univention
be open.

UCS@school Kelvin REST API Documentation

Release 1.8.3

Univention GmbH

Jan 30, 2023

CONTENTS:

1 Overview **1**
- 1.1 Components . . . 1

2 Installation and configuration **3**
- 2.1 Installation . . . 3
- 2.2 Configuration . . . 3
- 2.3 File locations . . . 7

3 Authentication and authorization **9**
- 3.1 Authentication . . . 9
- 3.2 Authorization . . . 10

4 OpenAPI / Swagger / ReDoc **11**
- 4.1 Interactive API browser . . . 11
- 4.2 Generating client code from OpenAPI schema . . . 11

5 Resources **13**
- 5.1 Resource Roles . . . 13
- 5.2 Resource Schools . . . 15
- 5.3 Resource Users . . . 19
- 5.4 Resource Classes . . . 29
- 5.5 Resource Workgroups . . . 34

6 Known Issues **41**
- 6.1 Rebuilding the UDM REST API Client . . . 41
- 6.2 No pagination . . . 41
- 6.3 Creating schools does not work in single server environments . . . 42

7 Python Client Compatibility **43**

8 Bibliography **45**

Bibliography **47**

Index **49**

CHAPTER

ONE

OVERVIEW

The *UCS@school Kelvin REST API* provides HTTP endpoints to create and manage UCS@school domain objects like school users, school classes, schools (OUs) and computer rooms.

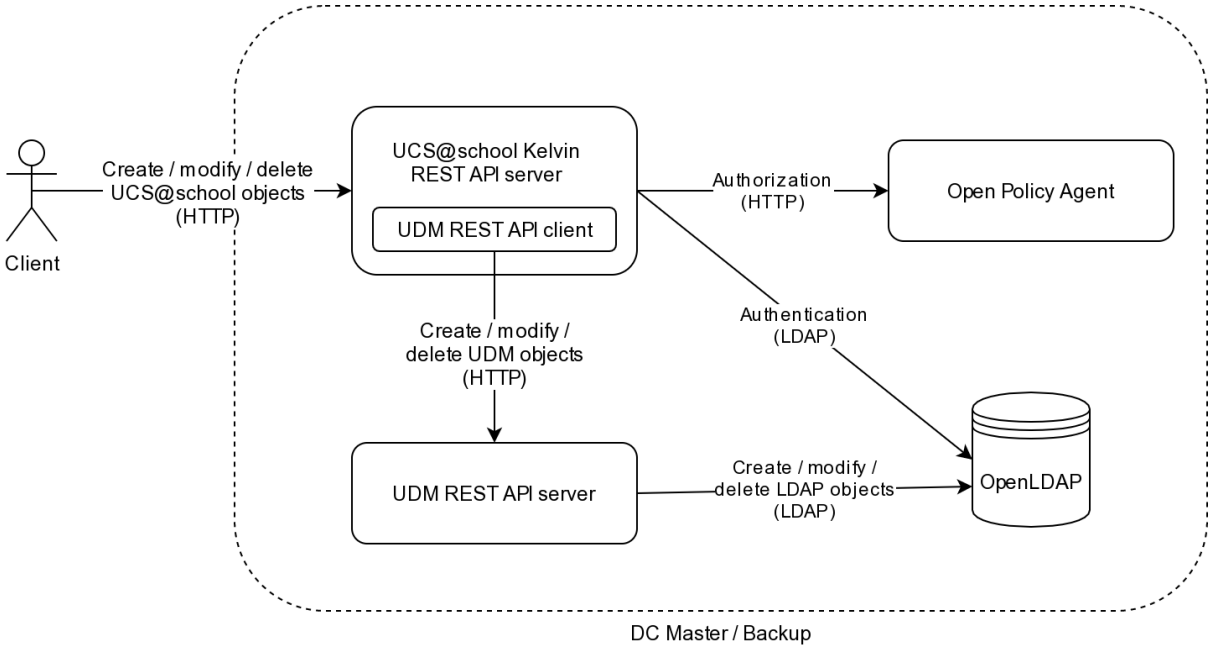
The same objects can be managed through a graphical web interface (UMC modules) and a Python API.

The components participating in the *UCS@school Kelvin REST API* are:

- clients of the *UCS@school Kelvin REST API* (for example the Python client)
- the *UCS@school Kelvin REST API* server on the DC master / DC backup[1]
- the Open Policy Agent used for authorization
- the UDM REST API client component of the *UCS@school Kelvin REST API*
- the UDM REST API server on the DC master
- the LDAP server on the DC master

1.1 Components

To be more precise, let's take a look at the topology picture, starting with the client:



[1] The app *UCS@school Kelvin REST API* can be installed on both DC master and DC backup. The rest of the text will talk only about *DC master*, but the same is valid for *DC backup*.

1. Clients are HTTP clients sending requests to `https://<fqdn>/ucsschool/kelvin/v1/<resource>/` with `<fqdn>` being the address of the DC master and `<resource>` being the object type (users, classes etc) to manage.
2. The *UCS@school Kelvin REST API* can only be accessed through HTTPS on the DC master or DC backup after installing an app (see installation and configuration instructions). The web server Apache runs a reverse proxy for the URL path `/ucsschool/kelvin/`. All access to that path is forwarded to the *UCS@school Kelvin REST API* server running *inside a Docker container*.
3. To manage the UCS@school resources, a token must first be retrieved at `https://<fqdn>/ucsschool/kelvin/token/`. The server will access the OpenLDAP server directly to authenticate and authorize the connecting client. If both succeed, a temporary token is issued to the client.
4. When the client requests a resource (including a valid token in the transmission), the *UCS@school Kelvin REST API* server will use its *UDM REST API client* component to access the UDM REST API on the DC master.
5. The UDM REST API will then query the OpenLDAP server to retrieve the required information or make the requested changes.

CHAPTER

TWO

INSTALLATION AND CONFIGURATION

2.1 Installation

The app *UCS@school Kelvin REST API* must be installed on the DC master or DC backup. This can be done either through the UMC module *Univention App Center* or on the command line:

```
$ univention-app install ucsschool-kelvin-rest-api
```

The join script `50ucsschool-kelvin-rest-api.inst` should run automatically. To verify if it succeeded, open the *Domain join* UMC module or run:

```
$ univention-check-join-status
```

If it hasn't run, start it in the UMC module or execute:

```
$ univention-run-join-scripts
```

If problems occur during installation or join script execution, relevant log files are:

1. `/var/log/univention/appcenter.log`
2. `/var/log/univention/join.log`

2.2 Configuration

The *UCS@school Kelvin REST API* can be used out of the box, but there are various parameters that can be configured.

Hint: `univention-app configure --list ucsschool-kelvin-rest-api` lists all available app settings and their current values. *Hint*: App Settings can be changed using the command line with `univention-app configure ucsschool-kelvin-rest-api --set ucsschool/kelvin/log_level=DEBUG && univention-app restart ucsschool-kelvin-rest-api`

2.2.1 Number of cores

The number of CPU cores used by the *UCS@school Kelvin REST API* app can be configured. The default is `2`. Values below `1` start one process for each available CPU. The value can be changed through the *app settings* of the *UCS@school Kelvin REST API* app in the *Univention App Center* UMC module.

2.2.2 Token validity

All HTTP requests to resources must carry a valid JWT token. The number of minutes a token is valid can be configured. The default is `60`. The value can be changed through the *app settings* of the *UCS@school Kelvin REST API* app in the *Univention App Center* UMC module.

2.2.3 Custom CA Certificates

By default, the *UCS@school Kelvin REST API* only connects to an LDAP server which is using the CA provided by UCS. If the LDAP server uses a different CA, that CA needs to be configured through the *app settings* in the *Univention App Center* UMC module.

2.2.4 Log level

The minimum severity for log messages written to `/var/log/univention/ucsschool-kelvin-rest-api/http.log` can be configured. The default is `INFO`. The value can be changed through the *app settings* of the *UCS@school Kelvin REST API* app in the *Univention App Center* UMC module.

2.2.5 Backup count of validation logging

The UCR variable `ucsschool/validation/logging/backupcount` sets the amount of copies of the log file `ucs-school-validation.log`, which should be kept in rotation. The default is `60`. The host's UCR-V is copied into the Docker container during the join script. To change it for the *UCS@school Kelvin REST API*, it has to be modified inside the Docker container.

2.2.6 Configuration of user object management (import configuration)

The directory `/var/lib/ucs-school-import/configs` is mounted as a *volume* into the Docker container where the *UCS@school Kelvin REST API* runs. This makes it accessible from the host as well as from inside the container.

The directory contains the file `kelvin.json`, which is the top level configuration file for the UCS@school import code, executed when `user` objects are managed. Documentation for the UCS@school import configuration is available only in German in *UCS@school - Handbuch zur CLI-Import Schnittstelle* [1].

2.2.7 UDM Properties

There was already an `udm_properties` functionality available for user resources within Kelvin. With the release of Kelvin 1.5.0 the `udm_properties` functionality was added to all other resources (except roles) as well. The list of `mapped_udm_properties` can be configured in `/etc/ucsschool/kelvin/mapped_udm_properties.json`.

