Sestco Documentation

Release 0.3

Christian Schur

Sep 05, 2017

Contents

CHAPTER 1

Installation and Overview: Connecting the University for Synchronization with the Servicestelle

Sestco connects university applications with the Servicestelle in a clean and powerful way. It takes under 5 minutes to set up a fresh installation and get connected with the Servicestelle.

1.1 Installation

The only requirement for the installation is a Python 2.7. Python is available for Windows, Linux, and Mac. On Windows, the location of the Python installation (e.g. `C:\Python34`) needs to be in `Path`. You can run `python --version` to check wether it can be found or if it needs to be added.

Having done that, create and activate a virtual environment:

```
# Create a virtual environment
virtualenv -p /usr/bin/python2 sestcoenv
# Activate the environment
source sestcoenv/bin/activate
```

If you're using Windows, to activate the virtualenv you'll need `sestcoenv\Scripts\activate`.

Install Sestco:

```
pip install https://konnektor.uni-mannheim.de/downloads/sestco-0.3.zip
```

Download a project template and unzip it:

https://konnektor.uni-mannheim.de/downloads/project-0.3.x.zip

Create a preconfigured project by running:

```
cookiecutter cookiecutter-sestco/
```

Answer the prompts with your own desired options. Press Ctrl+C to exit the process at any time.

In case you want to adjust any of those settings later, open the project's `settings.py` file in the editor of your choice and edit its *Sestco Settings*.

1.2 Usage

The following commands can be run in the project directory, e.g. by `python manage.py download_sest`. Remember to activate the virtual environment before, as shown in the installation instruction.

The following commands are available out of the box, without further custom implemention:

`download_sest` Download data from the Servicestelle.

`runserver` All locally available data is structured and can be browsed via a web browser at http://localhost.de:8000/.

After the implementation of a university specific module, the following commands become available:

`sync_uni` Synchronization of university data and Servicestelle data.

`rang_show_studienpakete` Shows Studienpakete created at the Servicestelle.

`rang_upload_schema` Upload a Rangliste schema.

`rang_upload_eintraege` Uploads Ranglisteneinträge.

Each command comes with a `--help` option that shows its correspondent help text. Before running any command, `download_sest` should be run to work on up-to-date data.

1.3 Configuration

If your application specific module has direct access to a university's database that doesn't support timezones, you might have to configure the timezone in the settings (see *TIME_ZONE*).

1.4 Implementing a university specific module (optional)

The range of features can be extended by university specific functionalities. In `proj/unispecific/` is a module, which inherits methods from the core module und supplies the base functions. In order to provide the university specific connection, the file `handlers.py` must be edited (in brackets is shown the reduced lines of code after the library has been opimized):

```
from sestco.pysest.handlers import BaseStudienangebotHandler, \
        BaseHandler, BaseBewerberHandler, \
        BaseBewerbungHandler, BaseRangHandler


# Lines: 200 (optimized: 150)
class BewerberHandler(BaseBewerberHandler):
    def update_localcopy(self):
        raise NotImplementedError()

    def update_university_data(self):
        raise NotImplementedError()

    def update_university_auth(self):
        raise NotImplementedError()


# Lines: 850 (optimized: 150)
class BewerbungHandler(BaseBewerbungHandler):
    def update_localcopy(self):
        raise NotImplementedError()

    def update_university(self):
        raise NotImplementedError()
```

```
    def update_servicestelle(self):
        raise NotImplementedError()


# Lines: 130 (optimized: 130)
class RangHandler(BaseRangHandler):
    def update_localcopy(self, studienpaket_schluessel):
        raise NotImplementedError()

    def get_eintrag_set(self, ranglistenschluessel, studienpaket):
        raise NotImplementedError()

    def update_university(self):
        raise NotImplementedError()


# Upload of Studienangebote will be implemented in the future.
class StudienangebotHandler(BaseStudienangebotHandler):
    pass


# UnterstuetzungHandler doesn't need any custom implementation so far.
class UnterstuetzungHandler(BaseUnterstuetzungHandler):
    pass
```

In the file `models.py` can be defined the intermediate storage of data. Here too can be inherited most of the functionalities from the core library.

CHAPTER 2

Usage

2.1 Adding data of the Servicestelle to the interface database

The interface runs its own database copy of the Servicestelle:

1. To be independent of temporary down times.
2. It makes it possible to check against the current data efficiently.
3. It allows to compare data coming from the Servicestelle with data held at university specific applications and to invoke synchronization manually.

To update the local database with the latest state of the Servicestelle, simply run

```
python manage.py download_sest
```

Warning: Deletion of objects is not supported by the SeSt webservice API yet, and therefore not considered here when looking up existent items.

Note: The `Dokument` items referenced by various fields in the Bewerbung, are not downloaded by default due to a bug in the Servicestelle webservice API. It does not return `Bewerbung` items of changed document references. This makes it impossible to partially update documents. The Sestco connector's aim is to make this missbehavior transparent and protect the user of the unexpected behavior. Because of that, document download is only active when the command is run to download *all* data from the Servicestelle and not only the recently changed one. This leads to data that is all up to date, and is achieved with the `--all` option:

```
python manage.py download_sest --all
```

When running the command on an empty database, it will collect all the missing data again. So, after the database has been deleted, it may be restored easily, though this may take some time then due to the amount of data.

