sr.solutions eine einfache Lösung zur Einwegsynchronisation von an:

- Benutzeraccounts
- Kategorien
- Kurse und Gruppen
- Mitgliedschaften
- Organisationseinheiten und Positionen



HUB2 ist als Middleware-Software zwischen beliebigen Drittsystemen und ILIAS zu verstehen. Die Synchronisation erfolgt in eine Richtung, aus Daten von Drittsystemen werden in ILIAS neue Objekte angelegt oder bestehende Objekte aktualisiert. Eine Zweiweg-Synchronisation ist derzeit nicht vorgesehen.

Standardmässig nutzt HUB2 eine Vollsynchronisation, das bedeutet das Drittsysteme den gesamten zu synchronisierenden Datenstand bereitstellen sollte. Dabei müssen Drittsysteme keinerlei Informationen über den Status von Datensätzen (bspw. ist ein Datensatz neu, aktualisiert oder gelöscht) liefern, HUB2 eruiert diese Informationen selber und löst je nach Status eines Datensatzes unterschiedliche Aktionen aus.

HUB2 ist als unter der GNU General Public License 3 erstellt und wird unter https://github.com/srsolutionsag/HUB2 veröffentlicht. Das Plugin bietet standalone keine direkte Funktionalität, sondern liefert lediglich das Framework für eigene Synchronisationen.

Kontakt:

Fabian Schmid, support@sr.solutions

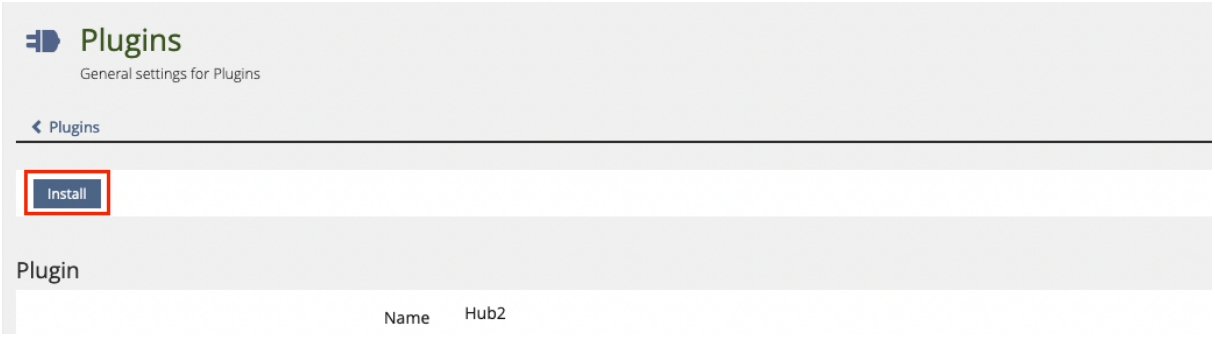
3 Einrichtung

3.1 Installation

Das HUB2-Plugin ist als ILIAS CronHook-Plugin verfasst und wird wie andere Plugins wie folgt auf dem Server installiert:

```
mkdir -p Customizing/global/plugins/Services/Cron/CronHook
cd Customizing/global/plugins/Services/Cron/CronHook
git clone https://github.com/studer-raimann/Hub2.git Hub2
```