The format of the `mapped_udm_properties.json` is:

```
{
    name_of_resource: ["name_of_property_to_map",...],
    ...
}
```

For example:

```
{
    "user": ["unixhome", "title"],
    "school_class": ["mailAddress"],
    "school": ["description"]
}
```

The following restrictions have to be observed:

1. The Kelvin configuration may contain also a `mapped_udm_properties`. This refers to the user resource. If there is also a configuration for the key `user` in `mapped_udm_properties.json`, it will override the `mapped_udm_propertes` kelvin configuration (for users only).
2. Any udm property that is directly linked to an already existing model field results in an invalid configuration. It is not allowed, for example, to configure the `description` of a school class as an udm property, since it is already present in the model itself. This is now also true for the user resource, where this was possible before.

Important Please be advised that this direct access to udm properties is in no way checked or validated by any UCS@school logic and thus can lead to corrupt objects and errors on your system, if not used correctly.

2.2.8 Python hooks for user object management (import hooks)

Read next chapter about hooks for non-user objects like school classes.

The directory `/var/lib/ucs-school-import/kelvin-hooks` is mounted as a *volume* into the Docker container, so it can be accessed from the host. The directory content is scanned when the Kelvin API server starts. If it contains classes that inherit from `ucsschool.importer.utils.import_pyhook.ImportPyHook`, they are executed when users are managed through the Kelvin API. The hooks are very similar to the Python hooks for the UCS@school import (see *UCS@school - Handbuch zur CLI-Import Schnittstelle* [1]). The differences are:

- Python 3.7 only
- Only three types of hooks are executed: `UserPyHook`, `FormatPyHook` and `ConfigPyHook` (all located in modules in the `ucsschool.importer.utils` package).
- `self.dry_run` is always `False`
- `self.lo` is always a LDAP connection with write permissions (`cn=admin`) as `dry_run` is always `False`
- `FormatPyHook` and `ConfigPyHook` are the same as in the UCS@school import, but a `UserPyHook` hook instance has an additional member `self.udm`.

`self.udm` is an instance of `udm_rest_client.udm.UDM` (see Python UDM REST Client). It can be used to comfortably query the UDM REST API running on the DC master. When using the UCS@school lib or import, it must be used in most places that `self.lo` was used before.

Important: When calling methods of *ucsschool* objects (e.g. `ImportUser`, `SchoolClass` etc.) `self.udm` must be used instead of `self.lo` and those methods may have to be used with `await`. Thus hooks methods will be `async`. For example:

```
async def post_create(self, user: ImportUser) -> None:
    user.firstname = "Sam"
    awaituser.modify(self.udm)

    udm_user_obj = await user.get_udm_object(self.udm)
    udm_user_obj["foo"] = "bar"
    await udm_user_obj.save()  # UDM REST Client object: "save", not "modify"
```

2.2.9 Python hooks for pre- and post-object-modification actions

Read previous chapter about hooks for user objects.

Since version `1.4.2` of the *UCS@school Kelvin REST API* app it is possible to execute custom Python code before and after the creation, modification, moving or deletion of any UCS@school object.

To use the hook functionality a Python class deriving from `ucsschool.lib.models.hook.Hook` (https://github.com/…/hook.py) must be created.

In the class methods `pre_create()`, `post_create()`, `pre_modify()` and so on can be implemented. They will be executed at the specified time.

The Python module with the hook class must be stored in the directory `/var/lib/ucs-school-lib/kelvin-hooks`. Please note that it is a different directory than the one from the previous chapter.

Two examples can be found at https://github.com/…/hook_example1.py and https://github.com/…/hook_example2.py.

The API for those hooks is almost identical to the one described in *Python hooks for user object management (import hooks)*. The main differences are that the attribute `self.dry_run` does not exist, a UCR instance is available in `self.ucr` and the class attribute `model`.

The class attribute `model` is used to determine for objects of which classes (models) the hook should be executed. The hook will also be executed for subclasses of the one defined here. If for example `model = Teacher` (from module `ucsschool.lib.models.user`), the hooks methods would also be execute for objects of `TeachersAndStaff`, but not for those of type `Staff` or `Student` (as they are not derived from `Teacher`).

The class attribute `priority` defines the order in which methods of hooks for the same type (same `model`) are executed, or if they are deactivated. Methods with higher numbers are executed before those with lower numbers. If the value is `None` the method will not run.

The methods `pre_create()`, `post_modify()` and so on receive the object being modified and return `None`. The type of `obj` is the one in `model` (or a subclass).

To add custom initialization code, `__init__()` can be implemented the following way:

```
from ucsschool.lib.models.hook import Hook
# from udm_rest_client import UDM
# from univention.admin.uldap import LoType

class MailForSchoolClass(Hook):
    def __init__(self, udm: UDM, lo: LoType = None, *args, **kwargs) -> None:
        super(MailForSchoolClass, self).__init__(udm, lo, *args, **kwargs)
        # From here on self.lo, self.logger and self.ucr are available.
        # You code here.
```

To activate a hook, or or a change to a hook, restart the *UCS@school Kelvin REST API* Docker container:

```
$ /etc/init.d/docker-app-ucsschool-kelvin-rest-api restart
```

Further reading about the UCS@school hooks is available for German readers in Python-Hooks in *UCS@school - Handbuch für Administratoren* [2]. Please note that the example in that text is for the synchronous variant, missing the `async/await` keywords and not using the UDM REST API client. Compare with the examples linked in this chapter.

2.3 File locations

2.3.1 Log files

`/var/log/univention/ucsschool-kelvin-rest-api` is a volume mounted into the docker container, so it can be accessed from the host. The directory contains the file `http.log`, which is the log of the HTTP-API (both ASGI server and API application) and the file `ucs-school-validation.log`, which is used to write sensitive information during the UCS@school validation.

2.3.2 User object (import) configuration

`/var/lib/ucs-school-import/configs` is a volume mounted into the docker container, so it can be accessed from the host. The directory contains the file `kelvin.json`, which is the top level configuration file for the UCS@school import code that is executed as part of the *UCS@school Kelvin REST API* that runs inside the Docker container when user objects are managed.

2.3.3 Python hooks

`/var/lib/ucs-school-import/kelvin-hooks` and `/var/lib/ucs-school-lib/kelvin-hooks` are volumes mounted into the docker container, so they can be accessed from the host. Their purpose is explained above in chapters *Python hooks for user object management (import hooks)* and *Python hooks for pre- and post-object-modification actions*.

CHAPTER

THREE

AUTHENTICATION AND AUTHORIZATION

3.1 Authentication

To use the API, a JSON Web Token (JWT) must be retrieved from `https://<fqdn>/ucsschool/kelvin/token`. The token will be valid for a configurable amount of time (default 60 minutes), after which it must be renewed. To change the value see chapter *Token validity*.

The time a token is valid is stored inside the JWT token in the `exp` attribute.

Example `curl` command to retrieve a token:

```
$ curl -i -k -X POST https://<fqdn>/ucsschool/kelvin/token \
    -H "Content-Type:application/x-www-form-urlencoded" \
    -d "username=Administrator" \
    -d "password=s3cr3t"
```

The response headers will be:

```
HTTP/1.1 200 OK
Date: Mon, 20 Jan 2020 10:32:17 GMT
Server: uvicorn
content-length: 176
content-type: application/json
Via: 1.1 <fqdn>
```

The response body will be:

```
{
    "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUz...",
    "token_type": "bearer"
}
```

Hint: to get a JSON response pretty printed omit the `-i` in the `curl` command and pipe the response through a JSON formatter:

```
$ curl -k -X POST https://<fqdn>/ucsschool/kelvin/token \
    -H "Content-Type:application/x-www-form-urlencoded" \
    -d "username=Administrator" \
    -d "password=s3cr3t" | python -m json.tool
```

3.2 Authorization

Only members of the group `ucsschool-kelvin-rest-api-admins` are allowed to access the API.

The user `Administrator` is automatically added to this group for testing purposes. In production a regular admin user account or a dedicated service account should be used.

Irrespective of the actually authenticated user, all operations will be executed using the `cn=admin` LDAP account.

CHAPTER

FOUR

OPENAPI / SWAGGER / REDOC

4.1 Interactive API browser

The *UCS@school Kelvin REST API* offers two web interfaces to explore the API graphically:

- The Swagger UI at `https://<fqdn>/ucsschool/kelvin/v1/docs`
- The ReDoc UI at `https://<fqdn>/ucsschool/kelvin/v1/redoc`

Both UIs are created in the browser by JavaScript code using the OpenAPI specification generated by the *UCS@school Kelvin REST API* server.

The OpenAPI specification can be downloaded at: `https://<fqdn>/ucsschool/kelvin/v1/openapi.json`.

The Swagger UI allows direct interaction (reading, creating users etc.) with the API server. To use it, a token must first be retrieved by clicking the `Authorize` button in the top right corner. In the pop-up, the username must be of a user in the group `ucsschool-kelvin-rest-api-admins`. The user `Administrator` is added to this group by default. After supplying the password and clicking the `Authorize` button the pop-up can be closed.