The `Bewerbung` items are fetched in packages of the amount configured in `settings.py`.

For getting **help**, run:

```
python manage.py --help
```

For getting **help, on a specific command**, run `--help` with the command. This also works with commands supplied by other apps as well. For example:

```
python manage.py download_sest --help
```

2.2 Synchronizing Servicestelle data and university's data

Synchronize data of the university and the *Servicestelle*.

2.2.1 Running the console command

To transfer data from the university to the Servicestelle, and update its data with the latest state of the Servicestelle, run:

```
python manage.py download_sest
python manage.py sync_uni
```

If some items could not be sent properly, they will be kept in the queue. After two more retries have been made the item will be included in the summary mail sent to the system administrator.

> **Warning:** Before running this command, always run the `download_sest` command first, to update the local database with the latest state of the Servicestelle.

It is recommended to configure a cron job to run these commands regularly, e.g. every 5 minutes.

class sestco.pysest.management.commands.sync_uni.**Command**(*stdout=None*, *stderr=None*, *no_color=False*)

Expects class variables `bewerbung_handler` and `bewerbung_handler`, which reference a class of type `BewerberHandler` and `BewerbungHandler`.

2.3 Manage Ranglisten

Before uploading a Rangliste, somebody may want to know which **Studienpakete are available**, and which keys they have. This information is shown by running:

```
python manage.py rang_show_studienpakete
```

Note: Always run *sestco.pysest.management.commands.download_sest* before this command.

Run the command with the `--help` option for detailed information. Upload the **schema** of Ranglisten, as defined in the `settings.py` file (see *Sestco Settings*):

```
python manage.py rang_upload_schema
```

Note: Always run *sestco.pysest.management.commands.download_sest* before this command.

Upload the **entries** of Ranglisten:

```
python manage.py rang_upload_eintraege
```

Note: Always run *sestco.pysest.management.commands.download_sest* and *sestco.pysest.management.commands.sync_uni* before this command.

class sestco.pysest.management.commands.rang_upload_eintraege.**Command**(**args*, ***kwargs*)

Provide full functionality to upload Eintraege for a Rangliste to the Servicestelle. Subclasses must only implement a few methods to provide university specific data, without having to care about the workflow, or uploading data to the Servicestelle.

`rang_handler` is a class variable supplied by the university specific subclass. It must point to a university specific `Vermittlungsprozess` class.

handle(**args*, ***options*)

The controller of the command. A general one is implemented, so only override this method if a different one is needed.

2.4 Background information about the console commands

2.4.1 Interrupting the management command

The command takes the timestamp of the last processed item, or for a set of items if for each item it is not possible, and in case of an exception or break by `CTRL+C`, stores it to the database before quitting. This makes it possible to stop the command and beeing able to continue it later where it has stopped.

2.4.2 Consequence of deleting the local database

The command is designed in the way to store as less stateful data as possible. In fact, only the log data gets lost if the database is deleted. *The synchronization stores state data but only for efficiency purposes.* So, it is not fully dependend on them, and therefore not affected by a local database loss. When running the command on an empty database, it will fall back to alternative dates and collect all the missing data again before the next update process, though this may take some time due to the amount of data.

CHAPTER 3

Sestco Settings

3.1 Obligatory Settings

The following constants in the project's `settings.py` **must be defined**:

`sestco.conf.obligatory_settings.`**`SECO_WEBSERVICE = {}`**
A dictionary like so:

```
SECO_WEBSERVICE = {
     'username': 'my_sest_webservice_username',
     'password': 'my_sest_webservice_password',
     'location': 'http://thedomain:theport/thepath',
 }
```

`sestco.conf.obligatory_settings.`**`SECO_HOCHSCHULNUMMER = ''`**
The identification number of the Hochschule using the interface.

`sestco.conf.obligatory_settings.`**`SECO_VERMITTLUNGSPROZESS = ()`**
The Vermittlungsprozess of the Serviceverfahren. Example:

```
SECO_VERMITTLUNGSPROZESS = {
    'jahr': 2018,
    'semester': 'WS',
    'typ': 'Koordinierung',
}
```

`sestco.conf.obligatory_settings.`**`SECO_HANDLERS = ''`**
(conditionally obligatory)

A string that represents a module with handler classes:

```
SECO_HANDLERS = 'path.to.handler_module'
```

Those classes implement the university specific methods of a `BaseHandler`. The module file can be placed in a module file of the university specific app.

`sestco.conf.obligatory_settings.`**`SECO_VERGABESCHEMA = ()`**
(conditionally obligatory)

(obligatory when using :mod:'sestco.pysest.management.command.rang_upload_schema' command)

For uploading and downloading `Rangliste` informations, the application goes through the list of available `Rangliste` types. Those are configured via the Servicestelle's Web–GUI, but cannot be retrieved automatically via the webservice. Note that the order defines the position for each *Ebene* and *Rangliste* type, and must match the ones of the Web–GUI. The `SECO_VERGABESCHEMA` is reduced to the informations needed to transfer the `Rangliste`:

```
SECO_VERGABESCHEMA = (
    ('Vorabquoten', (
        (id_nr, name, options_dict),
        (id_nr, name, options_dict),
        # ...
    )),
    ('geminderte_Vorabquoten', (
        (id_nr, name, options_dict),
        (id_nr, name, options_dict),
        # ...
    )),
    ('Hauptquoten', (
        (id_nr, name, options_dict),
        (id_nr, name, options_dict),
        # ...
    )),
)
```

For `id_nr` only numbers or characters of the ASCII character set are allowed, and no special characters. Otherwise, a `UnicodeEncodeError` exception is thrown.

