Python UCS@school Kelvin REST API Client Documentation

Release 2.2.3

Daniel Troeder

Jun 23, 2023

Contents:

CHAPTER 1

Python UCS@school Kelvin REST API Client

Python library to interact with the UCS@school Kelvin REST API.

- Free software: GNU Affero General Public License version 3
- Documentation: https://kelvin-rest-api-client.readthedocs.io

1.1 Features

- Asynchronous
- Automatic handling of HTTP(S) sessions
- Type annotations
- ~95% test coverage (unittests + integration tests)
- Python 3.7, 3.8, 3.9, 3.10

1.2 Compatibility

A list of UCS@school Kelvin REST API server versions which introduce breaking changes can be found in the [UCS@school Kelvin REST API Documentation](https://docs.software-univention.de/ucsschool-kelvin-rest-api/kelvin-client-compatibility.html).

1.3 Usage

The `Session` context manager opens and closes a HTTP session:

```
>>> import asyncio
>>> from ucsschool.kelvin.client import Session, User, UserResource
>>>
>>> async def get_user(username: str) -> User:
...     async with Session(
...         "USERNAME",
...         "PASSWORD",
...         "master.ucs.local",
...         verify="ucs-root-ca.crt"
...     ) as session:
...         return await UserResource(session=session).get(name=username)
...
>>> obj = asyncio.run(get_user("demo_student"))
>>>
>>> print(obj)
User('name'='test_user', dn='uid=test_user,cn=schueler,cn=users,ou=DEMOSCHOOL,
↪dc=example,dc=com')
>>> print(obj.firstname, obj.lastname)
Test User
```

There are more examples in the docs *usage* section.

For HTTPS to work, the SSL CA of the target system (UCS Master) must either be publicly signed, installed on the client system or available as file (as in the example above). If the SSL CA certificate is not available `verify=False`. Obviously that is *not safe*! The CA of any UCS server can always be downloaded from `http://FQDN.OF.UCS/ucs-root-ca.crt`.

1.4 Installation

Install *UCS@school Kelvin REST API Client* via pip from PyPI:

```
$ pip install kelvin-rest-api-client
```

1.5 Tests

There are some isolated unittests, but most tests run against a real *UCS@school Kelvin REST API*. A UCS Docker container has been prepared for this (additionally to the Kelvin API Docker container). The `Makefile` automates downloading and starting the Docker containers (3.2 GB GB) and running the tests. It is also possible to use an existing UCS DC Master with UCS@school and the Kelvin API installed.

The tests expect the existence of two schools (`OUs`) on the target system (the Kelvin API does not support creation of schools yet). The schools are `DEMOSCHOOL` and `DEMOSCHOOL2`. The first one usually already exists, but trying to create it again is safe. To create the schools run *on the UCS DC Master*:

```
$ /usr/share/ucs-school-import/scripts/create_ou DEMOSCHOOL
$ /usr/share/ucs-school-import/scripts/create_ou DEMOSCHOOL2
```

Furthermore an email domain must exist:

```
$ udm mail/domain create \
    --ignore_exists \
    --position "cn=domain,cn=mail,$(ucr get ldap/base)" \
    --set name="$(ucr get domainname)"
```

Since version `1.5.0` the Kelvin REST API supports UDM properties in all resources. A configuration is required for the tests for this feature:

```
$ cat > /etc/ucsschool/kelvin/mapped_udm_properties.json <<__EOF__
{
    "user": ["title"],
    "school_class": ["mailAddress"],
    "school": ["description"]
}
__EOF__
```

The provided UCS Docker containers already contain both OUs. They can be started using the Makefile:

```
$ make start-docker-containers

Downloading Docker image '..-ucsschool-udm-rest-api-only:stable-4.4-8'...
Downloading Docker image '../ucsschool-kelvin-rest-api:1.5.5'...
Starting UCS docker container...
Waiting for UCS docker container to start...
Waiting for IP address of UCS container...
Waiting for UDM REST API..........
Creating Kelvin REST API container...
Configuring Kelvin REST API container...
Rebuilding the OpenAPI client library in the Kelvin API Container...
Starting Kelvin REST API server...
Waiting for Kelvin docker container to start...
Waiting for IP address of Kelvin container...
Waiting for Kelvin API...
Fixing log file permissions...
Setting up reverse proxy...
==> UDM REST API log file: /tmp/udm-rest-api-log/directory-manager-rest.log
==> UDM REST API: http://172.17.0.2/univention/udm/
==> Kelvin API configs: /tmp/kelvin-api/configs/
==> Kelvin API hooks: /tmp/kelvin-api/kelvin-hooks/
==> Kelvin API log file: /tmp/kelvin-api/log/http.log
==> Kelvin API: http://172.17.0.3:8911/ucsschool/kelvin/v1/docs
==> Kelvin API: https://172.17.0.2/ucsschool/kelvin/v1/docs
```

The Docker containers can be stopped and removed by running:

```
$ make stop-and-remove-docker-containers
```

The Docker images will not be removed, only the running containers.

Run tests with current Python interpreter:

```
$ make test
```

Using tox the tests can be executed with all supported Python versions:

```
$ make test-all
```

To use an existing UCS server for the tests, copy the file `tests/test_server_example.yaml` to `tests/test_server.yaml` and adapt the settings before starting the tests:

```
$ cp tests/test_server_example.yaml tests/test_server.yaml
$ $EDITOR tests/test_server.yaml
# check settings with a single test:
```

(continues on next page)

(continued from previous page)

```
$ python -m pytest tests/test_user.py::test_get
# if OK, run all tests:
$ make test
```

1.6 Logging

Standard logging is used for tracking the libraries activity. To capture the log messages for this project, subscribe to a logger named `ucsschool.kelvin.client`. *Attention:* Passwords and session tokens will be logged at log level `DEBUG`!

The *UCS@school Kelvin REST API* on the UCS server logs into the file `/var/log/univention/ucsschool-kelvin-rest-api/http.log`. The *UDM REST API* on the UCS server logs into the file `/var/log/univention/directory-manager-rest.log`.

1.7 Repo permissions

- Github: @dansan and @JuergenBS
- Gitlab: @JuergenBS
- PyPI: @dansan and @SamuelYaron
- RTD: @dansan and @SamuelYaron

1.8 Credits

CHAPTER 2

Installation

2.1 Stable release

To install *Python UCS@school Kelvin REST API Client*, run this command in your terminal:

```
$ pip install kelvin-rest-api-client
```

This is the preferred method to install *Python UCS@school Kelvin REST API Client*, as it will always install the most recent stable release. The major and the minor version of the *Python UCS@school Kelvin REST API Client* must be at least as high as the version of the *UCS@school Kelvin REST API* which is used. The patch level may differ.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for *Python UCS@school Kelvin REST API Client* can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/univention/kelvin-rest-api-client
```

Or download the tarball:

```
$ curl -OL https://github.com/univention/kelvin-rest-api-client/tarball/master
```

Once you have a copy of the source, you can install it (after unpacking) with:

```
$ python setup.py install
```

CHAPTER 3

Usage

The Kelvin APIs resources (users, school classes etc) support a varying range of operations: retrieve, search, create, modify, move and delete. Some resources support all operations, others only a subset. See each resources usage section about which operations are supported.

All requests to the Kelvin API must be authenticated. The `Session` class takes care of that. Section *Authentication and authorization* describes its usage.

3.1 Authentication and authorization

All requests to the Kelvin API must be authenticated. Requests to resource endpoints must carry a valid token. The token will expire after an hour and must then be refreshed. The `Session` class of the *Kelvin REST API Client* takes care of all that. All it needs are the credentials of a user that is a members of the group `ucsschool-kelvin-rest-api-admins` on the host that is running the Kelvin 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.

To use the *Kelvin REST API Client*, first get the UCS servers CA certificate (from `http://FQDN.OF.UCS/ucs-root-ca.crt`). Then use the `Session` context manager to open an authenticated HTTPS session for use by the *Kelvin REST API Client* resource classes.