Then methods of resources can be used. After opening on one, a click on `Try it out` on the right side will allow to `Execute` the request.

4.2 Generating client code from OpenAPI schema

Source code for *UCS@school Kelvin REST API* clients can be auto-generated by a number of tools. Popular open source tools that can generate source code from the servers `openapi.json` for over 50 programming languages are:

- OpenAPI Generator: https://github.com/OpenAPITools/openapi-generator
- Swagger Codegen: https://github.com/swagger-api/swagger-codegen

CHAPTER

FIVE

RESOURCES

Resources may support a varying range of operations: retrieve, search, create, modify, move and delete.

Pagination has not yet been implemented. When it is, there will be `<Link>` entries in the response headers. The format of the JSON response in the body will not change.

Requests to resource endpoints must carry a valid token. Section *Installation and configuration* describes how to obtain one. Sending no or an invalid token leads to the server responding with HTTP status `401`.

The token must be in the `Authorization` header with a value `Bearer <token>`. E.g.:

```
"Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9...."
```

When objects are loaded that are called by the **UDM Rest API**, the expected properties of the corresponding UCS@school object are checked and errors are logged. The output of the complete object as well as the stack trace are written to `ucs-school-validation.log`.

5.1 Resource Roles

The `Roles` resource represents the roles a school user can have. Currently there are exactly three roles supported: `staff`, `student` and `teacher`. A user has either one of those roles or the combination of `staff` and `teacher`.

The resource objects have no direct representation in LDAP. They are only required to classify user objects.

The item list of the `Roles` resource is hard coded. It does only support listing objects. It does not support creating, modifying or deleting roles.

5.1.1 Roles resource representation

The following JSON is an example Roles resource in the *UCS@school Kelvin REST API*:

```
{
    "display_name": "staff",
    "name": "staff",
    "url": "https://<fqdn>/ucsschool/kelvin/v1/roles/staff"
}
```

Table 1: Property description

name	value	Description	Notes
display_name	string	The name of the role (for views).	read only
name	string	The name of the role (technically).	read only
url	URL	The URL of the role object in the UCS@school Kelvin API.	read only

5.1.2 Roles list and search

Example `curl` command to retrieve the list of all roles:

```
$ curl -i -k -X GET "https://<fqdn>/ucsschool/kelvin/v1/roles/" \
    -H "accept: application/json"
    -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJh...."
```

The response headers will be:

```
HTTP/1.1 200 OK
Date: Mon, 20 Jan 2020 14:21:00 GMT
Server: uvicorn
content-length: 334
content-type: application/json
Via: 1.1 <fqdn>
```

The response body will be:

```
[
    {
        "display_name": "staff",
        "name": "staff",
        "url": "https://<fqdn>/ucsschool/kelvin/v1/roles/staff"
    },
    {
        "display_name": "student",
        "name": "student",
        "url": "https://<fqdn>/ucsschool/kelvin/v1/roles/student"
    },
    {
        "display_name": "teacher",
        "name": "teacher",
        "url": "https://<fqdn>/ucsschool/kelvin/v1/roles/teacher"
    }
]
```

Searching for roles (with `?name=abc*` or similar) is *not* supported.

5.1.3 Roles retrieve

Example `curl` command to retrieve a single role:

```
$ curl -X GET "https://<fqdn>/ucsschool/kelvin/v1/roles/student"\
    -H "accept: application/json" \
    -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJh...."
```

The queried role must exist, matching is case-sensitive. The response body will be the second element of the list in the example above.

5.2 Resource Schools

The `Schools` resource is represented in the LDAP tree as an `OU`.

To list those LDAP objects run:

```
$ univention-ldapsearch -LLL "objectClass=ucsschoolOrganizationalUnit"
```

UCS@school uses the UDM REST API in *Univention Developer Reference* [3] which in turn uses UDM to access LDAP. UDM properties have different names than their associated LDAP attributes. Their values may also differ. To list the same UDM objects run:

```
$ udm container/ou list --filter "objectClass=ucsschoolOrganizationalUnit"
```

All UCS@school objects exist below an `OU` and have that OUs name as the `school` attributes value. Staff, students and teachers may attend or work at multiple schools. So `User` objects have an additional `schools` attribute, that is a list of all schools a user belongs to.

Currently the `Schools` resource does only support listing and creating objects. It does not yet support modifying or deleting OUs.

5.2.1 Schools resource representation

The following JSON is an example Schools resource in the *UCS@school Kelvin REST API*:

```
{
    "dn": "ou=test,dc=uni,dc=ven",
    "url": "https://<fqdn>/ucsschool/kelvin/v1/schools/test",
    "ucsschool_roles": ["school:school:test"],
    "name": "test",
    "display_name": "Test School",
    "educational_servers": ["dctest-01"],
    "administrative_servers": [],
    "class_share_file_server": "dctest-01",
    "home_share_file_server": "dctest-01",
    "udm_properties": {}
}
```

Table 2: Property description

name	value	Description	Notes
dn	string	The DN of the OU in LDAP.	read only
url	URL	The URL of the role object in the UCS@school Kelvin API.	read only
ucsschool_roles	list	List of roles the OU has. Format is `ROLE:CONTEXT_TYPE:CONTEXT`, for example: `["school:school:gym1"]`.	auto-managed by system, setting and changing discouraged
name	string	The name of the school (technically: the name of the OU).	read only
display_name	string	The name of the school (for views).	
educational_servers	list	List of server host names for the educational school network.	(*)
administrative_servers	list	List of server host names for the administrative school network.	(*)
class_share_file_server	string	Host name of server with the class shares.	if unset: the schools educational server, (*)
home_share_file_server	string	Host name of server with the home shares.	if unset: the schools educational server, (*)
udm_properties	nested object	Object with UDM properties. For example: `{"description": "Gymnasium"}`	Must be configured, see below.

(*) **API CHANGE**: before version `1.4.0` this was a DN or list of DNs

5.2.2 Schools udm_properties

The attribute `udm_properties` is an object that can contain arbitrary UDM properties. It must be configured in the file `/etc/ucsschool/kelvin/mapped_udm_properties.json`, see *UDM Properties*.

Attention: Due to the technical way schools are created, udm_properties are set after the initial creation of the school. This can lead to a school being created with an error following the subsequent alteration. In this case the Kelvin API returns a 500 status code, but the school was created anyways.

5.2.3 Schools list and search

Example `curl` command to retrieve the list of all schools (OUs):

```
$ curl -i -k -X GET "https://<fqdn>/ucsschool/kelvin/v1/schools/" \
    -H "accept: application/json" \
    -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJh...."
```

The response headers will be:

```
HTTP/1.1 200 OK
Date: Mon, 20 Jan 2020 14:00:41 GMT
Server: uvicorn
content-length: 1957
content-type: application/json
Via: 1.1 <fqdn>
```

The response body will be:

```
[
    {
```

(continues on next page)

(continued from previous page)

```
        "dn": "ou=DEMOSCHOOL,dc=uni,dc=ven",
        "url": "https://<fqdn>/ucsschool/kelvin/v1/schools/DEMOSCHOOL",
        "name": "DEMOSCHOOL",
        "display_name": "Demo School",
        "educational_servers": ["dc-demoschool"],
        "administrative_servers": [],
        "class_share_file_server": "dc-demoschool",
        "home_share_file_server": "dc-demoschool",
        "udm_properties": {}
    }
]
```

To search for schools with a name that starts with `abc`, append `?name=abc*` to the school resource. The search is case-insensitive. The URL would be: `https://<fqdn>/ucsschool/kelvin/v1/schools/?name=abc%2A`

`name` is the only attribute that can be used to search for OUs.

5.2.4 Schools exist

Example `curl` command to check for the existence of a single school (OU):

```
$ curl -i --head "https://<fqdn>/ucsschool/kelvin/v1/schools/demoschool" \
    -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJh...."
```

The response headers will be:

```
HTTP/1.1 200 OK
Date: Tue, 13 Sep 2022 20:28:27 GMT
Server: uvicorn
x-request-id: fd07836e6564438287efe1f2de0772d8
access-control-expose-headers: X-Request-ID
Via: 1.1 <fqdn>
```

With the search being case-insensitive, this matches an OU named `DEMOSCHOOL`. The response body will be *empty*.

A response status code of `200` means, that the school object exists, `404` means that it does not.

5.2.5 Schools retrieve

Example `curl` command to retrieve a single school (OU):

```
$ curl -X GET "https://<fqdn>/ucsschool/kelvin/v1/schools/demoschool" \
    -H "accept: application/json" \
    -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJh...."
```

With the search being case-insensitive, this matches an OU named `DEMOSCHOOL`. The response body will be the first element of the list in the search example above.

5.2.6 Schools create

Since version `1.4.0` of the *UCS@school Kelvin REST API* app it is possible to create school objects (OUs).