`options_dict` is a dictionary of options. For each entry of a Rangliste type (e.g. `'Vorabquoten'`), the following fields can be set (example showing all possible fields):

```
(id_nr, name, {
    # plaetze: Either 'absolut' or 'prozentual' is allowed.
    'plaetze': ('prozentual', 2.0),
    'klauseln': {
        # The entrie's value of a *Klausel* is ``True`` or a
        # dictionary of subfields.
        'Binnenquotenklausel': {
            'binnenquotenkennung': u'my_kennung',
            'ist_binnenquotendachrangliste': True,
        },
        'Mindestplatzklausel': {
            'grenzkapazitaet': 20,
        },
        # If not set, the value defaults to ``False``.
        #'Limitierungsklausel': True,
        'Unterbelegungsklausel': {
            # A tuple of (object_type, id_name). object_type can
            # be either 'ebene' or 'ranglisten_type', id_name is
            # its name, e.g. 'Hauptquoten' or 66.
            'platzuebergabe': ('ebene', 'Hauptquoten'),
        },
        'Nichtauffuellklausel': True,
    },
    'ist_chance': True,
    # Optional and only allowed for level 'Vorabquoten'.
    'ist_vorwegzulasser': True,
}).
```

3.2 Optional Settings

The following configuration constants are **optional** in the project's `settings.py`:

sestco.conf.optional_settings.**SECO_ABSCHLUSS_SCHLUESSEL = ()**
A tuple of Abschlussschlüssel (strings). Updating data at the Servicestelle is limited to the Abschluesse given here. Example:

```
SECO_ABSCHLUSS_SCHLUESSEL = ('82',)
```

sestco.conf.optional_settings.**SECO_FACH_SCHLUESSEL = ()**
For some functionalities, the items can be restricted to a Fach. Example:

```
SECO_FACH_SCHLUESSEL = ('132',)
```

If not set, all of the Bewerbung items at the Servicestelle are fetched.

sestco.conf.optional_settings.**SECO_STUDIENANGEBOT_DOWNLOAD = True**
A switch to turn off the download of Studienangebot. This is turned on by default, because there is no way to find out automatically if the Studienangebot items have changed at the Servicestelle. Although changes to the Studienangebot are quite rare practically, there is technically the possibility. The download can be turned off, but it should be made sure the administrator of the connector is informed about updates by the manager of the Studienangebot to switch back to download mode.

sestco.conf.optional_settings.**SECO_BEARBEITUNGSSTATUS_IN_OUT = {}**
A dictionary that overwrites specific bearbeitungsstatus flow possiblities for the Hochschule, which is by default as defined in the `Bewerbung` model. The tuples are having booleans or numbers evaluated like `(sest_in, sest_out, uni_in, uni_out)`. The default values are at `sestco.pysest.models.Bewerbung`. Example:

```
SECO_BEARBEITUNGSSTATUS_IN_OUT = {
    # Overwrites (1, 1, 1, 1).
    'in_Vorbereitung': (0, 1, 1, 1),
    # Overwrites (1, 1, 0, 1).
    'zurueckgezogen': (1, 0, 0, 1),
    # Overwrites (1, 0, 1, 1).
    'ausgeschlossen': (1, 0, 0, 1),
}
```

Warning: You should check those values and make sure they are adjusted to the specific case of the Hochschule.

sestco.conf.optional_settings.**SECO_PROXY = None**
A dictionary of `'protocol': 'http/the.url.com'` pairs, that becomes necessary if the project is on a machine that is forced to use a proxy server for outgoing connections. Example:

```
SECO_PROXY = {
    'https': 'http://www-cache.example.com:3128',
    'http': 'http://www-cache.example.com:3128',
    'ftp': 'http://www-cache.example.com:3128',
}
```

sestco.conf.optional_settings.**SECO_DEBUG_BEWERBER = False**
This variable is only relevant when developing and currently not part of the official configuration settings and should not be treated as so. It is used for internal development to allow Django having in debug mode while some handlers acting like in productive environment.

sestco.conf.optional_settings.**SECO_DEBUG_BEWERBUNG = False**
This variable is only relevant when developing and currently not part of the official configuration settings and should not be treated as so. It is used for internal development to allow Django having in debug mode while some handlers acting like in productive environment.

sestco.conf.optional_settings.**SECO_DEBUG_RANG = False**
This variable is only relevant when developing and currently not part of the official configuration settings

and should not be treated as so. It is used for internal development to allow Django having in debug mode while some handlers acting like in productive environment.

`sestco.conf.optional_settings.`**`SECO_DOKUMENT_DOWNLOAD = False`**
A switch to turn on or off file download. If set to `False`, files will be linked to but not downloaded during synchronization process. Once set to `True`, the missing files will be downloaded.