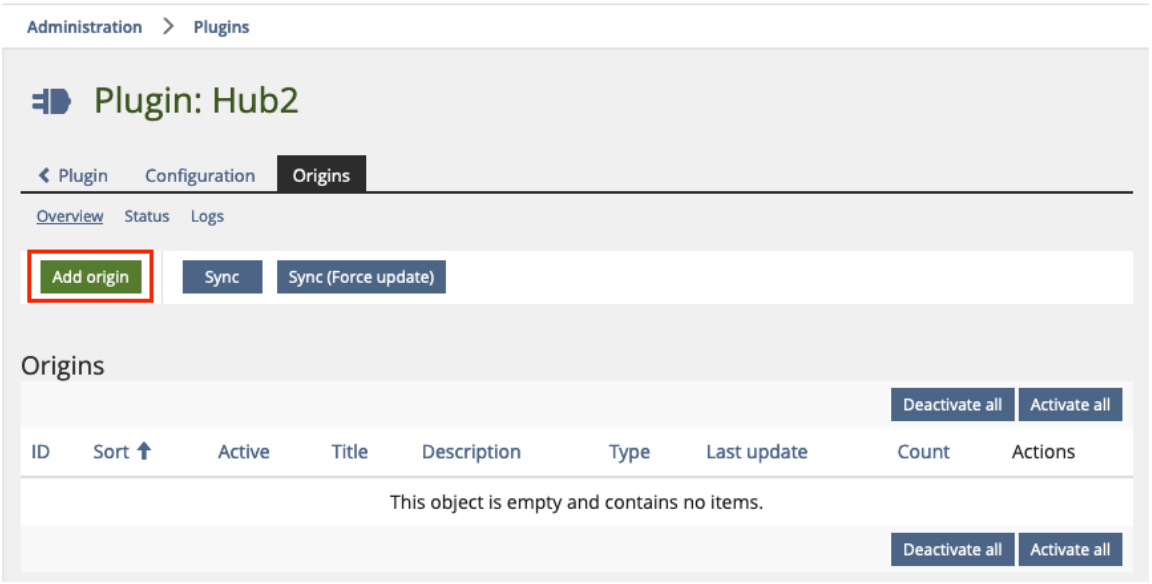
Anschliessend kann das Plugin in der ILIAS Plugin-Aministration installiert und aktiviert werden:

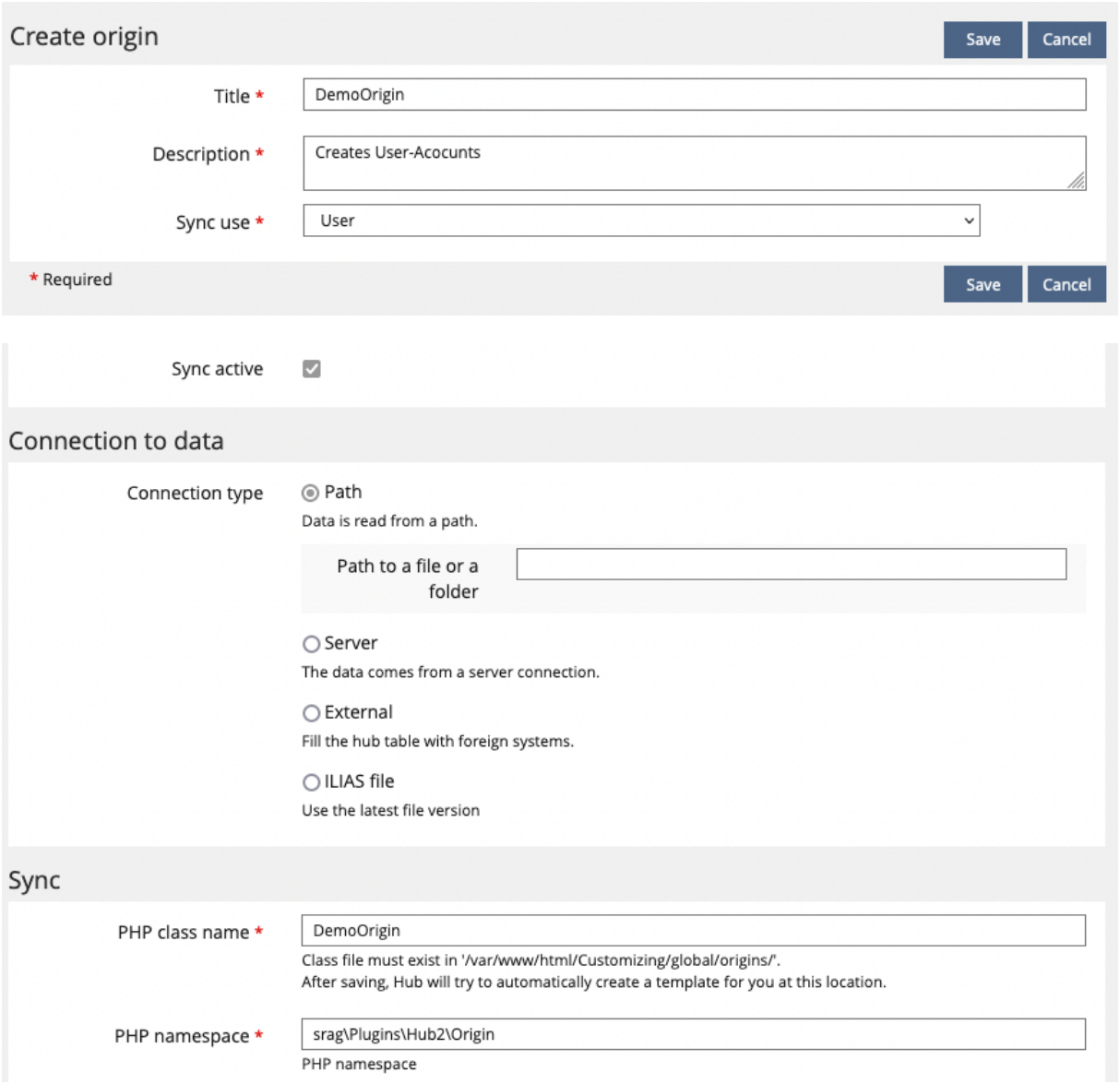


3.2 Anlegen von Anbindungen

Anbindungen (Origins) sind die Schnittstellen zu den Drittsystemen. Deren Aufgabe ist es, Daten von Drittsystemen in für HUB2 verarbeitbare Datenobjekte (DTOs) zu verwanden. In den meisten Fällen lesen solche Anbindungen Daten aus Export-Dateien, die die Drittsysteme bereitstellen. Anbindungen können aber bspw. auch Daten direkt von einem entfernten Dienst (bspw. Datenbank) auslesen und die Daten in DTOs konvertieren.

Alle Anbindungen bestehen aus einer Code-Komponente (Implementierung) und eine Konfiguration in der Benutzeroberfläche von HUB2. Über die Konfiguration kann bspw. festgelegt werden, welche Datenfelder in den ILIAS-Objekten aktualisiert werden, wenn eine Veränderung in den Daten eines einzelnen Datensatzes gefunden wird.




Create origin
Save
Cancel
Title *
DemoOrigin
Description *
Creates User-Acocunts
Sync use *
User
* Required
Save
Cancel
Sync active
Connection to data
Connection type
Path
Data is read from a path.
Path to a file or a folder
Server
The data comes from a server connection.
External
Fill the hub table with foreign systems.
ILIAS file
Use the latest file version
Sync
PHP class name *
DemoOrigin
Class file must exist in '/var/www/html/Customizing/global/origins/'.
After saving, Hub will try to automatically create a template for you at this location.
PHP namespace *
srag\Plugins\Hub2\Origin
PHP namespace

3.3 Implementierung

Eine Beispielimplementierung sieht wie folgt aus:

```
class DemoOrigin extends AbstractOriginImplementation
{

    public function connect() : bool
    {
        return is_readable($this->config()->getPath());
    }

    public function parseData() : int
    {
        $csv = new Csv(
            $this->config()->getPath(),
            'UserUniqueId',
            ['UserUniqueId', 'FirstName', 'LastName', 'Mail'],
        );
        $this->data = $csv->parseData();
        return count($this->data);
    }

    public function buildObjects() : array
    {
        $items = [];
        foreach ($this->data as $user_data) {
            $items[] = $this->factory()->user($user_data['UserUniqueId'])
                            ->setFirstname($user_data['FirstName'])
                            ->setLastname($user_data['LastName'])
                            ->setEmail($user_data['Mail']);
        }
        return $items;
    }
 // […]
}
```