When creating a school, two attributes must be set:

- `name`
- `display_name`

As an example, with the following being the content of `/tmp/create_ou.json`:

```
{
    "name": "example",
    "display_name": "Example School"
}
```

This `curl` command will create a school from the above data:

```
$ curl -i -k -X POST "https://<fqdn>/ucsschool/kelvin/v1/schools/" \
    -H "accept: application/json" \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJh...." \
    -d "$(</tmp/create_ou.json)"
```

Response headers:

```
HTTP/1.1 201 Created
Date: Mon, 26 Mar 2021 13:10:00 GMT
Server: uvicorn
content-length: 335
content-type: application/json
Via: 1.1 <fqdn>
```

Response body:

```
{
    "dn": "ou=Example,dc=uni,dc=ven",
    "url": "https://<fqdn>/ucsschool/kelvin/v1/schools/Example",
    "ucsschool_roles": ["school:school:Example"],
    "name": "Example",
    "display_name": "Example School",
    "educational_servers": ["dcExample"],
    "administrative_servers": [],
    "class_share_file_server": "dcExample",
    "home_share_file_server": "dcExample"
}
```

5.2.7 Schools modify and move

Not supported.

5.2.8 Schools delete

Not supported.

5.3 Resource Users

The `Users` resource is represented in the LDAP tree as user objects.

To list those LDAP objects run:

```
$ FILTER=
↪'(|(objectClass=ucsschoolStaff)(objectClass=ucsschoolStudent)(objectClass=ucsschoolTeacher))
↪'
$ univention-ldapsearch -LLL "$FILTER"
```

UCS@school uses the UDM REST API which in turn uses UDM to access LDAP. UDM properties have different names than their associated LDAP attributes. Their values may also differ. To list the same UDM objects run:

```
$ FILTER=
↪'(|(objectClass=ucsschoolStaff)(objectClass=ucsschoolStudent)(objectClass=ucsschoolTeacher))
↪'
$ udm users/user list --filter "$FILTER"
```

5.3.1 Users resource representation

The following JSON is an example User resource in the *UCS@school Kelvin REST API*:

```
{
    "dn": "uid=demo_student,cn=schueler,cn=users,ou=DEMOSCHOOL,dc=uni,dc=ven",
    "url": "https://<fqdn>/ucsschool/kelvin/v1/users/demo_student",
    "ucsschool_roles": ["student:school:DEMOSCHOOL", "student:school:DEMOSCHOOL2"],
    "name": "demo_student",
    "school": "https://<fqdn>/ucsschool/kelvin/v1/schools/DEMOSCHOOL",
    "firstname": "Demo",
    "lastname": "Student",
    "birthday": "2003-10-24",
    "disabled": false,
    "email": "demo_student@uni.ven",
    "expiration_date": "2030-02-14",
    "record_uid": "demo_student12",
    "roles": ["https://<fqdn>/ucsschool/kelvin/v1/roles/student"],
    "schools": [
        "https://<fqdn>/ucsschool/kelvin/v1/schools/DEMOSCHOOL",
        "https://<fqdn>/ucsschool/kelvin/v1/schools/DEMOSCHOOL2"
    ],
    "school_classes": {
        "DEMOSCHOOL": ["Democlass"],
        "DEMOSCHOOL2": ["demoklasse2"]
    },
    "workgroups": {
        "DEMOSCHOOL": ["demoworkgroup"],
        "DEMOSCHOOL2": ["demoworkgroup2"]
    },
    "source_uid": "Kelvin",
    "udm_properties": {
        "description": "An example user attending two school.",
        "gidNumber": 5023,
        "employeeType": null,
```

(continues on next page)

(continued from previous page)


```
        "organisation": null,
        "phone": ["+49 177 3578031", "+49 241 3456232"],
        "title": null,
        "uidNumber": 2007
    }
}
```

Table 3: Attribute description

name	value	Description	Notes
dn	string	The DN of the user in LDAP.	read only
name	string	The users username.	
url	URL	The URL of the user object in the UCS@school Kelvin API.	read only
firstname	string	The users given name.	
lastname	string	The users family name.	
birthday	date	The users birthday in ISO 8601 format: `YYYY-MM-DD`.	
disabled	boolean	Whether the user should be deactivated.	
email	string	The users email address (`mailPrimaryAddress`), used only when the emails domain is hosted on UCS, not to be confused with the *contact* attribute `e-mail`.	
expiration_date	string	The users password expiration date. The user wont be able to log in from that date on. Format: `YYYY-MM-DD`.	The year must be between 1961 and 2099.
roles	list	The users UCS@school roles. A list of URLs in the `roles` resource.	required when creating, see section `Changing roles` below about changing a user's roles
school	string	School (OU) the user belongs to. A URL in the `schools` resource.	required for creation when `schools` is not set
schools	list	List of schools (OUs) the user belongs to. A list of URLs in the `schools` resource.	required for creation when `school` is not set
school_classes	nested object	School classes the user is a member of. A mapping from school names to class names, for example: `{"school1": ["class1", "class2"], "school2": ["class3"]}`.	The schools must also be listed (as URLs) in the `schools` attribute.
workgroups	nested object	Workgroups the user is a member of. A mapping from school names to workgroup names, for example: `{"school1": ["wg1", "wg2"], "school2": ["wg3"]}`.	The schools must also be listed (as URLs) in the `schools` attribute.
record_uid	string	Unique identifier of the user in the upstream database the user was imported from. Used in combination with `source_uid` by the UCS@school import to uniquely identify users in both LDAP and upstream databases.	changing is strongly discouraged
source_uid	string	Identifier of the upstream database the user was imported from. Defaults to `Kelvin` if unset.	changing is strongly discouraged
ucsschool_roles	list	List of ucsschool_roles strings auto-managed by the system and custom addition ucsschool_roles strings . ucsschool_role strings with context type school are ignored. Format is `ROLE:CONTEXT_TYPE:CONTEXT`, for example: `["myrole:mycontext:gym1", "student:school:gym1"]`.	
udm_properties	nested object	Object with UDM properties. For example: `{"street": "Luise Av.", "phone": ["+49 30 321654987", "123 456 789"]}`	Must be configured, see below.

The `password` and `kelvin_password_hashes` attributes are not listed, because they cannot be retrieved, they can only be *set* when creating or modifying a user. UCS systems never store or send clear text passwords.

The `password` attribute is a single string containing the clear text password to set for the user.

The `kelvin_password_hashes` attribute is an object where all of the following attributes must be set. Setting all hashes ensures a consistent behavior for authenticating against OpenLDAP, Kerberos and Samba services:

- `user_password`: list of strings containing the LDAPs `userPassword` attribute
- `samba_nt_password`: string containing the LDAPs `sambaNTPassword` attribute
- `krb_5_key`: list of strings containing the LDAPs `krb5Key` attribute, each item is base64 encoded
- `krb5_key_version_number`: : integer containing the LDAPs `krb5KeyVersionNumber` attribute
- `samba_pwd_last_set`: integer containing the LDAPs `sambaPwdLastSet` attribute

Run the following command on a UCS system to see how those values should look like:

```
$ univention-ldapsearch -LLL uid=Administrator userPassword sambaNTPassword␣
↪krb5Key krb5KeyVersionNumber sambaPwdLastSet
```

When transmitted in a valid POST/PATCH/PUT operation, the values of `kelvin_password_hashes` will be set on the users LDAP object as given (`krb_5_key` will be base64 decoded), without further validation.

school[s]

The Users resource has a `school` attribute whose primary meaning is the position of its LDAP object in the LDAP tree. More important is its `schools` attribute. It is the list of schools that students are enrolled in or where staff and teachers work.

When creating/changing a user and sending only a value for `school`, `schools` will be a list of that one item.

When creating a user and only `schools` is sent, `school` will automatically be chosen as the alphabetically first of the list. When changing a user, the user object will stay in its OU, if it is the `schools` list, regardless of alphabetical order.

When both `school` and `schools` are used, the value of `school` must be in the list of values in `schools`.

school_classes

All school names in `school_classes` must exist (as URLs) in `schools`.

If the value of `school_classes` contains an empty dictionary in a modify request, the user will be removed from all classes.

workgroups

All school names in `workgroups` must exist (as URLs) in `schools`.

If the value of `workgroups` contains an empty dictionary in a modify request, the user will be removed from all workgroups. To avoid this behavior, simply don't pass the attribute in PUT or in PATCH and the current workgroups will be kept.

udm_properties for resource users

The attribute `udm_properties` is an object that can contain arbitrary UDM properties. It must be configured in the file `/var/lib/ucs-school-import/configs/kelvin.json`, or `/etc/ucsschool/kelvin/mapped_udm_properties.json`; see *Configuration of user object management (import configuration)* and *UDM Properties*. It must not contain UDM properties that are already available as regular attributes (like `username` → `name`, `mailPrimaryAddress` → `email`, ...).