See also *SECO_DOKUMENT_ROOT*.

`sestco.conf.optional_settings.`**`SECO_DOKUMENT_ROOT = ''`**
Absolute filesystem path to the directory that will hold downloaded files.

Example: `"/var/www/unimaseco/dokumente/"`.

See also *SECO_DOKUMENT_DOWNLOAD*.

`sestco.conf.optional_settings.`**`SECO_NO_SSL_CERTIFICATE_VERIFICATION = False`**
Disable verification of HTTPS certificates. You might want to set this to `True` in case there is a problem with the certificate verification on the server. Using this option in production is highly discouraged.

3.3 Django Settings

These configuration variables are part of the Django framework itself but provided with additional information here.

3.3.1 TIME_ZONE

If your application specific module has direct access to a university's database that doesn't support timezones, you might have to configure the time zone in the settings. This makes sure the values are stored in the correct time according to the implicit timezone of that database. For example, if its implicit timezone is Berlin, set:

```
TIME_ZONE = 'Europe/Berlin'

USE_TZ = True
```

3.3.2 LOGGING

Apart from introducing as a strict definition of log levels that is explained here, setting up logging may be confusing at first due to its mature and flexible system, so an example setup is given here.

For filtering log messages, the logger name can be used, and the severity levels. While the first is about the place / type of action, the severity level allows to react on different types of, well, severity. An administrator can configure logging so it only picks up the logs of a specific logger (which has a hierarichal name of dot seperated syntax), having the name of a module or more general name. He can then filter for severity levels, for example to log all levels including debug informations with detailed descriptions to a file, print out warnings on console, and send errors by email. Additionally he can define the formatting, for example with detailed informations for file and more simple for console.

Severity levels

DEBUG For low level system informations like variable data.

INFO Status informations about the current process that could be interesting for end users as additional informations, e.g. what is or will be happening in the next step, or what has just been done.

WARNING Issued for situations that may be problematic but do not cause the program to terminate prematurely. The data produced may or may not be reliable, but should be examined carefully.

ERROR Used for serious problems that cause the program to exit.

CRITICAL This severity level is not used as it is difficult to differentiate between several error terms, and may not be clear to the user.

Example setup

The example given here is sufficient to get logging up running, for detailed informations about logging see the Django documentation about logging.

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'formatters': {
        'verbose': {
            'format': (
                '%(levelname)s %(asctime)s %(module)s %(funcName)s '
                '%(lineno)d %(process)d %(thread)d %(message)s'),
        },
        'simple': {
            'format': '%(levelname)s %(message)s',
        },
    },
    'handlers': {
        'mail_admins': {
            'level': 'ERROR',
            'class': 'django.utils.log.AdminEmailHandler',
        },
        'console':{
            'level':'INFO',
            'class':'logging.StreamHandler',
            'formatter': 'simple',
        },
        'file':{
            'level':'DEBUG',
            'class':'logging.FileHandler',
            'filename': 'debug.log',
            'formatter': 'verbose',
        },
    },
    'loggers': {
        'django': {
            'handlers': ['file'],
            # Use ``'DEBUG'`` to see database info and others more.
            #'level': 'DEBUG',
            'level': 'INFO',
            'propagate': True,
        },
        'django.request': {
            'handlers': ['mail_admins'],
            'level': 'ERROR',
            # Use ``False`` to not handle the log messages handled by
            # this logger also by the 'django' logger.
            #'propagate': False,
            'propagate': True,
        },
        'sestco.pysest': {
            'handlers': ['console', 'file', 'mail_admins'],
            'level': 'DEBUG',
            'propagate': True,
        },
        # Use ``'maapp.handlers'`` to narrow the set of loggers.
        'maapp': {
            'handlers': ['console', 'file', 'mail_admins'],
            'level': 'DEBUG',
            'propagate': True,
```