Die drei relevantesten Methoden einer Anbindung sind

- **connect:** Prüft, ob eine Verbindung zur Datenquelle möglich ist, bspw. ob der konfigurierte Pfad zu einer Exportdatei lesbar ist
- **parseData:** Liest die Datenquelle ein und speichert die Rohdaten im Speicher zur späteren Weiterverarbeitung. Die Methode liefert die Anzahl eingelesener Datensätzen des Quellsystems
- **buildObjects:** Erstellt aus den Rohdaten entsprechende DTOs (im Beispiel «user»). Über eine eindeutige ID eines Datensatzes wird später die Verbindung zu einem ILIAS-Objekt (Benutzeraccount) hergestellt.

3.4 Beispieldaten für Exporte aus Drittsystemen

HUB2 setzt voraus, dass jeder Datensatz eine eindeutige ID besitzt (diese kann nummerisch oder als Zeichenkette vorliegen). Neben dieser eindeutigen ID sollten Datensätze weitere für die Erstellung von ILIAS-Objekten wichtigen oder erforderlichen Attribute besitzen. Grundsätzlich kann eine HUB-Anbindung-Implementierung beliebige Datenformate (JSON, XML, CSV, …) einlesen, wir bevorzugen aber JSON oder XML.

Nachfolgend einige Beispiele, wie Datenexporte aus Drittsystemen aufgebaut sein können.

3.4.1 Benutzeraccounts

```
{ "Users" : [
        {
                "UserUniqueId":"gf855698",
                "Login":"vorname1.nachname1",
                "Salutation" : "None",
                "FirstName":"Vorname1",
                "LastName":"Nachname1",
                "Mail":"vorname1.nachname1@example.com",
                "Role":"Admin"
        },
        {
                "UserUniqueId":"zh788541",
                "Login":"vorname2.nachname2",
                "Salutation" : "Ms",
                "FirstName":"Vorname2",
                "LastName":"Nachname2",
                "Mail":"vorname2.nachname2@example.com",
                "Role":"User"
        },
        {
                "UserUniqueId":"kh2369852",
                "Login":"vorname3.nachname3",
                "Salutation" : "Mr",
                "FirstName":"Vorname3",
                "LastName":"Nachname3",
                "Mail":"vorname3.nachname3@example.com",
                "Role":"User"
        },
        {
                "UserUniqueId":"ar4821530",
                "Login":"vorname4.nachname4",
                "Salutation" : "None",
                "FirstName":"Vorname4",
                "LastName":"Nachname4",
                "Mail":"vorname4.nachname4@example.com",
                "Role":"User"
        },

…

]}
```

3.5 Kurse


```
{ "Courses" : [
        {
                "CourseTitle" : "Onboarding 2022",
                "CourseExtId" : "bhuzfdd285",
                "CategoryLv3Title" : "Kategorie D1 auf Ebene 3 (innerhalb Kategorie C1)",
                "CategoryLv3ExtId" : 4,
                "CategoryLv2Title" : "Kategorie C1 auf Ebene 2 (innerhalb Kategorie B1)",
                "CategoryLv2ExtId" : 3,
                "CategoryLv1Title" : "Kategorie B1 auf Ebene 1 (innerhalb Kategorie A1)",
                "CategoryLv1ExtId" : 2,
                "CategoryLv0Title" : "Kategorie A1 auf Ebene 0 (innerhalb Repository)",
                "CategoryLv0ExtId" : 1
        },
        {
                "CourseTitle" : "Brandschutz März 2023",
                "CourseExtId" : "fkdhiln845",
                "CategoryLv3Title" : "Kategorie D2 auf Ebene 3 (innerhalb Kategorie C2)",
                "CategoryLv3ExtId" : 8,
                "CategoryLv2Title" : "Kategorie C2 auf Ebene 2 (innerhalb Kategorie B2)",
                "CategoryLv2ExtId" : 7,
                "CategoryLv1Title" : "Kategorie B2 auf Ebene 1 (innerhalb Kategorie A2)",
                "CategoryLv1ExtId" : 6,
                "CategoryLv0Title" : "Kategorie A2 auf Ebene 0 (innerhalb Repository)",
                "CategoryLv0ExtId" : 5
        },
        {
                "CourseTitle" : "Datensicherheit April 2023",
                "CourseExtId" : "jsldukn784",
                "CategoryLv3Title" : "Kategorie D3 auf Ebene 3 (innerhalb Kategorie C3)",
                "CategoryLv3ExtId" : 12,
                "CategoryLv2Title" : "Kategorie C3 auf Ebene 2 (innerhalb Kategorie B3)",
                "CategoryLv2ExtId" : 11,
                "CategoryLv1Title" : "Kategorie B3 auf Ebene 1 (innerhalb Kategorie A3)",
                "CategoryLv1ExtId" : 10,
                "CategoryLv0Title" : "Kategorie A3 auf Ebene 0 (innerhalb Repository)",
                "CategoryLv0ExtId" : 9
        }
]}
```