```
$ wget --no-check-certificate -O /tmp/ucs-root-ca.crt https://master.ucs.local/ucs-
↪root-ca.crt
```

We'll store the credentials and path to the UCS CA certificate for the following examples in a dictionary:

```
credentials = {
    "username": "Administrator",
    "password": "s3cr3t",
    "host": "master.ucs.local",
    "verify": "/tmp/ucs-root-ca.crt",
}
```

For testing purposes the clients certificate check can be disabled by setting the value of `verify` to the boolean value `False`.

3.2 Correlation ID

A unique, random correlation ID will be sent with each request. The value can be set, when creating the `Session` object. If not set, a random ID will be generated automatically.

The header name defaults to `X-Request-ID`. A different one can be set, by passing it with the `request_id_header` argument to the `Session` constructor. The name of the header that is sent, will be in the header `Access-Control-Expose-Headers`.

If an ID already exists, e.g. when inside a micro services chain, pass the ID on to the Kelvin REST API server with `Session(..., request_id="a1b2c3d4e5")`.

3.3 Language Header

An `Accept-Language` header can be sent with each request. The value can be set, when creating the `Session` object. If not set, the `Accept-Language` Header will not be sent.

When an `Accept-Language` header is sent, the Kelvin REST API error messages are translated into the corresponding language. (currently available languages: German and English)

To set the `Accept-Language` header, pass the `language` attribute to the `Session` constructor: `Session(..., language="de-DE")`. It is also possible to change the `Accept-Language` header within a `Session` context by passing the `language` attribute to the `KelvinObject` or the `KelvinRessource` constructor.

Note: The Kelvin REST API server version must be greater than `1.7.0` to handle the `Accept-Language` header.

3.3.1 Set `Accept-Language` header within a `Session` context

Create user example:

```
from ucsschool.kelvin.client import Session, User

async with Session(**credentials) as session:
    user = User(
        ...,
        session=session,
        language="de-DE"
    )
    await user.save()
```

Retrieve User example:

```
from ucsschool.kelvin.client import Session, UserResource

async with Session(**credentials) as session:
    user = await UserResource(session=session, language="de-DE").get(name="test1")
```

3.4 Resource Role

The `Role` resource is *not* represented in the LDAP tree. The objects exist only as a vehicle to classify user objects.

3.4.1 Kelvin API documentation

Please see the Kelvin API documentation section Resource Roles about allowed values for the attributes.

3.4.2 Role class

The *ucsschool.kelvin.client.Role* class has the following public attributes and methods:

```
class Role(KelvinObject):
    def __init__(
        self,
        name: str,
        *,
        display_name: str = None,
        url: str = None,
        session: Session = None,
        language: str = None,
        **kwargs,
    ):
        self.name = name
        self.display_name = display_name
        self.url = url
        self.session = session
        if language:
            self.session.language = language
        del self.dn
        del self.ucsschool_roles
        del self.udm_properties

    async def reload(self) -> School:
        ...

    async def save(self) -> School:
        raise NotImplementedError()

    async def delete(self) -> None:
        raise NotImplementedError()

    def as_dict(self) -> Dict[str, Any]:
        ...
```

Note: The Kelvin API does not yet support creating, changing or deleting role objects, and thus the Kelvin API client doesn't either. Using `Role.save()` or `Role.delete()` will raise a `NotImplementedError` exception.

3.4.3 RoleResource class

The *ucsschool.kelvin.client.RoleResource* class has the following public attributes and methods:

```
class RoleResource(KelvinResource):
    def __init__(self, session: Session, language: str = None):
        ...
    async def get(self, **kwargs) -> School:
        ...
    async def get_from_url(self, url: str) -> School:
        ...
    async def search(self, **kwargs) -> AsyncIterator[School]:
        ...
```

3.4.4 Create role

The Kelvin API does not yet support creating role objects, and thus the Kelvin API client doesn't either.

3.4.5 Retrieve role

```
from ucsschool.kelvin.client import Session, RoleResource

async with Session(**credentials) as session:
    role = await RoleResource(session=session).get(name="student")

role.as_dict()
{'name': 'student',
 'display_name': 'student',
 'url': 'https://master.ucs.local/ucsschool/kelvin/v1/roles/student'}
```

3.4.6 Check if role exists

```
from ucsschool.kelvin.client import Session, RoleResource

async with Session(**credentials) as session:
    if await RoleResource(session=session).exists(name="student"):
        print("The role 'student' exists!")
```

Note: This method only works with Kelvin server version 1.8.8 or newer.

3.4.7 Search roles

The `search()` method allows searching for roles. No filter argument are supported.

```
from ucsschool.kelvin.client import Session, RoleResource

async with Session(**credentials) as session:
    async for role in RoleResource(session=session).search():
        print(role)

Role('name'='staff')
Role('name'='student')
Role('name'='teacher')
```

3.4.8 Change role properties

The Kelvin API does not yet support changing role objects, and thus the Kelvin API client doesn't either.

3.4.9 Move role

Role objects do not support moving.

3.4.10 Delete role

The Kelvin API does not yet support deleting role objects, and thus the Kelvin API client doesn't either.

3.5 Resource School

The `School` resource is represented in the LDAP tree as `OU` objects.

To list those LDAP objects run in a terminal:

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

UCS@school uses the UDM to access the LDAP directory. UDM properties have different names than their associated LDAP attributes. Their values may also differ. To list the same UDM objects as above, run:

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

3.5.1 Kelvin API documentation

Please see the Kelvin API documentation section Resource Schools about allowed values for the attributes.

3.5.2 School class

The *ucsschool.kelvin.client.School* class has the following public attributes and methods:

```
class School(KelvinObject):
    def __init__(
        self,
        name: str,
        *,
        display_name: str = None,
        educational_servers: List[str] = None,
        administrative_servers: List[str] = None,
        class_share_file_server: str = None,
        home_share_file_server: str = None,
        udm_properties: Dict[str, Any] = None,
        ucsschool_roles: List[str] = None,
        dn: str = None,
        url: str = None,
        session: Session = None,
```

(continues on next page)

(continued from previous page)

```
        language: str = None,
        **kwargs,
    ):
        self.name = name
        self.display_name = display_name
        self.educational_servers = educational_servers
        self.administrative_servers = administrative_servers
        self.class_share_file_server = class_share_file_server
        self.home_share_file_server = home_share_file_server
        self.udm_properties = udm_properties or {}
        self.ucsschool_roles = ucsschool_roles
        self.dn = dn
        self.url = url
        self.session = session
        if language:
            self.session.language = language

    async def reload(self) -> School:
        ...

    async def save(self) -> School:
        ...

    async def delete(self) -> None:
        raise NotImplementedError()

    def as_dict(self) -> Dict[str, Any]:
        ...
```

Note: The Kelvin API does not yet support changing or deleting school objects, and thus the Kelvin API client doesn't either. Using `School.save()` or `School.delete()` on existing school objects will raise a `NotImplementedError` exception.

3.5.3 SchoolResource class

The *`ucsschool.kelvin.client.SchoolResource`* class has the following public attributes and methods:

```
class SchoolResource(KelvinResource):
    def __init__(self, session: Session, language: str = None):
        ...
    async def get(self, **kwargs) -> School:
        ...
    async def get_from_url(self, url: str) -> School:
        ...
    async def search(self, **kwargs) -> AsyncIterator[School]:
        ...
    async def exists(self, **kwargs) -> bool:
        ...
```

3.5.4 Create school

Since version `1.4.0` the Kelvin REST API supports the creation of school (OU) objects. The result should be the same as using the `Schools` UMC module or running the `/usr/share/ucs-school-import/scripts/create_ou` script from the command line. The *Kelvin REST API Client* supports this feature since version `0.3.0`.

The only required attribute is `name`. An educational domain controller for each school is required however. If none is passed in the request, one will be created automatically as `dc<name>`. If `name` is longer than 11 characters this will fail. In that case the hostname must be passed in `educational_servers`.

For historical reasons `administrative_servers` and `educational_servers` are lists that must contain exactly one item.

```
from ucsschool.kelvin.client import Session, School

async with Session(**credentials) as session:
    school = School(
        name="testou",
        display_name="A test school",
        session=session,
    )
    await school.save()

school.as_dict()
{'name': 'testou',
 'ucsschool_roles': ['school:school:testou'],
 'display_name': 'A test school',
 'educational_servers': ['dctestou'],
 'administrative_servers': [],
 'class_share_file_server': 'dctestou',
 'home_share_file_server': 'dctestou',
 'udm_properties': {},
 'dn': 'ou=testou,dc=example,dc=com',
 'url': 'https://master.ucs.local/ucsschool/kelvin/v1/schools/testou'}
```

Schools are saved as containers in the UCS LDAP. The result can be verified on the target system using UDM:

```
$ udm container/ou list --filter ou=testou

DN: ou=testou,dc=example,dc=com
  name: testou
  displayName: A test school
  ucsschoolRole: school:school:testou
  ucsschoolClassShareFileServer: cn=dctestou,cn=dc,cn=server,cn=computers,ou=testou,
↪dc=example,dc=com
  ucsschoolHomeShareFileServer: cn=dctestou,cn=dc,cn=server,cn=computers,ou=testou,
↪dc=example,dc=com
  ...
```

The administrative and educational server information is stored as group membership. If interested, search using the hostname prefixed with a dollar (`dctestou$`):

```
$ udm groups/group list --filter 'memberUid=dctestou$'
```

3.5.5 Retrieve school

```
from ucsschool.kelvin.client import Session, SchoolResource

async with Session(**credentials) as session:
    school = await SchoolResource(session=session).get(name="DEMOSCHOOL")

school.as_dict()
```

(continues on next page)

(continued from previous page)

```
{'name': 'DEMOSCHOOL',
 'ucsschool_roles': ['school:school:DEMOSCHOOL'],
 'display_name': 'Demo School',
 'educational_servers': ['DEMOSCHOOL'],
 'administrative_servers': [],
 'class_share_file_server': 'DEMOSCHOOL',
 'home_share_file_server': 'DEMOSCHOOL',
 'dn': 'ou=DEMOSCHOOL,dc=example,dc=com',
 'url': 'https://master.ucs.local/ucsschool/kelvin/v1/schools/DEMOSCHOOL'}
```

3.5.6 Check if school exists

```
from ucsschool.kelvin.client import Session, SchoolResource

async with Session(**credentials) as session:
    if await SchoolResource(session=session).exists(name="DEMOSCHOOL"):
        print("The school exists!")
```

3.5.7 Search schools

The `search()` method allows searching for schools. The optional `name` argument supports an inexact search using `*` as a placeholder.