```
        },
        'suds.client': {
            'handlers': ['file'],
            'level': 'DEBUG',
            'propagate': True,
        },
    },
}
```

CHAPTER 4

Developing a university specific module

4.1 A local representation of the Servicestelle: `pysest`

The handler API provides **classes that handle as much work as possible** for synchronizing a university's data with the Servicestelle. They handle the workflow, command line user interaction, logging and much more. By subclassing them and implementing some obligatory methods that provide case specific data it is possible to get most of the work done with just a few lines of code. This kind of high level API is what the first chapter is about.

In case the local database copy of the Servicestelle data needs to be accessed, or when creating a custom workflow, some more knowledge about the lower level API is needed. This API is still a great support to interact with the interface's components and the Servicestelle webservice. To write an app (module) from scratch three main areas of background knowledge must be known: How to **write a Django module**, the structure of the SfH mapping **database model** for querying against locally stored data, and the handling of the SfH webservice **client wrapper**, which simplifies the communication with the Servicestelle webservice.

4.1.1 Handlers

Introduction

Handlers bundle the core functionality of the synchronization process. They provide a flexible way to add university specific tasks. The handler system consists of university handlers in a seperate app (module), that inherit from the base handlers in *`sestco.pysest.handlers`*. The university specific handler file will be found by being registerd in the `settings.py` file of the project that makes use of the app (see *Sestco Settings*)

In the handlers live four types of methods:

1. Methods that must be implemented by subclasses:
    - *`BaseBewerberHandler.update_localcopy()`*
    - *`BaseBewerberHandler.update_university_data()`*
    - *`BaseBewerberHandler.update_university_auth()`*
    - *`BaseBewerberHandler.cleanup_bid_auths()`*
    - *`BaseBewerbungHandler.update_localcopy()`*
    - *`BaseBewerbungHandler.update_university()`*

- *BaseBewerbungHandler.update_servicestelle()*
- *BaseRangHandler.update_localcopy()*
- *BaseRangHandler.get_eintrag_set()*
- *BaseRangHandler.update_university()*

It must be admitted that those methods could be abstracted much more (and being replaced by more specific ones to be overwritten), but it was the idea to make few assumptions about the university system. There might be implemented a seperate app (module) that surves these methods making more concrete assumptions about the subclassing handler, reducing the coding that must be implemented by inheriting handlers down to approximately 200 lines in the future.

2. Methods that *can* be *overwritten* by sub classes:
   - *BaseUnterstuetzungHandler.get_dokument_filename()*
3. Methods meant to be called from outside the object. Those are:
   - *BaseStudienangebotHandler.download_sest()*
   - *BaseBewerberHandler.download_sest()*
   - *BaseBewerberHandler.update()*
   - *BaseBewerbungHandler.download_sest()*
   - *BaseBewerbungHandler.update()*
   - *BaseRangHandler.download_sest()*
   - *BaseRangHandler.get_rangliste_set()*
   - *BaseRangHandler.upload_eintraege()*
   - *BaseRangHandler.update_university()*
   - *BaseUnterstuetzungHandler.download_dokumente()*

   Apart from those, each base handler inherits two methods, that are meant to be started before and after a whole synchronization process. They can be overwritten by subclasses, but only by calling the parents' method as well:
   - *BaseHandler.set_up()*
   - *BaseHandler.tear_down()*

   These methods are invoked by the management commands that are already part of this module, so there is no work to do on these. It is mentioned here to explain how the system works, and in case there is the need to customize the provided management commands.
4. Private API, e.g. helper methods. There are many of them, and none of them is relevant for subclasses. Not being part of the public API, these methods may change without a deprecation path.

All handler classes inherit instance attributes that are useful for managing synchronization with the Servicestelle:

- `BaseHandler._sest`
- `BaseHandler._vermittlungsprozess_typ`
- `BaseHandler._serviceverfahren_jahr`
- `BaseHandler._serviceverfahren_semester`
- `BaseHandler._serviceverfahren`
- `BaseHandler._einfachabschluess_schluessel`
- `BaseHandler._mehrfachabschluess_schluessel`
- `BaseHandler._amount`
- `BaseHandler.startdate_main`

Summarizing, all that needs to be implemented are less than ten methods, placed in five classes, in one file, that is addressed in `SECO_HANDLERS`. All together it takes around 1500 lines of code, including comments and doc strings, and as mentioned above, they could be reduced to approximately 200 lines of code by a more specific middle handler that might be implemented in the future.