5.3.2 Users list and search

Example `curl` command to retrieve the list of all users:

```
$ curl -i -k -X GET "https://<fqdn>/ucsschool/kelvin/v1/users/" \
    -H "accept: application/json"
    -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJh...."
```

The response headers will be:

```
HTTP/1.1 200 OK
Date: Mon, 20 Jan 2020 15:11:14 GMT
Server: uvicorn
content-length: 43274
content-type: application/json
Via: 1.1 <fqdn>
```

The response body will be:


```
[
    {
        "dn": "uid=demo_admin,cn=lehrer,cn=users,ou=DEMOSCHOOL,dc=uni,dc=ven",
        "url": "https://<fqdn>/ucsschool/kelvin/v1/users/demo_admin",
        "ucsschool_roles": ["teacher:school:DEMOSCHOOL"],
        "name": "demo_admin",
        "school": "https://<fqdn>/ucsschool/kelvin/v1/schools/DEMOSCHOOL",
        "firstname": "Demo",
        "lastname": "Admin",
        "birthday": null,
        "disabled": false,
        "email": null,
        "expiration_date": null,
        "record_uid": null,
        "roles": ["https://<fqdn>/ucsschool/kelvin/v1/roles/teacher"],
        "schools": ["https://<fqdn>/ucsschool/kelvin/v1/schools/DEMOSCHOOL"],
        "school_classes": {},
        "workgroups": {},
        "source_uid": null,
        "udm_properties": {}
    }
]
```


To search for users with usernames that contain `Brian`, append `?name=*Brian*` to the school resource. The search is case-insensitive. The URL would be: `https://<fqdn>/ucsschool/kelvin/v1/users/?name=%2ABrian%2A`

The Users resource supports searching for all attributes and to combine those. To search for users that are both `staff` and `teacher` with usernames that start with `demo`, birthday on the 3rd of February, have a lastname that ends with `sam` and are enrolled in school `demoschool`, the URL is: `https://<fqdn>/ucsschool/kelvin/v1/users/?school=demoschool&name=demo%2A&birthday=2001-02-03&lastname=%2Asam&roles=staff&roles=tea`

The user in the example response is working in two schools as both staff and teacher:

```
[
    {
        "dn": "uid=test.staff.teach,cn=lehrer und mitarbeiter,cn=users,ou=test,
↪dc=uni,dc=ven",
        "url": "https://<fqdn>/ucsschool/kelvin/v1/users/test.staff.teach",
        "ucsschool_roles": [
            "staff:school:test",
            "teacher:school:test",
            "staff:school:other",
            "teacher:school:other"
        ],
        "name": "test.staff.teach",
        "school": "https://<fqdn>/ucsschool/kelvin/v1/schools/test",
        "firstname": "staffer",
        "lastname": "teach",
        "birthday": "1988-03-18",
        "disabled": false,
        "email": "test.staff.teach@uni.dtr",
        "expiration_date": null,
        "record_uid": "test.staff.teach12",
        "roles": [
            "https://<fqdn>/ucsschool/kelvin/v1/roles/staff",
            "https://<fqdn>/ucsschool/kelvin/v1/roles/teacher"
        ],
        "schools": [
            "https://<fqdn>/ucsschool/kelvin/v1/schools/test",
            "https://<fqdn>/ucsschool/kelvin/v1/schools/other"
        ],
        "school_classes": {
            "test": ["testclass", "testclass2"],
            "other": ["otherklasse", "otherklasse2"]
        },
        "workgroups": {
            "test": ["testworkgroup", "testworkgroup2"],
            "other": ["otherworkgroup", "otherworkgroup2"]
        },
        "source_uid": "TESTID",
        "udm_properties": {
            "description": "Working at two schools.",
            "gidNumber": 9319,
            "employeeType": "Lehrer und Mitarbeiter",
            "organisation": "School board",
            "phone": ["+123-456-789", "0321-456-987"],
            "title": "Mr.",
            "uidNumber": 12503
        }
    }
]
```


5.3.3 Users retrieve

Example `curl` command to retrieve a single user object:

```
$ curl -k -X GET "https://<fqdn>/ucsschool/kelvin/v1/users/demo_staff" \
    -H "accept: application/json" \
    -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJh...." | python -m json.tool
```

With the search being case-insensitive, the URL could also have ended in `DeMo_StAfF`. The response body will be similar to the following (shortened):

```
{
    "dn": "uid=demo_staff,cn=mitarbeiter,cn=users,ou=DEMOSCHOOL,dc=uni,dc=ven",
    "url": "https://<fqdn>/ucsschool/kelvin/v1/users/demo_staff",
    "ucsschool_roles": ["staff:school:DEMOSCHOOL"],
    "name": "demo_staff"
}
```

5.3.4 Users create

When creating a user, a number of attributes must be set, unless formatted from a template (see Formatierungsschema in *UCS@school - Handbuch zur CLI-Import Schnittstelle* [1]):

- `name`
- `firstname`
- `lastname`
- `record_uid`
- `roles`
- `school` or `schools` (or both)
- `source_uid`

As an example, with the following being the content of `/tmp/create_user.json`:

```
{
    "name": "bob",
    "school": "https://<fqdn>/ucsschool/kelvin/v1/schools/DEMOSCHOOL",
    "firstname": "Bob",
    "lastname": "Marley",
    "birthday": "1945-02-06",
    "disabled": true,
    "email": null,
    "expiration_date": null,
    "record_uid": "bob23",
    "password": "s3cr3t.s3cr3t.s3cr3t",
    "roles": ["https://<fqdn>/ucsschool/kelvin/v1/roles/teacher"],
    "schools": ["https://<fqdn>/ucsschool/kelvin/v1/schools/DEMOSCHOOL"],
    "source_uid": "Reggae DB",
    "udm_properties": {
        "title": "Mr."
    }
}
```

This `curl` command will create a user from the above data:

```
$ curl -i -k -X POST "https://<fqdn>/ucsschool/kelvin/v1/users/" \
    -H "accept: application/json" \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJh...." \
    -d "$(</tmp/create_user.json)"
```

Response headers:

```
HTTP/1.1 201 Created
Date: Mon, 20 Jan 2020 16:24:33 GMT
Server: uvicorn
content-length: 714
content-type: application/json
Via: 1.1 <fqdn>
```

Response body:


```
{
    "dn": "uid=bob,cn=lehrer,cn=users,ou=DEMOSCHOOL,dc=uni,dc=ven",
    "url": "https://<fqdn>/ucsschool/kelvin/v1/users/bob",
    "ucsschool_roles": ["teacher:school:DEMOSCHOOL"],
    "name": "bob",
    "school": "https://<fqdn>/ucsschool/kelvin/v1/schools/DEMOSCHOOL",
    "firstname": "Bob",
    "lastname": "Marley",
    "birthday": "1945-02-06",
    "disabled": true,
    "email": null,
    "expiration_date": null,
    "record_uid": "bob23",
    "roles": ["https://<fqdn>/ucsschool/kelvin/v1/roles/teacher"],
    "schools": ["https://<fqdn>/ucsschool/kelvin/v1/schools/DEMOSCHOOL"],
    "school_classes": {},
    "workgroups": {},
    "source_uid": "Reggae DB",
    "udm_properties": {
        "description": null,
        "gidNumber": 5023,
        "employeeType": null,
        "organisation": null,
        "phone": [],
        "title": "Mr.",
        "uidNumber": 12711
    }
}
```


The `password` attribute is missing in the response, because UCS systems never stores or sends clear text passwords.

5.3.5 Users modify and move

It is possible to perform complete and partial updates of existing user objects. The `PUT` method expects a JSON object with all user attributes set. Nevertheless, the attribute `workgroups` can be skipped to preserve its current value. The `password` attribute should *not* be sent repeatedly, as most password policies forbid reusing the same password. The `PATCH` method will update only those attributes sent in the request. Both methods return a complete Users resource in the response body, exactly as a `GET` request would.

PUT example

All required attributes must be sent with a `PUT` request.

As an example, with the following being the content of `/tmp/mod_user.json`:


```
{
    "name": "bob",
    "school": "https://<fqdn>/ucsschool/kelvin/v1/schools/DEMOSCHOOL",
    "firstname": "Bob72",
    "lastname": "Marley72",
    "record_uid": "bob72",
    "roles": ["https://<fqdn>/ucsschool/kelvin/v1/roles/teacher"],
    "schools": ["https://<fqdn>/ucsschool/kelvin/v1/schools/DEMOSCHOOL"],
    "source_uid": "Kelvin Test2",
    "udm_properties": {"title": "Mr.2"}
}
```


This `curl` command will modify the user with the above data:

```
$ curl -i -k -X PUT "https://<fqdn>/ucsschool/kelvin/v1/users/bob" \
    -H "accept: application/json" \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJh...." \
    -d "$(</tmp/mod_user2.json)"
```

Response headers:

```
HTTP/1.1 200 OK
Date: Tue, 21 Jan 2020 22:40:21 GMT
Server: uvicorn
content-length: 721
content-type: application/json
Via: 1.1 <fqdn>
```