3.6 Kursmitgliedschaften


```
{ "CourseMemberships" : [
        {
                "UserExtId" : "ar4821530",
                "CourseExtId" : "fkdhiln845",
                "Role" : "CourseAdmin"
        },
        {
                "UserExtId" : "kh2369852",
                "CourseExtId" : "bhuzfdd285",
                "Role" : "CourseTutor"
        },
        {
                "UserExtId" : "kh2369852",
                "CourseExtId" : "jsldukn784",
                "Role" : "CourseMember"
        },
        {
                "UserExtId" : "gf855698",
                "CourseExtId" : "jsldukn784",
                "Role" : "CourseMember"
        },

…

]}
```

4 Funktionsweise

4.1 Statusänderungen in Datensätzen erkennen

HUB2 ermittelt in jedem Synchronisationsdurchgang den Status jedes einzelnen durch eine Anbindung gelieferten DTOs. Dabei erkennt HUB2 neue Datensätze (die eindeutige ID eines Datensatzes ist HUB2 bisher unbekannt), weiterhin vorhandene Datensätzen (die eindeutige ID eines Datensatzes war beim letzten Synchronisationsdurchgang vorhanden und ist es auch beim derzeitigen Synchronisationsvorgang) und gelöschte Datensätze (ein Datensatz mit einer bestimmten eindeutigen ID wurde beim letzten Durchgang gelidert, beim jetzigen Durchgang nicht mehr). Damit finde eine generelle Unterscheidung des Status jedes einzelnen Datensatzes (DTO) statt:

- **NEW:** Der Datensatz ist neu und wird erstmalig verarbeitet
- **UPDATED:** Der Datensatz wurde als bereits bekannt erkannt
- **OUTDATED:** Der Datensatz wurde im aktuellen Durchgang nicht mehr vorgefunden

Darüber hinaus erkennt HUB2 auch Datensätze, die bereits seit mehreren Vorgängen nicht mehr geliefert wurde (und damit zwischenzeitlich als gelöscht galten), beim aktuellen Durchgang aber wieder geliefert werden: **TO_RESTORE**

Aus den initialen Status wird vor der Verarbeitung eines DTOs zu einem ILIAS-Objekt ein Zwischenstatus ermittelt:

NEW	TO_CREATE	In der Verarbeitung des Datensatzes wird versucht, das entsprechende ILIAS-Objekt für den Datensatz anzulegen. Davor wird über eine allfällige MappingStrategy (siehe 4.3) versucht, ein entsprechendes bereits existierendes ILIAS-Objekt zu finden, in diesem Fall wechselt der Status zu TO_UPDATE
UPDATED	IGNORED	Sofern sich an den gelieferten Daten in einem DTO keine Veränderung gegenüber der im letzten Durchgang gelieferten Daten nicht verändert (siehe 4.2), wird dieser Datensatz ignoriert und nicht weiter verarbeitet (ausser bei Forced-Sync, siehe 4.6)
	TO_UPDATE	Das verlinkte ILIAS-Objekt zu dem Datensatz (DTO) wird nachfolgenden verarbeitet, dabei unter Berücksichtigung der konfigurierten Felder, siehe 0
	TO_RESTORE	Der Datensatz wurde mind. einmal als OUTDATED verarbeitet, daher existiert das verlinkte ILIAS-Objekt ggf. nicht mehr. Es wird entweder als TO_CREATE oder als TO_UPDATE weiter verarbeitet.
OUTDATED	TO_OUTDATED	Das verlinkte ILIAS-Objekt wird anhand der konfigurierten Aktion gelöscht, umbenannt oder verschoben, siehe 4.5.

Nach der Verarbeitung erhalten die Datensätze dann wiederum einen finalen Status, welcher so in der Datenbank gespeichert wird:

- CREATED
- UPDATED
- IGNORED
- OUTDATED

4.2 Ignorierte Datensätze

Um die Performance der Verarbeitung der Datensätze (DTOs) zu steigert, lösen nur veränderte DTOs eine Aktualisierung des verlinkten ILIAS-Objektes aus.

Bspw. wird in einem Durchgang der folgende Datensatz eines Benutzeraccounts geliefert:

```
        {
                "UserUniqueId":"gf855698",
                "Login":"vorname1.nachname1",
                "Salutation" : "None",
                "FirstName":"Vorname1",
                "LastName":"Nachname1",
                "Mail":"vorname1.nachname1@example.com",
                "Role":"Admin"
        },
```

Daraus wird in der Anbindung ein DTO wie folgt befüllt:

```
UserDTO {
        "ext_id" => "gf855698",
        "firstName" => "Vorname1",
        "lastName" => "Nachname1",
        "mail" => " vorname1.nachname1@example.com"
}
```

Nach der Verarbeitung des DTOs hin zu einem ILIAS-Objekt speichert sich HUB2 einen Hash der gelieferten Daten in der Datenbank, bspw.

```
md5(gf855698+Vorname1+Nachname1+vorname1.nachname1@example.com) ➔ a847fc5bed71a36ffee130283ba3b1bb
```

Beim nächsten Durchlauf wird aus Basis der gelieferten Daten genau dasselbe DTO zusammengestellt. HUB2 vergleicht den Hash der gelieferten Daten (DTO) mit dem in der Datenbank gespeicherten Hash. Stimmen diese Überein, hat sich kein Attribut des DTOs gegenüber dem letzten Durchlauf verändert, der Status wechselt von UPDATED zu IGNORED und der Datensatz wird nicht weiter verarbeitet (ausser bei Forced-Sync, siehe 4.6).

Ändert sich lediglich eines der Attribute, hat das DTO einen anderen Hash und wird dadurch weiterverarbeitet, bspw. ändert sich der Datensatz wie folgt:

```
        {
                "UserUniqueId":"gf855698",
                "Login":"vorname1.nachname1",
                "Salutation" : "None",
                "FirstName":"Vorname2",
                "LastName":"Nachname1",
                "Mail":"vorname2.nachname1@example.com",
                "Role":"Admin"
        },
```

Daraus wird in der Anbindung ein DTO wie folgt befüllt:

```
UserDTO {
        "ext_id" => "gf855698",
        "firstName" => "Vorname2",
        "lastName" => "Nachname1",
        "mail" => " vorname2.nachname1@example.com"
}
```

Nach der Verarbeitung des DTOs hin zu einem ILIAS-Objekt speichert sich HUB2 einen Hash der gelieferten Daten in der Datenbank, bspw.

```
md5(gf855698+Vorname2+Nachname1+vorname2.nachname1@example.com) ➔ 657621c3f60e14a1baca6a9350c6a29a
```

Da der gespeicherte Hash (a847fc5bed71a36ffee130283ba3b1bb) nicht mit dem neuen Hash (657621c3f60e14a1baca6a9350c6a29a) übereinstimmt, wechselt der Status von UPDATED zu TO_UPDATE und das entsprechende ILIAS-Objekt wird gemäss Konfiguration (siehe 4.4) aktualisiert.