class sestco.pysest.handlers.**BaseHandler**

The BaseHandler class. If *jahr*, *semester* and *vermittlungsprozess_typ* is `None` (default), the current Vermittlungsprozess is detected automatically. There is no need to inherit this class, but it shows methods and attributes available in subclasses.

Variables

- **_sest** – A Servicestelle client that allows to call webservice methods directly from Python code. See *sestco.sestclient.ServiceStelle*.
- **_hochschulnummer** – The Hochschulnummer of the current Vermittlungsprozess.
- **_vermittlungsprozess_typ** – The Vermittlungsprozesstyp of the current Vermittlungsprozess.
- **_serviceverfahren_jahr** – The year of the Serviceverfahren of the current Vermittlungsprozess.
- **_serviceverfahren_semester** – The semester of the Serviceverfahren of the current Vermittlungsprozess.
- **_serviceverfahren** – The serviceverfahren of the current Vermittlungsprozess, an object of type `sestco.pysest.models.sest.Serviceverfahren`.
- **_mehrfachabschluess_schluessel** – All Mehrfachabschlussschluessel of all Studienangebote at the Servicestelle.
- **_einfachabschluess_schluessel** – All Einfachabschlussschluessel of all Studienangebote at the Servicestelle.
- **_amount** – The amount of items being fetched or sent via the webservice in a group. Will be configurable for each webservice method in the future.
- **start_date** – Time when the whole synchronization process covering several handlers has started.

set_up(*start_date*)

Set up actions that need to be performed once for a cicle of methods, setting *start_date* as the time that is considered to be the start time of a *whole* synchronization process. The start time must be defined and set from outside the handler object, as a process can start before a handler is instanciated, and is the same for all handlers during a synchronization process.

tear_down()

Tear down actions that need to be performed once after a cicle of methods.

set_attr(*to_obj*, *to_field*, *from_obj*, *from_field*, *tolerate_missing=True*, *set_default=False*, *default=None*)

Set the attribute of name *to_field* to *to_obj*, if the *from_obj* object's field *from_field* is set and not `None`. If *tolerate_missing* is `False`, the error handling is left to the standard python way, if `True`, a debug entry is being logged. In case there is no field, or the field is `None`, a *default* value is set if *set_default* is `True`. (There are two fields to set a default value to be able to set a `None` value as default.)

get_timezone_aware(*datetime_obj*)

Fixes time zone deficiencies, e.g. Suds': https://fedorahosted.org/suds/ticket/353.

get_or_create_bewerber(*bewerberid*)

Retrieve and return a `sestco.pysest.models.sest.Bewerber` object having *bewerberid* from the database, or create one and return it, if not existent yet.

update_and_save_bewerber(*bewerber*, *bewerber_ws*)
Update *bewerber*, a `sestco.pysest.models.sest.Bewerber` object, with the data of *bewerber_ws*, a data object received by the werbservice.

get_or_create_land(*land_ws*)
Return the `Land` object from the database with name of `suds` Webservice object *land_ws*, and if not existent, create one in database and return it.

get_einfachstudienangebot(*einfachstudienangebotsSchluessel*)
Return the `sestco.pysest.models.sest.EinfachStudienangebot` having key *einfachstudienangebotsSchluessel* from database. *einfachstudienangebotsSchluessel* is the key as returned by the SOAP webservice.

The handler classes

class sestco.pysest.handlers.**BaseHandler**
The BaseHandler class. If *jahr*, *semester* and *vermittlungsprozess_typ* is `None` (default), the current Vermittlungsprozess is detected automatically. There is no need to inherit this class, but it shows methods and attributes available in subclasses.

Variables

- **_sest** – A Servicestelle client that allows to call webservice methods directly from Python code. See *sestco.sestclient.ServiceStelle*.
- **_hochschulnummer** – The Hochschulnummer of the current Vermittlungsprozess.
- **_vermittlungsprozess_typ** – The Vermittlungsprozesstyp of the current Vermittlungsprozess.
- **_serviceverfahren_jahr** – The year of the Serviceverfahren of the current Vermittlungsprozess.
- **_serviceverfahren_semester** – The semester of the Serviceverfahren of the current Vermittlungsprozess.
- **_serviceverfahren** – The serviceverfahren of the current Vermittlungsprozess, an object of type `sestco.pysest.models.sest.Serviceverfahren`.
- **_mehrfachabschluess_schluessel** – All Mehrfachabschlussschluessel of all Studienangebote at the Servicestelle.
- **_einfachabschluess_schluessel** – All Einfachabschlussschluessel of all Studienangebote at the Servicestelle.