```
from ucsschool.kelvin.client import Session, SchoolResource

async with Session(**credentials) as session:
    async for school in SchoolResource(session=session).search(name="DEMO*"):
        print(school)

School('name'='DEMOSCHOOL', dn='ou=DEMOSCHOOL,dc=example,dc=com')
School('name'='DEMOSCHOOL2', dn='ou=DEMOSCHOOL2,dc=example,dc=com')
```

3.5.8 Change school properties

The Kelvin API does not yet support changing school objects, and thus the Kelvin API client doesn't either.

3.5.9 Move school

School objects do not support moving.

3.5.10 Delete school

The Kelvin API does not yet support deleting school objects, and thus the Kelvin API client doesn't either.

3.6 Resource SchoolClass

The `SchoolClass` resource is represented in the LDAP tree as group objects.

To list those LDAP objects run in a terminal:

```
FILTER='(&(objectClass=ucsschoolGroup)(ucsschoolRole=school_class:*))'
univention-ldapsearch -LLL "$FILTER"
```

UCS@school uses the UDM to access the LDAP directory. UDM properties have different names than their associated LDAP attributes. Their values may also differ. To list the same UDM objects as above, run:

```
$ FILTER='(&(objectClass=ucsschoolGroup)(ucsschoolRole=school_class:*))'
$ udm groups/group list --filter "$FILTER"
```

3.6.1 SchoolClass class

The *ucsschool.kelvin.client.SchoolClass* class has the following public attributes and methods:

```
class SchoolClass(KelvinObject):
    def __init__(
        self,
        name: str,
        school: str,
        *,
        description: str = None,
        users: List[str] = None,
        create_share: bool = True,
        udm_properties: Dict[str, Any] = None,
        ucsschool_roles: List[str] = None,
        dn: str = None,
        url: str = None,
        session: Session = None,
        language: str = None,
        **kwargs,
    ):
        self.name = name
        self.school = school
        self.description = description
        self.users = users
        self.create_share = create_share
        self.udm_properties = udm_properties or {}
        self.ucsschool_roles = ucsschool_roles
        self.dn = dn
        self.url = url
        self.session = session
        if language:
            self.session.language = language


    async def reload(self) -> SchoolClass:
        ...
    async def save(self) -> SchoolClass:
        ...
    async def delete(self) -> None:
        ...
    def as_dict(self) -> Dict[str, Any]:
        ...
```

3.6.2 SchoolClassResource class

The *ucsschool.kelvin.client.SchoolClassResource* class has the following public attributes and methods:

```
class SchoolClassResource(KelvinResource):
    def __init__(self, session: Session, language: str = None):
        ...
    async def get(self, **kwargs) -> SchoolClass:
        ...
    async def get_from_url(self, url: str) -> SchoolClass:
        ...
    async def search(self, **kwargs) -> AsyncIterator[SchoolClass]:
        ...
```

3.6.3 Create school class

School classes can be created explicitly or implicitly when creating or modifying users.

School classes will be automatically created when mentioned in a users `school_classes` attribute. They will however not be deleted automatically if they are removed from all users and are thus empty.

```
from ucsschool.kelvin.client import Session, SchoolClass

async with Session(**credentials) as session:
    sc = SchoolClass(
        name="testclass",
        school="DEMOSCHOOL",
        description="A test class",
        users=["demo_student", "demo_teacher"],
        create_share=True,
        session=session,
    )
    await sc.save()

sc.as_dict()
{'name': 'testclass',
 'ucsschool_roles': ['school_class:school:DEMOSCHOOL'],
 'school': 'DEMOSCHOOL',
 'description': 'A test class',
 'users': ['demo_student', 'demo_teacher'],
 'create_share': True,
 'udm_properties': {},
 'dn': 'cn=DEMOSCHOOL-testclass,cn=klassen,cn=schueler,cn=groups,ou=DEMOSCHOOL,
↪dc=example,dc=com',
 'url': 'https://master.ucs.local/ucsschool/kelvin/v1/classes/DEMOSCHOOL/testclass'}
```

School classes are saved as groups in the UCS LDAP. The result can be verified on the target system using UDM:

```
$ udm groups/group list --filter cn=DEMOSCHOOL-testclass

DN: cn=DEMOSCHOOL-testclass,cn=klassen,cn=schueler,cn=groups,ou=DEMOSCHOOL,dc=example,
↪dc=com
  name: DEMOSCHOOL-testclass
  description: A test class
  ucsschoolRole: school_class:school:DEMOSCHOOL
```

(continues on next page)

(continued from previous page)

```
  users: uid=demo_student,cn=schueler,cn=users,ou=DEMOSCHOOL,dc=example,dc=com
  users: uid=demo_teacher,cn=lehrer,cn=users,ou=DEMOSCHOOL,dc=example,dc=com
  ...
```

Every school class has a share with the same name:

```
$  udm shares/share list --filter cn=DEMOSCHOOL-testclass

DN: cn=DEMOSCHOOL-testclass,cn=klassen,cn=shares,ou=DEMOSCHOOL,dc=example,dc=com
  name: DEMOSCHOOL-testclass
  host: DEMOSCHOOL.example.com
  path: /home/DEMOSCHOOL/groups/klassen/DEMOSCHOOL-testclass
  directorymode: 0770
  group: 7110
  ...
```

Example creating two school classes as a byproduct of creating a user:

```
from ucsschool.kelvin.client import Session, SchoolClassResource, User

async with Session(**credentials) as session:
    user = User(
        school="DEMOSCHOOL", schools=["DEMOSCHOOL"],
        roles=["student"], name="test2",
        firstname="test", lastname="two",
        record_uid="test2", source_uid="TESTID",
        school_classes={"DEMOSCHOOL": ["class1", "class2"]},
        session=session)
    await user.save()

    async for sc in SchoolClassResource(session=session).search(school="DEMOSCHOOL"):
        print(sc)

SchoolClass('name'='class1', 'school'='DEMOSCHOOL', dn='cn=DEMOSCHOOL-class1,
↪cn=klassen,cn=schueler,cn=groups,ou=DEMOSCHOOL,dc=example,dc=com')
SchoolClass('name'='class2', 'school'='DEMOSCHOOL', dn='cn=DEMOSCHOOL-class2,
↪cn=klassen,cn=schueler,cn=groups,ou=DEMOSCHOOL,dc=example,dc=com')
SchoolClass('name'='Democlass', 'school'='DEMOSCHOOL', dn='cn=DEMOSCHOOL-Democlass,
↪cn=klassen,cn=schueler,cn=groups,ou=DEMOSCHOOL,dc=example,dc=com')
```

3.6.4 Retrieve school class

It is necessary to pass both `name` and `school` arguments to the `get()` method, as the name alone wouldn't be unique in a domain (there can be classes of the same name in multiple schools).

```
from ucsschool.kelvin.client import Session, SchoolClassResource

async with Session(**credentials) as session:
    sc = await SchoolClassResource(session=session).get(
        school="DEMOSCHOOL", name="testclass"
    )

sc.as_dict()
{'name': 'testclass',
 'ucsschool_roles': ['school_class:school:DEMOSCHOOL'],
```

(continues on next page)

(continued from previous page)

```
 'school': 'DEMOSCHOOL',
 'description': 'A test class',
 'users': ['demo_student', 'demo_teacher'],
 'create_share': True,
 'dn': 'cn=DEMOSCHOOL-testclass,cn=klassen,cn=schueler,cn=groups,ou=DEMOSCHOOL,
↪dc=example,dc=com',
 'url': 'https://10.200.3.70/ucsschool/kelvin/v1/classes/DEMOSCHOOL/testclass'}
```

3.6.5 Check if school class exists

```
from ucsschool.kelvin.client import Session, SchoolClassResource

async with Session(**credentials) as session:
    if await SchoolClassResource(session=session).exists(name="testclass", school=
↪"DEMOSCHOOL"):
        print("The school class exists!")
```

3.6.6 Search school classes

The `search()` method allows searching for school classes, filtering by `school` (mandatory) and `name` (optional).

The mandatory `school` argument must be exact while the optional `name` argument support an inexact search using `*` as a placeholder.