Response body:


```
{
    "birthday": null,
    "disabled": false,
    "dn": "uid=bob,cn=lehrer,cn=users,ou=DEMOSCHOOL,dc=uni,dc=ven",
    "email": null,
    "expiration_date": null,
    "firstname": "Bob72",
    "lastname": "Marley72",
    "name": "bob",
    "record_uid": "bob72",
    "roles": ["https://<fqdn>/ucsschool/kelvin/v1/roles/teacher"],
    "school": "https://<fqdn>/ucsschool/kelvin/v1/schools/DEMOSCHOOL",
    "school_classes": {},
    "workgroups": {},
    "schools": ["https://<fqdn>/ucsschool/kelvin/v1/schools/DEMOSCHOOL"],
    "source_uid": "Kelvin Test2",
    "ucsschool_roles": ["teacher:school:DEMOSCHOOL"],
    "udm_properties": {
        "description": null,
        "employeeType": null,
        "gidNumber": 5023,
        "organisation": null,
        "phone": [],
        "title": "Mr.2",
        "uidNumber": 12816
    },
    "url": "https://<fqdn>/ucsschool/kelvin/v1/users/bob"
}
```


PATCH example

Only the attributes that should be changed are sent with a `PATCH` request. The following `curl` command will modify the users given name only:

```
$ curl -i -k -X PATCH "https://<fqdn>/ucsschool/kelvin/v1/users/bob" \
    -H "accept: application/json" \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJh...." \
    -d '{"firstname": "Robert Nesta"}'
```

Response headers:

```
HTTP/1.1 200 OK
Date: Tue, 21 Jan 2020 22:51:40 GMT
Server: uvicorn
content-length: 728
content-type: application/json
Via: 1.1 <fqdn>
```

Response body, abbreviated: the rest is the same:

```
{
    "birthday": null,
    "disabled": false,
    "dn": "uid=bob,cn=lehrer,cn=users,ou=DEMOSCHOOL,dc=uni,dc=ven",
    "email": null,
    "expiration_date": null,
    "firstname": "Robert Nesta"
}
```

Move

When a `PUT` or `PATCH` request change the `school` or `schools` attribute, the users LDAP object may be moved to a new position in the LDAP tree.

A move will only happen, when the new value for `school` is not in `schools`.

When using `PATCH` and changing only `school`, `schools` may be updated to contain the new value of `school`.

While changing the `name` attribute is technically also a move, the objects *position* in the LDAP tree will not change - only its name.

Changing a users roles

Since version `1.3.0` of the *UCS@school Kelvin REST API* app it is possible to change a users roles. Not all role combination or changes are allowed though, and roles may have extra requirements. The following lists transitions where the API user has to take extra care:

Old	New	Note
any	staff	Staff users have no school classes. The `school_class` attribute will be cleared automatically by the Kelvin API.
any	student	Students must be member of one school class for each school they are a member of. When changing the `roles` attribute, the user must already have a corresponding `school_class` entry or a new value for `school_class` must be sent in the same request.
any	student	The transition is not allowed if the user is also a school administrator.

UCS@school user objects have a few attributes and group memberships that must be set correctly. The online article KB 15630 - How a UCS@school user should look like describes those. The Kelvin API will take care of those settings, when changing user roles.

Please be aware that changing a users role can have serious side effects. It might be necessary to make further changes to a user object or to other systems. For some of these processes hooks for the Kelvin API could be written. Please test all your role changing scenarios thoroughly. A few examples of possible problems:

- The UCS@school import is used to provision users. The `source_uid` user attribute is used to select which user accounts to include in searches for existing uses. If the imports are done through the graphical UMC module, the `source_uid` attribute contains the role of the imported user. When user roles are changed through the *UCS@school Kelvin REST API*, the `source_uid` attribute is *not* adapted. If in the mentioned import case the CSV source data is not adapted, a new user would be created with the old roles and the user with the modified roles would be deleted.

- When creating users, their email addresses are created from different templates for different roles. For example `<firstname>.<lastname>@staff.<domain>` for staff members and `<firstname>[0].<lastname>@teacher.<domain>` for teachers. When user roles are changed through the *UCS@school Kelvin REST API*, the email address is *not* adapted.
- Home directories of UCS@school users are located on school servers in a directory structure containing the user's role (e.g. `/home/$OU/lehrer/$USERNAME`). The directory path is stored in the LDAP attribute `homeDirectory` / the UDM property `unixhome`. The location of home directories is of no technical consequence. When user roles are changed, the *UCS@school Kelvin REST API* will not modify the users home directory property and will not move its files and directories.

5.3.6 Users delete

The `DELETE` method is used to delete a user object:

```
$ curl -i -k -X DELETE "https://<fqdn>/ucsschool/kelvin/v1/users/bob" \
    -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJh...."
```

Response headers:

```
HTTP/1.1 204 No Content
Date: Tue, 21 Jan 2020 22:57:03 GMT
Server: uvicorn
content-type: application/json
Via: 1.1 <fqdn>
```

No response body.

5.4 Resource Classes

The `Classes` resource represents the classes a school user is a member of.

The resource objects are represented as group objects in the LDAP.

`Classes` can be created, retrieved, modified, deleted and searched for with the Kelvin API.

5.4.1 Resource representation

The following JSON is an example Classes resource in the *UCS@school Kelvin REST API*:

```
{
    "dn": "cn=DEMOSCHOOL-Democlass,cn=klassen,cn=schueler,cn=groups,ou=DEMOSCHOOL,
↪dc=******,dc=******",
    "url": "https://<fqdn>/ucsschool/kelvin/v1/classes/DEMOSCHOOL/Democlass",
    "ucsschool_roles":[
        "school_class::school:DEMOSCHOOL"
    ],
    "udm_properties": {},
    "name": "Democlass",
    "school": "https://<fqdn>/ucsschool/kelvin/v1/schools/DEMOSCHOOL",
    "description": null,
    "users": [
        "https://<fqdn>/ucsschool/kelvin/v1/users/demo_student"
    ],
    "create_share": true
}
```

Table 4: Property description

name	value	Description	Notes
dn	string	The DN of the group in LDAP	read only
url	URL	The URL of the class object in the UCS@school Kelvin API.	read only
ucsschool_roles	list	List of roles the class has. Format is `ROLE:CONTEXT_TYPE:CONTEXT`, for example: `["school_class:school:DEMOSCHOOL"]`.	auto-managed by system, setting and changing discouraged
udm_properties	nested object	Object with UDM properties. For example: `{"street": "Luise Av.", "phone": ["+49 30 321654987", "123 456 789"]}`	Must be configured.
name	string	Name of the class	editable
school	URL	School (OU) the class belongs to. A URL in the `schools` resource.	read_only
description	nulllstring	Descriptive information about a class.	editable
users	List<URL>	A list with the URL in the UCS@school Kelvin API per user within the class.	editable
create_share	boolean	Whether a share should be created for the class.	read only

5.4.2 udm_properties for resource classes

The attribute `udm_properties` is an object that can contain arbitrary UDM properties. It must be configured in the file `/etc/ucsschool/kelvin/mapped_udm_properties.json`, see *UDM Properties*.

5.4.3 List / Search

Example `curl` command to retrieve the list of all classes at `DEMOSCHOOL`

```
$ curl  -X GET  "https://<fqdn>/ucsschool/kelvin/v1/classes/?school=DEMOSCHOOL" \
        -H "accept: application/json"
        -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciO..."
```

The response headers will be:

```
HTTP/1.1 200 OK
Connection: Keep-Alive
Content-length: 462
Content-type: application/json
Date: Tue,10 May 2022 06:44:09 GMT
Keep-alive: timeout=5,max=99
Server: uvicorn
Via: 1.1 <fqdn>
```

The response body will be:

```
[
    {
        "dn": "cn=DEMOSCHOOL-Democlass,cn=klassen,cn=schueler,cn=groups,
↪ou=DEMOSCHOOL,dc=******,dc=******",
        "url": "https://<fqdn>/ucsschool/kelvin/v1/classes/DEMOSCHOOL/Democlass",
        "ucsschool_roles": [
            "school_class:school:DEMOSCHOOL"
        ],
        "udm_properties": {},
```

(continues on next page)

(continued from previous page)

```
        "name": "Democlass",
        "school": "https://<fqdn>/ucsschool/kelvin/v1/schools/DEMOSCHOOL",
        "description": null,
        "users": [
            "https://<fqdn>/ucsschool/kelvin/v1/users/demo_student"
        ],
        "create_share": true
    }
]
```

It is required to provide the `?school=<schoolname>` in the query. The search for the school name is case sensitive and requires exact match.

Only providing the school will list all classes of that school. Optionally you can search for specific class names in that school by appending `?name=<classname>` to the school resource. This search for the class name is case-insensitive and supports wildcards (*). For example to search for a class with the name `DEMOCLASS` you can append `?name=*class`. The URL would be: `https://<fqdn>/ucsschool/kelvin/v1/classes/?school=DEMOSCHOOL?name=%2class`.