4.3 Mapping bestehendes ILIAS-Objekte

Standardmässig erstellt HUB2 für DTOs im Status NEW entsprechende neue ILIAS-Objekte (bspw. Benutzeraccount). In einigen Fällen möchten aber bereits bestehende ILIAS-Objekte «übernommen» werden, weil bspw. bereits von einer früheren LDAP-Synchronisation entsprechende Accounts bestehen, diese aber neu durch HUB2 synchronisiert werden soll. Für diesen Fall kann einem DTO in der Anbindung eine «MappingStrategy» mitgegeben werden. HUB2 prüft in diesem Fall vor der Verarbeitung eines DTOs im Status TO_CREATE, ob die MappingStrategy ein passendes ILIAS-Objekt liefern kann. Ist dies der Fall, wechselt der Status zu TO_UPDATE und HUB2 aktualisiert das bestehende verlinkte ILIAS-Objekt.

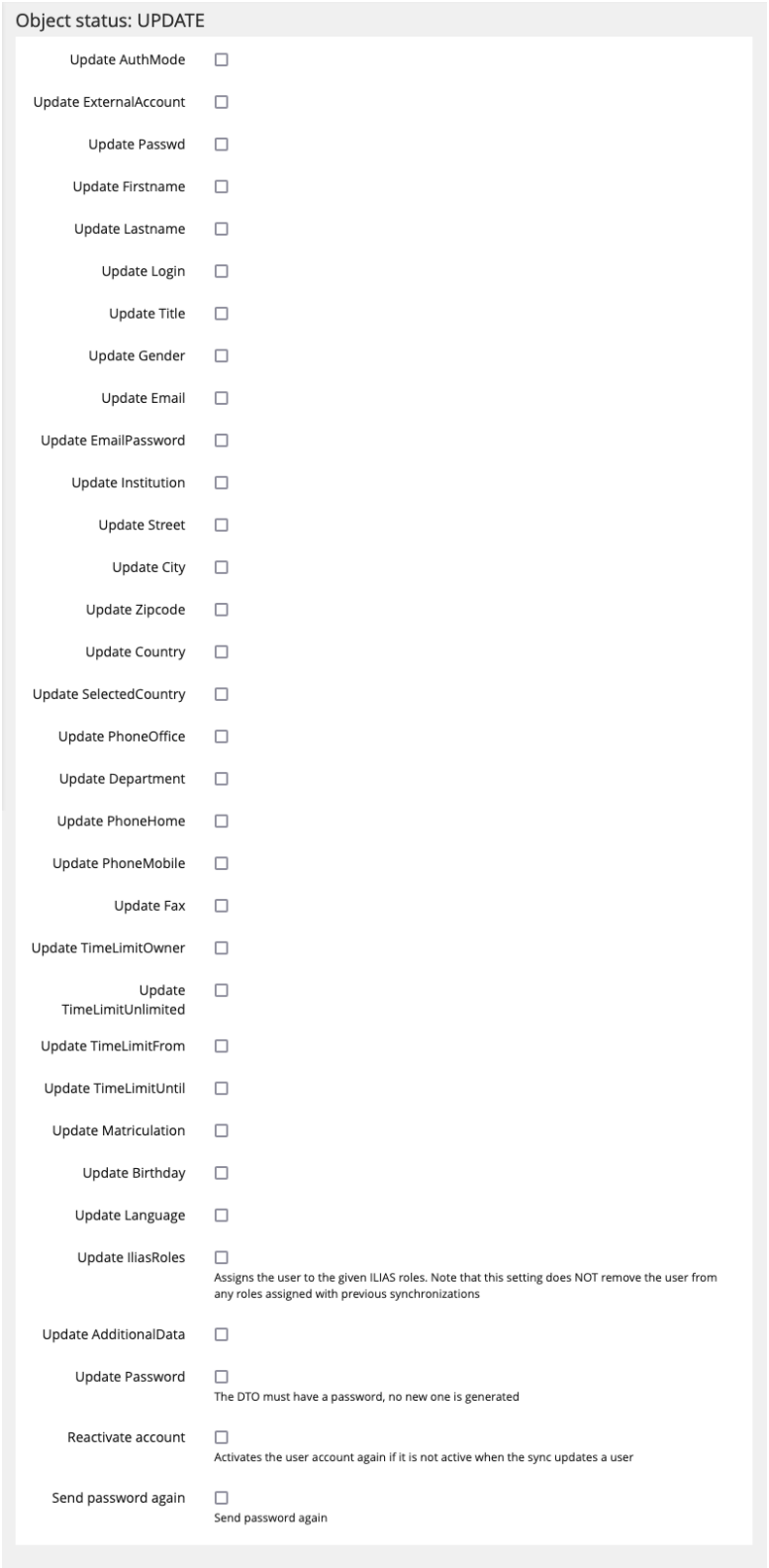
Im Code von HUB2 liegen bereits einige Standard-MappingStrategies bereit:

- ByEmail
- ByExternalAccount
- ByImportId
- ByLogin
- ByTitle
- FromHubToHub2

Eigene MappingStrategies können implementiert und ebenfalls mitgeliefert werden.

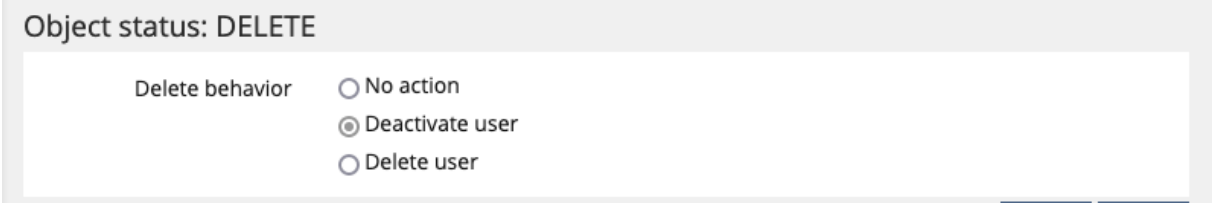
4.4 Konfiguration zu aktualisierten Datensätzen

Über die Bearbeitungsansicht einer Anbindung kann festgelegt werden, welche Attribute eine ILIAS-Objektes aktualisiert werden sollen, wenn ein DTO im Status TO_UPDATE verarbeitet wird. Es stehen pro Objekttyp (Benutzeraccount, Kurs, …) eine Auswahl von ILIAS-Attributen oder Einstellungen zur Verfügung.



4.5 Konfiguration zu gelöschten Datensätzen

Objekttyp (Benutzeraccount, Kurs, …) kann in der Konfiguration einer Anbindung festgelegt werden, was mit verlinkten ILIAS-Objekten geschehen soll, wenn ein DTO im Status TO_OUTDATED verarbeitet wird.



4.6 Forced-Sync

Eine Synchronisierung kann als Forced-Sync durchgeführt werden, dafür gibt es zwei Varianten:

- Benutzeroberfläche: Auslösen über **«Sync (Force update)»**
- CLI: mit Umgebungsvariable **«HUB2_FORCED_SYNC=true»**

Bei einer Forced-Sync wird auf das Vergleichen von Hashes verzichtet (siehe 4.2) und alle als UPDATED gelieferten DTOs werden als TO_UPDATE verarbeitet und gem. Konfiguration aktualisiert (siehe 4.4).

Generell wird von dem Ausführen von Synchronisationen über die Benutzeroberfläche abgeraten, da diese durch Speicher-Limitation oder Beschränkungen der Laufzeit eines Prozesses abgebrochen werden können.