```
from ucsschool.kelvin.client import Session, SchoolClassResource

async with Session(**credentials) as session:
    async for sc in SchoolClassResource(session=session).search(school="DEMOSCHOOL"):
        print(sc)

SchoolClass('name'='Democlass', 'school'='DEMOSCHOOL', dn='cn=DEMOSCHOOL-Democlass,
↪cn=klassen,cn=schueler,cn=groups,ou=DEMOSCHOOL,dc=example,dc=com')
SchoolClass('name'='testclass', 'school'='DEMOSCHOOL', dn='cn=DEMOSCHOOL-testclass,
↪cn=klassen,cn=schueler,cn=groups,ou=DEMOSCHOOL,dc=example,dc=com')

    async for sc in SchoolClassResource(session=session).search(
        school="DEMOSCHOOL", name="test*"
    ):
        print(sc)

SchoolClass('name'='testclass', 'school'='DEMOSCHOOL', dn='cn=DEMOSCHOOL-testclass,
↪cn=klassen,cn=schueler,cn=groups,ou=DEMOSCHOOL,dc=example,dc=com')
```

3.6.7 Change school class properties

Get the current school class object, change some attributes and save the changes back to LDAP:

```
from ucsschool.kelvin.client import Session, SchoolClassResource

async with Session(**credentials) as session:
    sc = await SchoolClassResource(session=session).get(
```

(continues on next page)

(continued from previous page)

```
        school="DEMOSCHOOL",
        name="testclass"
    )
    sc.description = "new description"
    sc.users.remove("demo_teacher")
    await sc.save()

sc.as_dict()
{'name': 'testclass',
 'ucsschool_roles': ['school_class:school:DEMOSCHOOL'],
 'school': 'DEMOSCHOOL',
 'description': 'new description',
 'users': ['demo_student'],
 'create_share': True,
 'dn': 'cn=DEMOSCHOOL-testclass,cn=klassen,cn=schueler,cn=groups,ou=DEMOSCHOOL,
↪dc=example,dc=com',
 'url': 'https://10.200.3.70/ucsschool/kelvin/v1/classes/DEMOSCHOOL/testclass'}
```

3.6.8 Move school class

School class objects do not support changing the `school`. Changing the `name` is allowed however.

```
from ucsschool.kelvin.client import Session, SchoolClassResource

async with Session(**credentials) as session:
    sc = await SchoolClassResource(session=session).get(
            school="DEMOSCHOOL",
            name="testclass"
        )
    sc.name = "testclass-new"
    await sc.save()

sc.dn
'cn=DEMOSCHOOL-testclass-new,cn,cn=klassen,cn=schueler,cn=groups,ou=DEMOSCHOOL,
↪dc=example,dc=com'
```

3.6.9 Delete school class

Get the current school class object and delete it:

```
from ucsschool.kelvin.client import Session, SchoolClassResource

async with Session(**credentials) as session:
    sc = await SchoolClassResource(session=session).get(
            school="DEMOSCHOOL",
            name="testclass"
        )
    await sc.delete()
```

3.7 Resource Users

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

To list those LDAP objects run in a terminal:

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

UCS@school uses the UDM to access the LDAP directory. UDM properties have different names than their associated LDAP attributes. Their values may also differ. To list the same UDM objects as above, run:

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

3.7.1 Kelvin API documentation

Please see the Kelvin API documentation section Resource Users about allowed values for the attributes.

3.7.2 User class

The *ucsschool.kelvin.client.User* class has the following public attributes and methods:

```
class User(KelvinObject):
    def __init__(
        self,
        name: str = None,
        school: str = None,
        *,
        firstname: str = None,
        lastname: str = None,
        birthday: datetime.date = None,
        disabled: bool = False,
        email: str = None,
        expiration_date: datetime.date = None,
        kelvin_password_hashes: PasswordsHashes = None,
        password: str = None,
        record_uid: str = None,
        roles: List[str],
        schools: List[str],
        school_classes: Dict[str, List[str]] = None,
        workgroups: Dict[str, List[str]] = None,
        source_uid: str = None,
        udm_properties: Dict[str, Any] = None,
        ucsschool_roles: List[str] = None,
        dn: str = None,
        url: str = None,
        session: Session = None,
        language: str = None,
        **kwargs,
    ):
    self.name = name
    self.school = school
    self.firstname = firstname
    self.lastname = lastname
```

(continues on next page)

(continued from previous page)

```
    self.birthday = birthday
    self.disabled = disabled
    self.email = email
    self.expiration_date = expiration_date
    self.kelvin_password_hashes = kelvin_password_hashes
    self.password = password
    self.record_uid = record_uid
    self.roles = roles
    self.schools = schools
    self.school_classes = school_classes or {}
    self.workgroups = workgroups or {}
    self.source_uid = source_uid
    self.udm_properties = udm_properties or {}
    self.ucsschool_roles = ucsschool_roles
    self.dn = dn
    self.url = url
    self.session = session
    if language:
        self.session.language = language

    async def reload(self) -> User:
        ...
    async def save(self) -> User:
        ...
    async def delete(self) -> None:
        ...
    def as_dict(self) -> Dict[str, Any]:
        ...
```

Note: The field `expiration_date` was added to the Kelvin REST API in version `1.5.1`. The client works with prior server versions, but the attribute will not be read or set.

Note: Since the Kelvin REST API client version `2.0.0`, the required argument *school* has the default argument *None*. The argument *name* is not required anymore.

3.7.3 UserResource class

ucsschool.kelvin.client.UserResource class has the following public attributes and methods:

```
class UserResource(KelvinResource):
    def __init__(self, session: Session, language: str = None):
        ...
    async def get(self, **kwargs) -> User:
        ...
    async def get_from_url(self, url: str) -> User:
        ...
    async def search(self, **kwargs) -> AsyncIterator[User]:
        ...
```

3.7.4 Create user

```
from ucsschool.kelvin.client import Session, User

async with Session(**credentials) as session:
    user = User(
        school="DEMOSCHOOL",
        schools=["DEMOSCHOOL"],
        roles=["student"],
        name="test1",
        firstname="test",
        lastname="one",
        record_uid="test1",
        source_uid="TESTID",
        session=session
    )
    await user.save()

user.dn
'uid=test1,cn=schueler,cn=users,ou=DEMOSCHOOL,dc=example,dc=com'
```

Note: Since version `2.0.0`, all attributes except `school`, `schools` and `roles` can be automatically generated on the server by defining a schema. If a schema is defined for an attribute, it can be skipped. The attributes `name` and `record_uid` have to be passed in either the constructor or a schema must exist. You can find more about schemas in the [UCS@school - Handbuch zur CLI-Import Schnittstelle (german only)](https://docs.software-univention.de/ucsschool-import/5.0/de/configuration/scheme-formatting.html#formatierungsschema).

3.7.5 Retrieve user

```
from ucsschool.kelvin.client import Session, UserResource

async with Session(**credentials) as session:
    user = await UserResource(session=session).get(name="test1")

user.as_dict()

{'name': 'test1',
 'ucsschool_roles': ['student:school:DEMOSCHOOL'],
 'school': 'DEMOSCHOOL',
 'firstname': 'test',
 'lastname': 'one',
 'birthday': None,
 'disabled': False,
 'email': None,
 'expiration_date': None,
 'kelvin_password_hashes': None,
 'password': None,
 'record_uid': 'test1',
 'roles': ['student'],
 'schools': ['DEMOSCHOOL'],
 'school_classes': {},
 'workgroups': {},
 'source_uid': 'TESTID',
```

(continues on next page)

(continued from previous page)

```
 'udm_properties': {},
 'dn': 'uid=test1,cn=schueler,cn=users,ou=DEMOSCHOOL,dc=example,dc=com',
 'url': 'https://master.ucs.local/ucsschool/kelvin/v1/users/test1'}
```

3.7.6 Check if user exists

```
from ucsschool.kelvin.client import Session, UserResource

async with Session(**credentials) as session:
    if await UserResource(session=session).exists(name="test1"):
        print("The user exists!")
```

3.7.7 Search users

The `search()` method allows searching for users, using a number of filters. Most (but now all) attributes support searching inexact, using an asterisk (`*`) as placeholder.

In the following examples the search is always limited to users of the school `DEMOSCHOOL`. In the 1. search *all* users (of the school `DEMOSCHOOL`) are searched, 2. users with a *username* starting with `t`, 3. users with a *family name* starting with `tea` and 4. users that have the *role* `teacher`.