5.4.4 Retrieve

Example `curl` command to retrieve the class `Democlass` at `DEMOSCHOOL`

```
$ curl  -X GET  "https://<fqdn>/ucsschool/kelvin/v1/classes/DEMOSCHOOL/Democlass" \
        -H "accept: application/json"
        -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciO..."
```

The response headers will be:

```
Connection: Keep-Alive
Content-length: 460
Content-type: application/json
Date: Tue,10 May 2022 07:55:51 GMT
Keep-alive: timeout=5,max=100
Server: uvicorn
Via: 1.1 <fqdn>
```

The response body will be:

```
{
    "dn": "cn=DEMOSCHOOL-Democlass,cn=klassen,cn=schueler,cn=groups,ou=DEMOSCHOOL,
↪dc=******,dc=******",
    "url": "https://<fqdn>/ucsschool/kelvin/v1/classes/DEMOSCHOOL/Democlass",
    "ucsschool_roles": [
        "school_class:school:DEMOSCHOOL"
    ],
    "udm_properties": {},
    "name": "Democlass",
    "school": "https://<fqdn>/ucsschool/kelvin/v1/schools/DEMOSCHOOL",
    "description": null,
    "users": [
        "https://<fqdn>/ucsschool/kelvin/v1/users/demo_student"
    ],
    "create_share": true
}
```

Matching of the queried `class` *and* `school` is case-insensitive. The response body will be identical to the response in the example above, if a school only has a single class registered. Otherwise the list of classes from the example above will contain the `class` which has been requested.

5.4.5 Modify

Example `curl` command to modify the class `Democlass2` at `DEMOSCHOOL`

```
$ curl  -X PATCH  "https://<fqdn>/ucsschool/kelvin/v1/classes/Demoschool/Democlass2
↪" \
        -H "accept: application/json" \
        -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1N..." \
        -H "Content-Type: application/json" \
        -d "{
            "name": "Democlass_2"
        }"
```

The response headers will be:

```
HTTP/1.1 200 OK
Connection: Keep-Alive
Content-length: 397
Content-type: application/json
Date: Tue,10 May 2022 07:49:13 GMT
Keep-alive: timeout=5,max=100
Server: uvicorn
Via: 1.1 <fqdn>
```

The response will be:

```
{
    "dn": "cn=Demoschool-Democlass_2,cn=klassen,cn=schueler,cn=groups,
↪ou=Demoschool,dc=******,dc=******",
    "url": "https://<fqdn>/ucsschool/kelvin/v1/classes/Demoschool/Democlass_2",
    "ucsschool_roles": [
    "school_class:school:Demoschool"
    ],
    "udm_properties": {},
    "name": "Democlass_2",
    "school": "https://<fqdn>/ucsschool/kelvin/v1/schools/Demoschool",
    "description": null,
    "users": [],
    "create_share": true
}
```

The example shows how to rename a certain `class`. Optionally `description`, `udm_properties` and/or `users` can be modified. But a `class` objects `school` or `create_share` can't be modified.

5.4.6 Create

Example `curl` command to create the class `Democlass2` at `DEMOSCHOOL`

```
$ curl  -X POST  "https://<fqdn>/ucsschool/kelvin/v1/classes/" \
        -H "accept: application/json" \
        -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1..." \
        -H "Content-Type: application/json" \
        -d "{
        "name": "Democlass2",
            "school": "https://<fqdn>/ucsschool/kelvin/v1/schools/DEMOSCHOOL"
        }"
```

The response headers will be:

```
HTTP/1.1 201 CREATED
Connection: Keep-Alive
```

(continues on next page)

(continued from previous page)

```
Content-length: 394
Content-type: application/json
Date: Tue,10 May 2022 07:45:30 GMT
Keep-alive: timeout=5,max=100
Server: uvicorn
Via: 1.1 <fqdn>
```

The response will be:

```
{
    "dn": "cn=DEMOSCHOOL-Democlass2,cn=klassen,cn=schueler,cn=groups,ou=DEMOSCHOOL,
↪dc=******,dc=******",
    "url": "https://<fqdn>/ucsschool/kelvin/v1/classes/DEMOSCHOOL/DEMOCLASS_2",
    "ucsschool_roles": [
        "school_class:school:DEMOSCHOOL"
    ],
    "udm_properties": {},
    "name": "Democlass2",
    "school": "https://<fqdn>/ucsschool/kelvin/v1/schools/DEMOSCHOOL",
    "description": null,
    "users": [],
    "create_share": true
}
```

The queried school has to exist, whilst the `class` to be created must **not** exist. To create a `class` its name and the corresponding school must be provided. Optionally a `description`, `udm_properties`, `users` and/or `create_share` can be provided on creation.

5.4.7 Delete

Example `curl` command to delete the class `Democlass2` at `DEMOSCHOOL`

```
$ curl  -X DELETE  "https://<fqdn>/ucsschool/kelvin/v1/classes/DEMOSCHOOL/
↪Democlass2" \
        -H "accept: */*" \
        -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9..."
```

The response headers will be:

```
HTTP/1.1 204 NO CONTENT
Connection: keep-alive
Date: Tue,10 May 2022 07:38:49 GMT
Keep-alive: timeout=5,max=100
Server: uvicorn
Via: 1.1 <fqdn>
```

The server responses with 204 (with no body), if a class got deleted successfully. Matching of the queried `class` *and* `school` is case-insensitive.

5.5 Resource Workgroups

The `Workgroups` resource represents the workgroups a school user is a member of.

The resource objects are represents as group objects in the LDAP.

`Workgroups` can be created, retrieved, modified, deleted and searched for with the Kelvin API.

5.5.1 Workgroups resource representation

The following JSON is an example Workgroups resource in the *UCS@school Kelvin REST API*:

```
{
    "dn": "cn=DEMOSCHOOL-Demoworkgroup,cn=schueler,cn=groups,ou=DEMOSCHOOL,
↪dc=******,dc=******",
    "url": "https://<fqdn>/ucsschool/kelvin/v1/workgroups/DEMOSCHOOL/Demoworkgroup
↪",
    "ucsschool_roles":[
        "workgroup::school:DEMOSCHOOL"
    ],
    "udm_properties": {},
    "name": "Demoworkgroup",
    "school": "https://<fqdn>/ucsschool/kelvin/v1/schools/DEMOSCHOOL",
    "description": null,
    "users": [
        "https://<fqdn>/ucsschool/kelvin/v1/users/demo_student"
    ],
    "create_share": true,
    "email": null,
    "allowed_email_senders_users": [],
    "allowed_email_senders_groups": []
}
```

Table 5: Property description

name	value	Description	Notes
dn	string	dn of the LDAP	read only
url	URL	The URL of the workgroup object in the UCS@school Kelvin API.	read only
ucsschool_roles	list	List of roles the workgroup has. Format is `ROLE:CONTEXT_TYPE:CONTEXT`, for example: `["workgroup:school:DEMOSCHOOL"]`.	auto-managed by system, setting and changing discouraged
udm_properties	nested object	Object with UDM properties. For example: `{"street": "Luise Av.", "phone": ["+49 30 321654987", "123 456 789"]}`	Must be configured.
name	string	Name of the workgroup	editable
school	URL	The URL of the school object a workgroup belongs to in the UCS@school Kelvin API.	read_only
description	null|string	Descriptive information about a workgroup.	editable
users	List<URL>	A list with the URL in the UCS@school Kelvin API per user within the workgroup.	editable
create_share	boolean	Whether a share should be created for the workgroup.	read only
email	string	Email	editable
allowed_email_senders_users	list	Users that are allowed to send e-mails to the workgroup.	editable
allowed_email_senders_groups	list	Groups that are allowed to send e-mails to the workgroup.	editable

5.5.2 udm_properties

The attribute `udm_properties` is an object that can contain arbitrary UDM properties. It must be configured in the file `/etc/ucsschool/kelvin/mapped_udm_properties.json`, see *UDM Properties*.

5.5.3 Workgroups list and search

Example `curl` command to retrieve the list of all workgroups at `DEMOSCHOOL`

```
$ curl  -X GET  "https://<fqdn>/ucsschool/kelvin/v1/workgroups/?school=DEMOSCHOOL"␣
↪\
        -H "accept: application/json"
        -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciO..."
```

The response headers will be:

```
HTTP/1.1 200 OK
Connection: Keep-Alive
Content-length: 462
Content-type: application/json
Date: Tue,10 May 2022 06:44:09 GMT
Keep-alive: timeout=5,max=99
Server: uvicorn
Via: 1.1 <fqdn>
```

The response body will be:


```
[
    {
        "dn": "cn=DEMOSCHOOL-Demoworkgroup,cn=schueler,cn=groups,ou=DEMOSCHOOL,
↪dc=******,dc=******",
        "url": "https://<fqdn>/ucsschool/kelvin/v1/workgroups/DEMOSCHOOL/
↪Demoworkgroup",
        "ucsschool_roles": [
            "workgroup:school:DEMOSCHOOL"
        ],
        "udm_properties": {},
        "name": "Demoworkgroup",
        "school": "https://<fqdn>/ucsschool/kelvin/v1/schools/DEMOSCHOOL",
        "description": null,
        "users": [
            "https://<fqdn>/ucsschool/kelvin/v1/users/demo_student"
        ],
        "create_share": true,
        "email": null,
        "allowed_email_senders_users": [],
        "allowed_email_senders_groups": []
    }
]
```


It is required to provide the `?school=<schoolname>` in the query. The search for the school name is case sensitive and requires exact match.