- **_amount** – The amount of items being fetched or sent via the webservice in a group. Will be configurable for each webservice method in the future.
- **start_date** – Time when the whole synchronization process covering several handlers has started.

set_up(*start_date*)
Set up actions that need to be performed once for a cicle of methods, setting *start_date* as the time that is considered to be the start time of a *whole* synchronization process. The start time must be defined and set from outside the handler object, as a process can start before a handler is instanciated, and is the same for all handlers during a synchronization process.

tear_down()
Tear down actions that need to be performed once after a cicle of methods.

set_attr(*to_obj*, *to_field*, *from_obj*, *from_field*, *tolerate_missing=True*, *set_default=False*, *default=None*)
Set the attribute of name *to_field* to *to_obj*, if the *from_obj* object's field *from_field* is set and not `None`. If *tolerate_missing* is `False`, the error handling is left to the standard python way, if `True`, a debug entry is being logged. In case there is no field, or the field is `None`, a *default* value is set

if *set_default* is `True`. (There are two fields to set a default value to be able to set a `None` value as default.)

get_timezone_aware(*datetime_obj*)
Fixes time zone deficiencies, e.g. Suds': https://fedorahosted.org/suds/ticket/353.

get_or_create_bewerber(*bewerberid*)
Retrieve and return a `sestco.pysest.models.sest.Bewerber` object having *bewerberid* from the database, or create one and return it, if not existent yet.

update_and_save_bewerber(*bewerber*, *bewerber_ws*)
Update *bewerber*, a `sestco.pysest.models.sest.Bewerber` object, with the data of *bewerber_ws*, a data object received by the werbservice.

get_or_create_land(*land_ws*)
Return the `Land` object from the database with name of `suds` Webservice object *land_ws*, and if not existent, create one in database and return it.

get_einfachstudienangebot(*einfachstudienangebotsSchluessel*)
Return the `sestco.pysest.models.sest.EinfachStudienangebot` having key *einfachstudienangebotsSchluessel* from database. *einfachstudienangebotsSchluessel* is the key as returned by the SOAP webservice.

class sestco.pysest.handlers.**BaseStudienangebotHandler**

download_sest()
Downlaod Studienangebot related data from Servicestelle and save it to the local copy.

create_or_update_mehrfachstudienangebot(*studienangebot_ws*)
Create a `sestco.pysest.models.sest.MehrfachStudienangebot`, or update an existent one. *studienangebot_ws* is the Studienangebot data as received from the webservice.

create_or_update_studiengang(*studiengang_ws*)
Create a `sestco.pysest.models.sest.Studiengang`, or update an existent one. *studiengang_ws* is the Studiengang data as received from the webservice.

get_and_update_or_create_abschluss(*abschluss_ws*)
Retrieve and return a `sestco.pysest.models.sest.Abschluss` object having data *abschluss_ws* from the database, or create one and return it, if not existent yet. *abschluss_ws* is the Abschluss data as received from the webservice.

add_general_studienangebot_fields(*studienangebot*, *studienangebot_ws*)
Add the data of *studienangebot_ws* webservice to *studienangebot*, a `sestco.pysest.models.sest.Studienangebot` object.

class sestco.pysest.handlers.**BaseBewerberHandler**

download_sest()
Downlaod Bewerber related data from Servicestelle and save it to the local copy.

update()
Update localcopy and university data. This method should be used for the update process externally.

update_localcopy()
Update the localcopy with the university's data.

Subclasses must implement this method.

update_university_data()
Update the university's data Servicestelle's data.

Subclasses must implement this method.

update_university_auth()
Authenticate the university user in the local copy against the Servicestelle, and write the result to the local copy and to the university's database.

Subclasses must implement this method.

cleanup_bid_auths()
Remove Bewerber IDs that are not needed any more, e.g. because there are no Bewerbung items pending for being sent to Servicestelle, in the university's database.

Subclasses must implement this method.

update_university()
Update university data.

class sestco.pysest.handlers.**BaseBewerbungHandler**

download_sest()
Downlaod Bewerbung related data from Servicestelle and save it to the local copy.

> **Note:** Always run *BaseBewerberHandler.download_sest()* and *BaseStudienangebotHandler.download_sest()* before this method.

update()
Run whole process to update Bewerbung items at the localcopy, at the university, and at the Servicestelle.

update_localcopy()
Update the localcopy with the university's data.

Subclasses must implement this method.

update_university()
Update the university with the Servicestelle's data.