```
from ucsschool.kelvin.client import Session, UserResource

async with Session(**credentials) as session:
    async for user in UserResource(session=session).search(school="DEMOSCHOOL"):
        print(user)

User('name'='demo_admin', dn='uid=demo_admin,cn=lehrer,cn=users,ou=DEMOSCHOOL,
↪dc=example,dc=com')
User('name'='demo_student', dn='uid=demo_student,cn=schueler,cn=users,ou=DEMOSCHOOL,
↪dc=example,dc=com')
User('name'='demo_teacher', dn='uid=demo_teacher,cn=lehrer,cn=users,ou=DEMOSCHOOL,
↪dc=example,dc=com')
User('name'='test1', dn='uid=test1,cn=schueler,cn=users,ou=DEMOSCHOOL,dc=example,
↪dc=com')

    async for user in UserResource(session=session).search(
        name="t*", school="DEMOSCHOOL"
    ):
        print(user)

User('name'='test1', dn='uid=test1,cn=schueler,cn=users,ou=DEMOSCHOOL,dc=example,
↪dc=com')

    async for user in UserResource(session=session).search(
        lastname="tea*", school="DEMOSCHOOL"
    ):
        print(user)

User('name'='demo_teacher', dn='uid=demo_teacher,cn=lehrer,cn=users,ou=DEMOSCHOOL,
↪dc=example,dc=com')

    async for user in UserResource(session=session).search(
```

(continues on next page)

(continued from previous page)

```
        roles=["teacher"], school="DEMOSCHOOL"
    ):
        print(user)

User('name'='demo_admin', dn='uid=demo_admin,cn=lehrer,cn=users,ou=DEMOSCHOOL,
↪dc=example,dc=com')
User('name'='demo_teacher', dn='uid=demo_teacher,cn=lehrer,cn=users,ou=DEMOSCHOOL,
↪dc=example,dc=com')
```

3.7.8 Change user properties

Get the current user object, change some attributes and save the changes back to LDAP:

```
from ucsschool.kelvin.client import Session, User, UserResource

async def change_properties(username: str, **changes) -> User:
    async with Session(**credentials) as session:
        user = await UserResource(session=session).get(name=username)
        for property, value in changes.items():
            setattr(user, property, value)
        return await user.save()

user = await change_properties(
    "test1",
    firstname="newfn",
    lastname="newln",
    password="password123",
)
assert user.firstname == "newfn"
assert user.lastname == "newln"
```

Hint: users cannot be modified, unless their `record_uid` and `source_uid` attributes are set (as is the case with the `demo_*` users).

3.7.9 Move user

User objects support changing both `school` and `name`.

When the `school` attribute of a user is changed, the new value *must* be part of the list in the `schools` attribute.

In the following example both `school` and `name` are changed.

```
from ucsschool.kelvin.client import Session, User, UserResource

async with Session(**credentials) as session:
    user = User(
        school="DEMOSCHOOL", schools=["DEMOSCHOOL"],
        roles=["student"], name="test1", firstname="test",
        lastname="one", record_uid="test1",
        source_uid="TESTID", session=session
    )
    await user.save()
    user.dn
    'uid=test1,cn=schueler,cn=users,ou=DEMOSCHOOL,dc=example,dc=com'
```

(continues on next page)

(continued from previous page)

```
    user.name = "test2"
    user.school = "DEMOSCHOOL2"
    user.schools = ["DEMOSCHOOL2"]
    await user.save()
    user.dn
    'uid=test2,cn=schueler,cn=users,ou=DEMOSCHOOL2,dc=example,dc=com'
```

3.7.10 Delete user

Get the current user object and delete it:

```
from ucsschool.kelvin.client import Session, User, UserResource

async with Session(**credentials) as session:
    user = await UserResource(session=session).get(name="test1")
    await user.delete()
```

Trying to retrieve the deleted user will raise a *ucsschool.kelvin.client.NoObject* exception.

3.8 Resource WorkGroup

The `WorkGroup` resource is represented in the LDAP tree as group objects.

To list those LDAP objects run in a terminal:

```
FILTER='(&(objectClass=ucsschoolGroup)(ucsschoolRole=workgroup:*))'
univention-ldapsearch -LLL "$FILTER"
```

UCS@school uses the UDM to access the LDAP directory. UDM properties have different names than their associated LDAP attributes. Their values may also differ. To list the same UDM objects as above, run:

```
$ FILTER='(&(objectClass=ucsschoolGroup)(ucsschoolRole=workgroup:*))'
$ udm groups/group list --filter "$FILTER"
```

3.8.1 WorkGroup class

The *ucsschool.kelvin.client.WorkGroup* class has the following public attributes and methods:

```
class WorkGroup(KelvinObject):
    def __init__(
        self,
        name: str,
        school: str,
        *,
        description: str = None,
        users: List[str] = None,
        email: str = None,
        allowed_email_senders_users: List[str] = None,
        allowed_email_senders_groups: List[str] = None,
        create_share: bool = True,
        udm_properties: Dict[str, Any] = None,
```

(continues on next page)

(continued from previous page)

```
        ucsschool_roles: List[str] = None,
        dn: str = None,
        url: str = None,
        session: Session = None,
        language: str = None
    ):
        self.name = name
        self.school = school
        self.description = description
        self.users = users
        self.email = email
        self.allowed_email_senders_users = allowed_email_senders_users
        self.allowed_email_senders_groups = allowed_email_senders_groups
        self.create_share = create_share
        self.udm_properties = udm_properties or {}
        self.ucsschool_roles = ucsschool_roles
        self.dn = dn
        self.url = url
        self.session = session
        if language:
            self.session.language = language


    async def reload(self) -> WorkGroup:
        ...
    async def save(self) -> WorkGroup:
        ...
    async def delete(self) -> None:
        ...
    def as_dict(self) -> Dict[str, Any]:
        ...
```

3.8.2 WorkGroupResource class

The *ucsschool.kelvin.client.WorkGroupResource* class has the following public attributes and methods:

```
class WorkGroupResource(KelvinResource):
    def __init__(self, session: Session, language: str = None):
        ...
    async def get(self, **kwargs) -> WorkGroup:
        ...
    async def get_from_url(self, url: str) -> WorkGroup:
        ...
    async def search(self, **kwargs) -> AsyncIterator[WorkGroup]:
        ...
```

3.8.3 Create workgroup

Workgroups can be created explicitly or implicitly when creating or modifying users.

workgroups will be automatically created when mentioned in a users `workgroups` attribute. They will however not be deleted automatically if they are removed from all users and are thus empty.

```
from ucsschool.kelvin.client import Session, WorkGroup

async with Session(**credentials) as session:
    wg = WorkGroup(
        name="testworkgroup",
        school="DEMOSCHOOL",
        description="A test workgroup",
        users=["demo_student", "demo_teacher"],
        create_share=True,
        session=session,
    )
    await wg.save()

wg.as_dict()
{'name': 'testworkgroup',
 'ucsschool_roles': ['workgroup:school:DEMOSCHOOL'],
 'school': 'DEMOSCHOOL',
 'description': 'A test workgroup',
 'users': ['demo_student', 'demo_teacher'],
 'create_share': True,
 'udm_properties': {},
 'dn': 'cn=DEMOSCHOOL-testworkgroup,cn=schueler,cn=groups,ou=DEMOSCHOOL,dc=example,
↪dc=com',
 'url': 'https://master.ucs.local/ucsschool/kelvin/v1/workgroups/DEMOSCHOOL/
↪testworkgroup'}
```

Workgroups are saved as groups in the UCS LDAP. The result can be verified on the target system using UDM:

```
$ udm groups/group list --filter cn=DEMOSCHOOL-testworkgroup

DN: cn=DEMOSCHOOL-testworkgroup,cn=schueler,cn=groups,ou=DEMOSCHOOL,dc=example,dc=com
  name: DEMOSCHOOL-testworkgroup
  description: A test workgroup
  ucsschoolRole: workgroup:school:DEMOSCHOOL
  users: uid=demo_student,cn=schueler,cn=users,ou=DEMOSCHOOL,dc=example,dc=com
  users: uid=demo_teacher,cn=lehrer,cn=users,ou=DEMOSCHOOL,dc=example,dc=com
  ...
```

Every workgroup has a share with the same name:

```
$  udm shares/share list --filter cn=DEMOSCHOOL-testworkgroup

DN: cn=DEMOSCHOOL-testworkgroup,cn=shares,ou=DEMOSCHOOL,dc=example,dc=com
  name: DEMOSCHOOL-testworkgroup
  host: DEMOSCHOOL.example.com
  path: /home/DEMOSCHOOL/groups/klassen/DEMOSCHOOL-testworkgroup
  directorymode: 0770
  group: 7110
  ...
```

Example creating two workgroups as a byproduct of creating a user:

```
from ucsschool.kelvin.client import Session, WorkGroupResource, User

async with Session(**credentials) as session:
    user = User(
        school="DEMOSCHOOL", schools=["DEMOSCHOOL"],
```

(continues on next page)

(continued from previous page)

```
        roles=["student"], name="test2",
        firstname="test", lastname="two",
        record_uid="test2", source_uid="TESTID",
        workgroups={"DEMOSCHOOL": ["workgroup1", "workgroup2"]},
        session=session)
    await user.save()

    async for wg in WorkGroupResource(session=session).search(school="DEMOSCHOOL"):
        print(sc)

WorkGroup('name'='workgroup1', 'school'='DEMOSCHOOL', dn='cn=DEMOSCHOOL-workgroup1,
↪cn=schueler,cn=groups,ou=DEMOSCHOOL,dc=example,dc=com')
WorkGroup('name'='workgroup2', 'school'='DEMOSCHOOL', dn='cn=DEMOSCHOOL-workgroup2,
↪cn=schueler,cn=groups,ou=DEMOSCHOOL,dc=example,dc=com')
WorkGroup('name'='Demoworkgroup', 'school'='DEMOSCHOOL', dn='cn=DEMOSCHOOL-
↪Demoworkgroup,cn=schueler,cn=groups,ou=DEMOSCHOOL,dc=example,dc=com')
```

3.8.4 Retrieve workgroup

It is necessary to pass both `name` and `school` arguments to the `get()` method, as the name alone wouldn't be unique in a domain (there can be workgroups of the same name in multiple schools).