Only providing the school will list all workgroups of that school. Optionally you can search for specific workgroup names in that school by appending `?name=<workgroupname>` to the school resource. This search for the workgroup name is case-insensitive and supports wildcards (*). For example to search for a workgroup with the name `DEMOWORKGROUP` you can append `?name=*workgroup`. The URL would be: `https://<fqdn>/ucsschool/kelvin/v1/workgroups/?school=DEMOSCHOOL?name=%2workgroup`.

5.5.4 Workgroups retrieve

Example `curl` command to retrieve the workgroup `Demoworkgroup` at `DEMOSCHOOL`

```
$ curl  -X GET  "https://<fqdn>/ucsschool/kelvin/v1/workgroups/DEMOSCHOOL/
↪Demoworkgroup" \
        -H "accept: application/json"
        -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciO..."
```

The response headers will be:

```
Connection: Keep-Alive
Content-length: 460
Content-type: application/json
Date: Tue,10 May 2022 07:55:51 GMT
Keep-alive: timeout=5,max=100
Server: uvicorn
Via: 1.1 <fqdn>
```

The response body will be:


```
{
    "dn": "cn=DEMOSCHOOL-Demoworkgroup,cn=schueler,cn=groups,ou=DEMOSCHOOL,
↪dc=******,dc=******",
    "url": "https://<fqdn>/ucsschool/kelvin/v1/workgroups/DEMOSCHOOL/Demoworkgroup
↪",
    "ucsschool_roles": [
```


(continues on next page)

(continued from previous page)

```
        "workgroup:school:DEMOSCHOOL"
    ],
    "udm_properties": {},
    "name": "Demoworkgroup",
    "school": "https://<fqdn>/ucsschool/kelvin/v1/schools/DEMOSCHOOL",
    "description": null,
    "users": [
        "https://<fqdn>/ucsschool/kelvin/v1/users/demo_student"
    ],
    "create_share": true,
    "email": null,
    "allowed_email_senders_users": [],
    "allowed_email_senders_groups": []
}
```

Matching of the queried `workgroup` *and* `school` is case-sensitive. The response body will be identical to the response in the example above, if a school only has a single workgroup registered. Otherwise the list of workgroups from the example above will contain the `workgroup` which has been requested.

5.5.5 Workgroups modify

Example `curl` command to modify the workgroup `Demoworkgroup2` at `DEMOSCHOOL`

```
$ curl  -X PATCH  "https://<fqdn>/ucsschool/kelvin/v1/workgroups/Demoschool/
↪Demoworkgroup2" \
        -H "accept: application/json" \
        -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1N..." \
        -H "Content-Type: application/json" \
        -d "{
        "description": "The new workgroup description."
        }"
```

The response headers will be:

```
HTTP/1.1 200 OK
Connection: Keep-Alive
Content-length: 397
Content-type: application/json
Date: Tue,10 May 2022 07:49:13 GMT
Keep-alive: timeout=5,max=100
Server: uvicorn
Via: 1.1 <fqdn>
```

The response will be:

```
{
    "dn": "cn=Demoschool-Demoworkgroup2,cn=schueler,cn=groups,ou=Demoschool,
↪dc=******,dc=******",
    "url": "https://<fqdn>/ucsschool/kelvin/v1/workgroups/Demoschool/Demoworkgroup2
↪",
    "ucsschool_roles": [
    "workgroup:school:Demoschool"
    ],
    "udm_properties": {},
    "name": "Demoworkgroup2",
    "school": "https://<fqdn>/ucsschool/kelvin/v1/schools/Demoschool",
    "description": "The new workgroup description.",
    "users": [],
    "create_share": true,
```

(continues on next page)

(continued from previous page)

```
    "email": null,
    "allowed_email_senders_users": [],
    "allowed_email_senders_groups": []
}
```

The example shows how to change the description of a `workgroup`. Optionally `udm_properties` and/or `users` can be modified. But a `workgroup` object's `school` or `create_share` can't be modified.

5.5.6 Workgroups create

Example `curl` command to create the workgroup `Demoworkgroup2` at `DEMOSCHOOL`

```
$ curl  -X POST  "https://<fqdn>/ucsschool/kelvin/v1/workgroups/" \
        -H "accept: application/json" \
        -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1..." \
        -H "Content-Type: application/json" \
        -d "{
        "name": "Demoworkgroup2",
        "school": "https://<fqdn>/ucsschool/kelvin/v1/schools/DEMOSCHOOL"
        }"
```

The response headers will be:

```
HTTP/1.1 201 CREATED
Connection: Keep-Alive
Content-length: 394
Content-type: application/json
Date: Tue,10 May 2022 07:45:30 GMT
Keep-alive: timeout=5,max=100
Server: uvicorn
Via: 1.1 <fqdn>
```

The response will be:

```
{
    "dn": "cn=DEMOSCHOOL-Demoworkgroup2,cn=schueler,cn=groups,ou=DEMOSCHOOL,
↪dc=******,dc=******",
    "url": "https://<fqdn>/ucsschool/kelvin/v1/workgroups/DEMOSCHOOL/Demoworkgroup_
↪2",
    "ucsschool_roles": [
        "workgroup:school:DEMOSCHOOL"
    ],
    "udm_properties": {},
    "name": "Demoworkgroup2",
    "school": "https://<fqdn>/ucsschool/kelvin/v1/schools/DEMOSCHOOL",
    "description": null,
    "users": [],
    "create_share": true,
    "email": null,
    "allowed_email_senders_users": [],
    "allowed_email_senders_groups": []
}
```

The queried school has to exist, whilst the `workgroup` to be created must **not** exist. To create a `workgroup` its name and the corresponding school must be provided. Optionally a `description`, `udm_properties`, `users` and/or `create_share` can be provided on creation.

5.5.7 Workgroups delete

Example `curl` command to delete the workgroup `Demoworkgroup2` at `DEMOSCHOOL`

```
$ curl  -X DELETE  "https://<fqdn>/ucsschool/kelvin/v1/workgroups/DEMOSCHOOL/
↪Demoworkgroup2" \
       -H "accept: */*" \
       -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9..."
```

The response headers will be:

```
HTTP/1.1 204 NO CONTENT
Connection: keep-alive
Date: Tue,10 May 2022 07:38:49 GMT
Keep-alive: timeout=5,max=100
Server: uvicorn
Via: 1.1 <fqdn>
```

The server responses with 204 (with no body), if a workgroup got deleted successfully. Matching of the queried `workgroup` *and* `school` is case-sensitive.

CHAPTER

SIX

KNOWN ISSUES

6.1 Rebuilding the UDM REST API Client

The *UCS@school Kelvin REST API* server connects to the UDM REST API in *Univention Developer Reference* [3] server to execute modifications of the LDAP database. So the *UCS@school Kelvin REST API* server is itself a *client* of the UDM REST API. The Python UDM REST API Client library is used for communication with the UDM REST API.

The UDM REST API does also provide an OpenAPI schema. A part of the Python UDM REST API Client library is auto-generated from it using the *OpenAPI Generator* mentioned above.

> **Warning:** Whenever a new UDM module, extended option or extended attribute is installed, the *OpenAPI client* library used by the Python UDM REST API Client library **must** be rebuild to be able to access the new module/attribute.

Although the *OpenAPI client* library rebuild is automatically triggered, because of Bug #50253 the UDM REST API server will not have been reloaded. This will prevent the changes to be incorporated into the UDM REST client of the UCS@school Kelvin REST API server.

It is thus necessary to rebuild the OpenAPI client manually.

This can be done with the following commands:

```
$ systemctl restart univention-directory-manager-rest.service
$ univention-app shell ucsschool-kelvin-rest-api /var/lib/univention-appcenter/
↪apps/ucsschool-kelvin-rest-api/data/update_openapi_client
$ univention-app shell ucsschool-kelvin-rest-api /etc/init.d/ucsschool-kelvin-rest-
↪api restart
```

6.2 No pagination

Pagination of resource collections has not yet been implemented. When it is, there will be Python UDM REST API Client entries in the response `headers`. The format of the JSON response in the *body will not change*.

6.3 Creating schools does not work in single server environments

A bug exists for this, see Bug #55506.

CHAPTER

SEVEN

PYTHON CLIENT COMPATIBILITY

This page gives information about the compatible Kelvin client versions, which can be used along with the App installed on your server. External software using client versions below the listed ones will break and is not supported.

For more details about the changes, please visit read the Python client version history and the changelog which is displayed during the upgrade of your *UCS@school Kelvin REST API*.

Table 1: *UCS@school Kelvin REST API* Client Compatibility

Server Version	Minimal Client Version
1.7.0	1.7.0
1.5.5	1.6.0

CHAPTER

EIGHT

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] *UCS@school - Handbuch zur CLI-Import Schnittstelle*. Univention GmbH, 2021. URL: https://docs.software-univention.de/ucsschool-import/5.0/de/.

[2] *UCS@school - Handbuch für Administratoren*. Univention GmbH, 2021. URL: https://docs.software-univention.de/ucsschool-manual/5.0/de/.

[3] *Univention Developer Reference*. Univention GmbH, 2021. URL: https://docs.software-univention.de/developer-reference/5.0/en/.

INDEX

B

Bugzilla
    Bug #50253, 41
    Bug #55506, 42

K

Knowledge Base
    KB 15630, 28