```
from ucsschool.kelvin.client import Session, WorkGroupResource

async with Session(**credentials) as session:
    wg = await WorkGroupResource(session=session).get(
        school="DEMOSCHOOL", name="testworkgroup"
    )

wg.as_dict()
{'name': 'testworkgroup',
 'ucsschool_roles': ['workgroup:school:DEMOSCHOOL'],
 'school': 'DEMOSCHOOL',
 'description': 'A test workgroup',
 'users': ['demo_student', 'demo_teacher'],
 'create_share': True,
 'dn': 'cn=DEMOSCHOOL-testworkgroup,cn=schueler,cn=groups,ou=DEMOSCHOOL,dc=example,
↪dc=com',
 'url': 'https://10.200.3.70/ucsschool/kelvin/v1/workgroups/DEMOSCHOOL/testworkgroup'}
```

3.8.5 Check if workgroup exists

```
from ucsschool.kelvin.client import Session, WorkGroupResource

async with Session(**credentials) as session:
    if await WorkGroupResource(session=session).exists(school="DEMOSCHOOL", name=
↪"testworkgroup"):
        print("The workgroup exists!")
```

3.8.6 Search workgroups

The `search()` method allows searching for workgroups, filtering by `school` (mandatory) and `name` (optional).

The mandatory `school` argument must be exact while the optional `name` argument support an inexact search using `*` as a placeholder.

```
from ucsschool.kelvin.client import Session, WorkGroupResource

async with Session(**credentials) as session:
    async for wg in WorkGroupResource(session=session).search(school="DEMOSCHOOL"):
        print(sc)

WorkGroup('name'='Demoworkgroup', 'school'='DEMOSCHOOL', dn='cn=DEMOSCHOOL-
↪Demoworkgroup,cn=schueler,cn=groups,ou=DEMOSCHOOL,dc=example,dc=com')
WorkGroup('name'='testworkgroup', 'school'='DEMOSCHOOL', dn='cn=DEMOSCHOOL-
↪testworkgroup,cn=schueler,cn=groups,ou=DEMOSCHOOL,dc=example,dc=com')

    async for wg in WorkGroupResource(session=session).search(
        school="DEMOSCHOOL", name="test*"
    ):
        print(sc)

WorkGroup('name'='testworkgroup', 'school'='DEMOSCHOOL', dn='cn=DEMOSCHOOL-
↪testworkgroup,cn=schueler,cn=groups,ou=DEMOSCHOOL,dc=example,dc=com')
```

3.8.7 Change workgroup properties

Get the current workgroup object, change some attributes and save the changes back to LDAP:

```
from ucsschool.kelvin.client import Session, WorkGroupResource

async with Session(**credentials) as session:
    wg = await WorkGroupResource(session=session).get(
        school="DEMOSCHOOL",
        name="testworkgroup"
    )
    wg.description = "new description"
    wg.users.remove("demo_teacher")
    await wg.save()

wg.as_dict()
{'name': 'testworkgroup',
 'ucsschool_roles': ['workgroup:school:DEMOSCHOOL'],
 'school': 'DEMOSCHOOL',
 'description': 'new description',
 'users': ['demo_student'],
 'create_share': True,
 'dn': 'cn=DEMOSCHOOL-testworkgroup,cn=schueler,cn=groups,ou=DEMOSCHOOL,dc=example,
↪dc=com',
 'url': 'https://10.200.3.70/ucsschool/kelvin/v1/workgroups/DEMOSCHOOL/testworkgroup'}
```

3.8.8 Move workgroup

Workgroup objects do not support changing the `school`. Changing the `name` is allowed however.

```
from ucsschool.kelvin.client import Session, WorkGroupResource

```

(continues on next page)

(continued from previous page)

```
async with Session(**credentials) as session:
    wg = await WorkGroupResource(session=session).get(
        school="DEMOSCHOOL",
        name="testworkgroup"
    )
    wg.name = "testworkgroup-new"
    await wg.save()

wg.dn
'cn=DEMOSCHOOL-testworkgroup-new,cn,cn=schueler,cn=groups,ou=DEMOSCHOOL,dc=example,
↪dc=com'
```

3.8.9 Delete workgroup

Get the current workgroup object and delete it:

```
from ucsschool.kelvin.client import Session, WorkGroupResource

async with Session(**credentials) as session:
    wg = await WorkGroupResource(session=session).get(
        school="DEMOSCHOOL",
        name="testworkgroup"
    )
    await wg.delete()
```

3.9 Note on moving of objects

Moving an object means changing its position in LDAP. That happens whenever the DN changes. The DN is created from the name of the object concatenated with the subtree in which the object is located. So both changing a users or groups `name` attribute as well as changing an objects `school` attribute initiates a move.

School class objects do not support changing the school.

When the `school` attribute of a user is changed, the new value *must* be part of the list in the `schools` attribute.

CHAPTER 4

ucsschool.kelvin.client package

4.1 Submodules

4.1.1 ucsschool.kelvin.client.base module

class ucsschool.kelvin.client.base.**KelvinObject**(*, *name: str = None*, *ucsschool_roles: List[str] = None*, *udm_properties: Dict[str, Any] = None*, *dn: str = None*, *url: str = None*, *session: ucsschool.kelvin.client.session.Session = None*, *language: str = None*, ***kwargs*)

Bases: abc.ABC

as_dict() → Dict[str, Any]

delete() → None

reload() → KelvinObjectType

Reload properties of object from the Kelvin API.

Raises *ucsschool.kelvin.client.NoObject* – if the object cannot be found

Returns self

save() → KelvinObjectType

class ucsschool.kelvin.client.base.**KelvinResource**(*session: ucsschool.kelvin.client.session.Session*, *language: str = None*)

Bases: abc.ABC

class Meta

Bases: object

kelvin_object

alias of *KelvinObject*

required_get_attrs = ('name',)

required_head_attrs = ('name',)

required_search_attrs = ('school',)

exists(***kwargs*) → bool

get(***kwargs*) → KelvinObjectType

get_from_url(*url: str*) → KelvinObjectType

search(***kwargs*) → AsyncIterator[KelvinObjectType]

4.1.2 ucsschool.kelvin.client.exceptions module

exception ucsschool.kelvin.client.exceptions.**InvalidRequest**(*msg: str = None*, *status: int = None*, *reason: str = None*, *url: str = None*)

Bases: *ucsschool.kelvin.client.exceptions.KelvinClientError*

exception ucsschool.kelvin.client.exceptions.**InvalidToken**(*msg: str = None*, *status: int = None*, *reason: str = None*, *url: str = None*)

Bases: *ucsschool.kelvin.client.exceptions.KelvinClientError*

exception ucsschool.kelvin.client.exceptions.**KelvinClientError**(*msg: str = None*, *status: int = None*, *reason: str = None*, *url: str = None*)

Bases: Exception

exception ucsschool.kelvin.client.exceptions.**NoObject**(*msg: str = None*, *status: int = None*, *reason: str = None*, *url: str = None*)

Bases: *ucsschool.kelvin.client.exceptions.KelvinClientError*

exception ucsschool.kelvin.client.exceptions.**ServerError**(*msg: str = None*, *status: int = None*, *reason: str = None*, *url: str = None*)

Bases: *ucsschool.kelvin.client.exceptions.KelvinClientError*

4.1.3 ucsschool.kelvin.client.role module

class ucsschool.kelvin.client.role.**Role**(*name: str*, ***, *display_name: str = None*, *url: str = None*, *session: ucsschool.kelvin.client.session.Session = None*, *language: str = None*, ***kwargs*)

Bases: *ucsschool.kelvin.client.base.KelvinObject*

as_dict() → Dict[str, Any]

delete() → None

save() → ucsschool.kelvin.client.role.RoleResource

class ucsschool.kelvin.client.role.**RoleResource**(*session: ucsschool.kelvin.client.session.Session*, *language: str = None*)

Bases: *ucsschool.kelvin.client.base.KelvinResource*

class Meta

Bases: object

kelvin_object

alias of *Role*

required_get_attrs = ('name',)

required_head_attrs = ('name',)

required_search_attrs = ()

4.1.4 ucsschool.kelvin.client.school module

class ucsschool.kelvin.client.school.**School**(*name: str*, ***, *display_name: str = None*, *educational_servers: List[str] = None*, *administrative_servers: List[str] = None*, *class_share_file_server: str = None*, *home_share_file_server: str = None*, *ucsschool_roles: List[str] = None*, *udm_properties: Dict[str, Any] = None*, *dn: str = None*, *url: str = None*, *session: ucsschool.kelvin.client.session.Session = None*, *language: str = None*, ***kwargs*)

Bases: *ucsschool.kelvin.client.base.KelvinObject*

delete() → None

save() → ucsschool.kelvin.client.school.SchoolResource

class ucsschool.kelvin.client.school.**SchoolResource**(*session: ucsschool.kelvin.client.session.Session*, *language: str = None*)

Bases: *ucsschool.kelvin.client.base.KelvinResource*

class Meta

Bases: object

kelvin_object

alias of *School*

required_get_attrs = ('name',)

required_head_attrs = ('name',)

required_save_attrs = ('name',)

required_search_attrs = ()

4.1.5 ucsschool.kelvin.client.school_class module

class ucsschool.kelvin.client.school_class.**SchoolClass**(*name: str*, *school: str*, ***, *description: str = None*, *users: List[str] = None*, *create_share: bool = True*, *ucsschool_roles: List[str] = None*, *udm_properties: Dict[str, Any] = None*, *dn: str = None*, *url: str = None*, *session: ucsschool.kelvin.client.session.Session = None*, *language: str = None*, ***kwargs*)

Bases: *ucsschool.kelvin.client.base.KelvinObject*

class ucsschool.kelvin.client.school_class.**SchoolClassResource**(*session: ucsschool.kelvin.client.session.Session*, *language: str = None*)

Bases: *ucsschool.kelvin.client.base.KelvinResource*

class Meta

Bases: object

kelvin_object

alias of *SchoolClass*

required_get_attrs = ('name', 'school')

required_head_attrs = ('name',)

required_save_attrs = ('name', 'school')

required_search_attrs = ('school',)

4.1.6 ucsschool.kelvin.client.session module

exception ucsschool.kelvin.client.session.**BadSettingsWarning**

Bases: *ucsschool.kelvin.client.session.KelvinClientWarning*

exception ucsschool.kelvin.client.session.**KelvinClientWarning**

Bases: Warning

class ucsschool.kelvin.client.session.**Session**(*username: str*, *password: str*, *host: str*, *max_client_tasks: int = 10*, *request_id: str = None*, *request_id_header: str = 'X-Request-ID'*, *language: str = None*, ***kwargs*)

Bases: object

client

close() → None

delete(*url: str*, ***kwargs*) → None

get(*url: str*, ***kwargs*) → Union[Dict[str, Any], List[Dict[str, Any]]]

head(*url: str*, ***kwargs*) → bool

json_headers

open() → httpx.AsyncClient

post(*url: str*, ***kwargs*) → Dict[str, Any]

put(*url: str*, ***kwargs*) → Dict[str, Any]

request(*async_request_method: Any*, *url: str*, *return_json: bool = True*, ***kwargs*) → Union[str, int, Dict[str, Any]]

token

class ucsschool.kelvin.client.session.**Token**(*expiry: datetime.datetime*, *value: str*)

Bases: object

classmethod from_str(*token_str: str*) → ucsschool.kelvin.client.session.Token

is_valid() → bool

4.1.7 ucsschool.kelvin.client.user module

class ucsschool.kelvin.client.user.**PasswordsHashes**(*user_password: List[str]*, *samba_nt_password: str*, *krb_5_key: List[str]*, *krb5_key_version_number: int*, *samba_pwd_last_set: int*)

Bases: object

as_dict() → Dict[str, Any]

as_dict_with_ldap_attr_names() → Dict[str, Any]

Wrapper around *as_dict()* that renames the keys to those used in a UCS' OpenLDAP.

krb_5_key_as_bytes

Value of *krb_5_key* as a list of bytes.

class ucsschool.kelvin.client.user.**User**(*name: str = None*, *school: str = None*, ***, *firstname: str = None*, *lastname: str = None*, *birthday: datetime.date = None*, *disabled: bool = False*, *email: str = None*, *expiration_date: datetime.date = None*, *password: str = None*, *record_uid: str = None*, *roles: List[str]*, *schools: List[str]*, *school_classes: Dict[str, List[str]] = None*, *workgroups: Dict[str, List[str]] = None*, *source_uid: str = None*, *udm_properties: Dict[str, Any] = None*, *ucsschool_roles: List[str] = None*, *kelvin_password_hashes: ucsschool.kelvin.client.user.PasswordsHashes = None*, *dn: str = None*, *url: str = None*, *session: ucsschool.kelvin.client.session.Session = None*, *language: str = None*, ***kwargs*)

Bases: *ucsschool.kelvin.client.base.KelvinObject*

class ucsschool.kelvin.client.user.**UserResource**(*session: ucsschool.kelvin.client.session.Session*, *language: str = None*)

Bases: *ucsschool.kelvin.client.base.KelvinResource*

class Meta

Bases: object

kelvin_object
alias of *User*

required_get_attrs = ('name',)

required_head_attrs = ('name',)

required_save_attrs = ('school', 'roles')

required_search_attrs = ()

4.1.8 ucsschool.kelvin.client.workgroup module

class ucsschool.kelvin.client.workgroup.**WorkGroup**(*name: str*, *school: str*, ***, *description: str = None*, *users: List[str] = None*, *email: str = None*, *allowed_email_senders_users: List[str] = []*, *allowed_email_senders_groups: List[str] = []*, *create_share: bool = True*, *ucsschool_roles: List[str] = None*, *udm_properties: Dict[str, Any] = None*, *dn: str = None*, *url: str = None*, *session: ucsschool.kelvin.client.session.Session = None*, *language: str = None*)

Bases: *ucsschool.kelvin.client.base.KelvinObject*

class ucsschool.kelvin.client.workgroup.**WorkGroupResource**(*session: ucsschool.kelvin.client.session.Session*, *language: str = None*)

Bases: *ucsschool.kelvin.client.base.KelvinResource*

class Meta

Bases: object

kelvin_object
alias of *WorkGroup*

required_get_attrs = ('name', 'school')

required_head_attrs = ('name',)

required_save_attrs = ('name', 'school')

required_search_attrs = ('school',)

4.2 Module contents

class ucsschool.kelvin.client.**KelvinObject**(***, *name: str = None*, *ucsschool_roles: List[str] = None*, *udm_properties: Dict[str, Any] = None*, *dn: str = None*, *url: str = None*, *session: ucsschool.kelvin.client.session.Session = None*, *language: str = None*, ***kwargs*)

Bases: abc.ABC

as_dict() → Dict[str, Any]

delete() → None

reload() → KelvinObjectType
Reload properties of object from the Kelvin API.

Raises *ucsschool.kelvin.client.NoObject* – if the object cannot be found

Returns self

save() → KelvinObjectType

class ucsschool.kelvin.client.**KelvinResource**(*session: ucsschool.kelvin.client.session.Session*, *language: str = None*)

Bases: abc.ABC

class Meta
Bases: object

kelvin_object
alias of *KelvinObject*

required_get_attrs = ('name',)

required_head_attrs = ('name',)

required_search_attrs = ('school',)

exists(***kwargs*) → bool

get(***kwargs*) → KelvinObjectType

get_from_url(*url: str*) → KelvinObjectType

search(***kwargs*) → AsyncIterator[KelvinObjectType]

exception ucsschool.kelvin.client.**InvalidRequest**(*msg: str = None*, *status: int = None*, *reason: str = None*, *url: str = None*)

Bases: *ucsschool.kelvin.client.exceptions.KelvinClientError*

exception ucsschool.kelvin.client.**InvalidToken**(*msg: str = None*, *status: int = None*, *reason: str = None*, *url: str = None*)

Bases: *ucsschool.kelvin.client.exceptions.KelvinClientError*

exception ucsschool.kelvin.client.**KelvinClientError**(*msg: str = None*, *status: int = None*, *reason: str = None*, *url: str = None*)

Bases: Exception

exception ucsschool.kelvin.client.**NoObject**(*msg: str = None*, *status: int = None*, *reason: str = None*, *url: str = None*)

Bases: *ucsschool.kelvin.client.exceptions.KelvinClientError*

class ucsschool.kelvin.client.**PasswordsHashes**(*user_password: List[str]*, *samba_nt_password: str*, *krb_5_key: List[str]*, *krb5_key_version_number: int*, *samba_pwd_last_set: int*)

Bases: object

as_dict() → Dict[str, Any]

as_dict_with_ldap_attr_names() → Dict[str, Any]
Wrapper around *as_dict()* that renames the keys to those used in a UCS' OpenLDAP.

krb_5_key_as_bytes
Value of *krb_5_key* as a list of bytes.

exception ucsschool.kelvin.client.**ServerError**(*msg: str = None*, *status: int = None*, *reason: str = None*, *url: str = None*)

Bases: *ucsschool.kelvin.client.exceptions.KelvinClientError*

class ucsschool.kelvin.client.**School**(*name: str*, ***, *display_name: str = None*, *educational_servers: List[str] = None*, *administrative_servers: List[str] = None*, *class_share_file_server: str = None*, *home_share_file_server: str = None*, *ucsschool_roles: List[str] = None*, *udm_properties: Dict[str, Any] = None*, *dn: str = None*, *url: str = None*, *session: ucsschool.kelvin.client.session.Session = None*, *language: str = None*, ***kwargs*)

Bases: *ucsschool.kelvin.client.base.KelvinObject*

delete() → None

save() → ucsschool.kelvin.client.school.SchoolResource

class ucsschool.kelvin.client.**SchoolResource**(*session: ucsschool.kelvin.client.session.Session*, *language: str = None*)

Bases: *ucsschool.kelvin.client.base.KelvinResource*

class Meta

Bases: object

kelvin_object

alias of *School*

required_get_attrs = ('name',)

required_head_attrs = ('name',)

required_save_attrs = ('name',)

required_search_attrs = ()

class ucsschool.kelvin.client.**SchoolClass**(*name: str*, *school: str*, ***, *description: str = None*, *users: List[str] = None*, *create_share: bool = True*, *ucsschool_roles: List[str] = None*, *udm_properties: Dict[str, Any] = None*, *dn: str = None*, *url: str = None*, *session: ucsschool.kelvin.client.session.Session = None*, *language: str = None*, ***kwargs*)

Bases: *ucsschool.kelvin.client.base.KelvinObject*

class ucsschool.kelvin.client.**SchoolClassResource**(*session: ucsschool.kelvin.client.session.Session*, *language: str = None*)

Bases: *ucsschool.kelvin.client.base.KelvinResource*

class Meta

Bases: object

kelvin_object

alias of *SchoolClass*

required_get_attrs = ('name', 'school')

required_head_attrs = ('name',)

required_save_attrs = ('name', 'school')

required_search_attrs = ('school',)

class ucsschool.kelvin.client.**Session**(*username: str*, *password: str*, *host: str*, *max_client_tasks: int = 10*, *request_id: str = None*, *request_id_header: str = 'X-Request-ID'*, *language: str = None*, ***kwargs*)

Bases: object

client

close() → None

delete(*url: str*, ***kwargs*) → None

get(*url: str*, ***kwargs*) → Union[Dict[str, Any], List[Dict[str, Any]]]

head(*url: str*, ***kwargs*) → bool

json_headers

open() → httpx.AsyncClient

post(*url: str*, ***kwargs*) → Dict[str, Any]

put(*url: str*, ***kwargs*) → Dict[str, Any]

request(*async_request_method: Any*, *url: str*, *return_json: bool = True*, ***kwargs*) → Union[str, int, Dict[str, Any]]

token

class ucsschool.kelvin.client.**Role**(*name: str*, ***, *display_name: str = None*, *url: str = None*, *session: ucsschool.kelvin.client.session.Session = None*, *language: str = None*, ***kwargs*)

Bases: *ucsschool.kelvin.client.base.KelvinObject*

as_dict() → Dict[str, Any]

delete() → None

save() → ucsschool.kelvin.client.role.RoleResource

class ucsschool.kelvin.client.**RoleResource**(*session: ucsschool.kelvin.client.session.Session*, *language: str = None*)

Bases: *ucsschool.kelvin.client.base.KelvinResource*

class Meta

Bases: object

kelvin_object

alias of *Role*

required_get_attrs = ('name',)

required_head_attrs = ('name',)

required_search_attrs = ()

class ucsschool.kelvin.client.**User**(*name: str = None, school: str = None, *, firstname: str = None, lastname: str = None, birthday: datetime.date = None, disabled: bool = False, email: str = None, expiration_date: datetime.date = None, password: str = None, record_uid: str = None, roles: List[str], schools: List[str], school_classes: Dict[str, List[str]] = None, workgroups: Dict[str, List[str]] = None, source_uid: str = None, udm_properties: Dict[str, Any] = None, ucsschool_roles: List[str] = None, kelvin_password_hashes: ucsschool.kelvin.client.user.PasswordsHashes = None, dn: str = None, url: str = None, session: ucsschool.kelvin.client.session.Session = None, language: str = None, **kwargs*)

Bases: *ucsschool.kelvin.client.base.KelvinObject*

class ucsschool.kelvin.client.**UserResource**(*session: ucsschool.kelvin.client.session.Session, language: str = None*)

Bases: *ucsschool.kelvin.client.base.KelvinResource*

class Meta

Bases: object

kelvin_object

alias of *User*

required_get_attrs = ('name',)

required_head_attrs = ('name',)

required_save_attrs = ('school', 'roles')

required_search_attrs = ()

class ucsschool.kelvin.client.**WorkGroup**(*name: str, school: str, *, description: str = None, users: List[str] = None, email: str = None, allowed_email_senders_users: List[str] = [], allowed_email_senders_groups: List[str] = [], create_share: bool = True, ucsschool_roles: List[str] = None, udm_properties: Dict[str, Any] = None, dn: str = None, url: str = None, session: ucsschool.kelvin.client.session.Session = None, language: str = None*)

Bases: *ucsschool.kelvin.client.base.KelvinObject*

class ucsschool.kelvin.client.**WorkGroupResource**(*session: ucsschool.kelvin.client.session.Session, language: str = None*)

Bases: *ucsschool.kelvin.client.base.KelvinResource*

class Meta

Bases: object

kelvin_object

alias of *WorkGroup*

required_get_attrs = ('name', 'school')

required_head_attrs = ('name',)

required_save_attrs = ('name', 'school')

```
    required_search_attrs = ('school',)
```

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/univention/kelvin-rest-api-client/issues.

If you are reporting a bug, please include:

- The operating system name and version and the Python version where the *Kelvin REST API Client* was used.
- The UCS version of the server and installed apps (the output of `univention-app info`) to which the *Kelvin REST API Client* connected.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

Additionally look at bugs in the Univention Bugzilla in the product `Components` with component `kelvin-rest-api-client`: http://forge.univention.org/bugzilla/buglist.cgi?component=kelvin-rest-api-client&product=Components&resolution=—

5.1.3 Implement Features

Look through the GitHub issues and Univention Bugzilla for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

Kelvin REST API Client could always use more documentation, whether as part of the official *Kelvin REST API Client* docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/univention/kelvin-rest-api-client/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up `kelvin-rest-api-client` for local development.

1. Fork the `kelvin-rest-api-client` repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/kelvin-rest-api-client.git
```

3. Install your local copy into a virtualenv:

```
$ cd kelvin-rest-api-client/
$ make setup_devel_env
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass the style checks and the tests, including testing other Python versions with tox:

```
$ make lint
$ make test
$ make test-all
```

5.1 Fix format and coverage problems:

```
$ make format
$ make coverage-html
```

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. Make sure style and coverage requirements are met (run `make lint` and `tox`).
4. The pull request should work for Python 3.7, 3.8, 3.9 and 3.10. Check https://app.travis-ci.com/github/univention/kelvin-rest-api-client and https://github.com/univention/kelvin-rest-api-client/actions and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ python -m pytest tests/test_user.py::test_get_from_url
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ $EDITOR VERSION.txt
$ git add VERSION.txt
$ git commit -m "new version"
$ git tag "$(cat VERSION.txt)"
$ git push
$ git push --tags
```

CHAPTER 6

History

2.2.3 (2023-06-22)

- `%xx` escaped names of school classes, users and workgroups are now unescaped.

6.1 2.2.2 (2023-04-14)

- Support HEAD for `SchoolClass`, `User`, `WorkGroup`, and `Role`.

6.2 2.2.1 (2022-12-15)

- Use deepcopy in `to_dict` method to prevent values of `udm_properties` from being updated in objects which are copied.

6.3 2.2.0 (2022-10-13)

- Support Http `Accept-Language` Header.

6.4 2.1.0 (2022-10-07)

- Support HEAD for `School`.

6.5 2.0.1 (2022-10-05)

- Use detailed upstream error message in `InvalidRequest` exception messages.

6.6 2.0.0 (2022-09-10)

- **API Change**: The required argument `school` in the `User` constructor has now the default argument `None`. The argument `name` is not required anymore. Optional values, which are set to `None`, are not passed to the Kelvin server anymore. This enables automatic value generation on the Kelvin REST API server. To make use of this, the attributes can be either set to `None`, the empty string `""` or left out completely. Additionally, you have to create a schema for the corresponding attribute on the Kelvin REST API server.
- Send a correlation ID with each request.

6.7 1.7.1 (2022-08-30)

- Loosen dependency constraints.

6.8 1.7.0 (2022-07-07)

- Support user `workgroups` attribute.

6.9 1.6.1 (2022-06-30)

- Ignore unknown attributes in KelvinObject child classes.

6.10 1.6.0 (2022-06-27)

- Add support for workgroup resource.

6.11 1.5.2.1 (2022-04-05)

- Fixed: Logger does replace values of credentials with placeholders.

6.12 1.5.2 (2022-02-22)

- Automatic tests now run with Python 3.7 - 3.10.
- Fixed: The timeout attribute from a session instance is now used for requests.

6.13 1.5.1 (2021-11-30)

- Add attribute `expiration_date` to the `User` class. The attribute was added to the Kelvin REST API app in version `1.5.1`.

6.14 1.5.0 (2021-09-21)

- Add attribute `udm_properties` to classes `School` and `SchoolClass`. The attributes were added to the Kelvin REST API app in version `1.5.0`.

6.15 0.3.0 (2021-05-04)

- Add support for the creation of school (OU) objects.

6.16 0.2.2 (2020-11-09)

- Add support for the `kelvin_password_hashes` attribute of the `User` class.

6.17 0.2.1 (2020-08-07)

- fix JWT token validity calculation: timestamp uses UTC
- documentation fixes
- dependency updates
- tests also run on Python 3.9-dev

6.18 0.2.0 (2020-04-17)

- move tox to test requirements
- fix user object creation with default parameters
- change `as_dict` to be a method instead of a property
- fix flaky tests
- improve test coverage
- pass more env args to tox
- fix AttributeError with repr(role)
- add complete usage documentation

6.19 0.1.0 (2020-04-16)

- First release.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

u

Index

S

T

U

W