Subclasses must implement this method.

update_servicestelle()
Send university's data in the localcopy to the Servicestelle.

Subclasses must implement this method.

create_or_update_einfachstudienangebotsbewerbung(*bewerbung_ws*)
Create a `sestco.pysest.models.sest.Einfachstudienangebot`, or update an existent one. *bewerbung_ws* is the Einfachstudienangebotsbewerbung data as received from the webservice.

add_general_bewerbung_fields(*bewerbung*, *bewerbung_ws*, *bewerber*)
Add the data of *bewerbung_ws* webservice and *bewerber*, a `sestco.pysest.models.sest.Bewerber` object, to *bewerbung*, a `sestco.pysest.models.sest.EinfachstudienangebotsBewerbung` or `sestco.pysest.models.sest.MehrfachstudienangebotsBewerbung` object.

update_general_bewerbung_fields(*bewerbung*, *bewerbung_ws*)
Update *bewerbung*, a `sestco.pysest.models.sestco.pysest.EinfachstudienangebotsBewer' 'bung` or `sestco.pysest.models.sestco.pysest.MehrfachstudienangebotsBewe' 'rbung`, with data of *bewerbung_ws*, a webservice object.

> **Warning:** This method has the side effect of saving the *bewerbung* in the middle of the process. It must be saved afterwards again, though.

set_or_create_dokument(*obj*, *obj_ws*)
Attach a document to *obj*, or change the attachment to another document if it differs from the one of *obj_ws*.

Nothing is done if the document of *obj* and *obj_ws* is the same. Creates the document if it doesn't exist yet.

class sestco.pysest.handlers.**BaseRangHandler**

download_sest()
Downlaod Rangliste related data from Servicestelle and save it to the local copy.

Note: Always run *BaseBewerberHandler.download_sest()* and *BaseStudienangebotHandler.download_sest()* before this method.

get_rangliste_set(*studienpaket_schluessel*, *get_eintrag_set_func=None*)
Return a list of items in serialized Python data for *studienpaket_schluessel*. Use *get_eintrag_set()* for receiving each entry, or *get_eintrag_set_func*.

upload_eintraege(*rangliste_set*, *kopfdaten_beibehalten=True*)
Update the Ranglisten named via the command line parameters with their entries provided by *get_eintrag_set()*, and their configuration data provided by *settings.py*, if *kopfdaten_beibehalten* is set to `False`.

update_localcopy(*studienpaket_schluessel*)
Update the local copy with university's data for the `get_rang_statistics()` method and *get_eintrag_set()* method to retrieve the data from.

Subclasses must implement this method.

get_eintrag_set(*ranglistenschluessel*, *studienpaket*)
Return an iterable like list or iterator object of `(bewerberid, rang)` tuples to upload for a given Rangliste.

Subclasses must implement this method.

update_university()
Update the university's rangliste's `statusanmerkung` field. This method transfers the result of the Servicestelle's rangliste back to the university.

Subclasses must implement this method.

get_rangliste_set_ws(*rangliste_set*, *kopfdaten_beibehalten=True*, *transmit_klauseln=True*)
Return a list of Rangliste items formatted for the `sestco.sestclient.client` webservice client wrapper.

class sestco.pysest.handlers.**BaseUnterstuetzungHandler**

DOKUMENT_AMOUNT = 10
The amount of `Dokument` items for each package to be downloaded at once in a request.

download_dokumente()
Download Dokumente from the Servicestelle in packages and save them to the file system to *SECO_DOKUMENT_ROOT*.

get_dokument_filename(*dokument*)
Return a filename (without extension) for *dokument*.

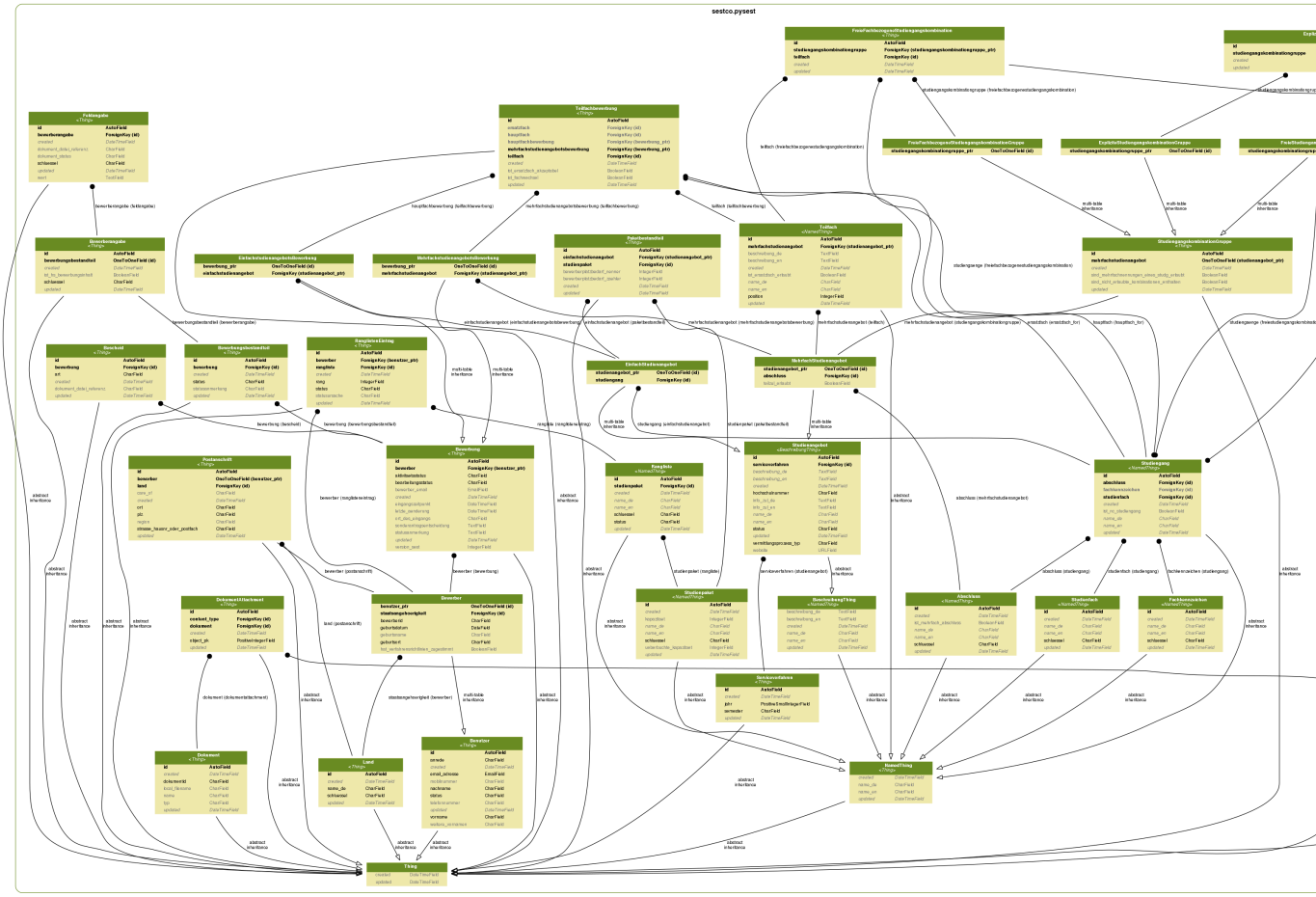
The file name is used to store *dokument* to the file system. Mostly a hook, this method can be overwritten in case a custom file name schema should be used. Via *dokument*, being of type `Dokument`, related values can be accessed to compose a file name, for example the original file name, the `dokumentid`, the `Feldangabe.schluessel`, the *Bewerbung* or `Bewerber` or any other. Currently, there is no support for the file name schema to be customizable via the settings, but it can be implemented if needed.

By default, this method creates a file name of the schema `{sest_bewerberid}__{sest_documentid}`. If the `Dokument` is not attached to a `Feldangabe`, it falls back to the more simple schema `{sest_documentid}`.

4.1.2 The Servicestelle interface mapping database model

The interface comes with an exact copy of the Servicestelle data, and modules can assume it is up to date, sincse the user is asked to run *`sestco.pysest.management.commands.download_sest`* before running any other command that relies on Servicestelle data. The relations between the classes are reconstructed during the update process, and objects are linked with each other according to it. Having the possibility of accessing the Servicestelle data locally in an object oriented fashion is very convenient.

The following graph shows the models, their relations and attributes:



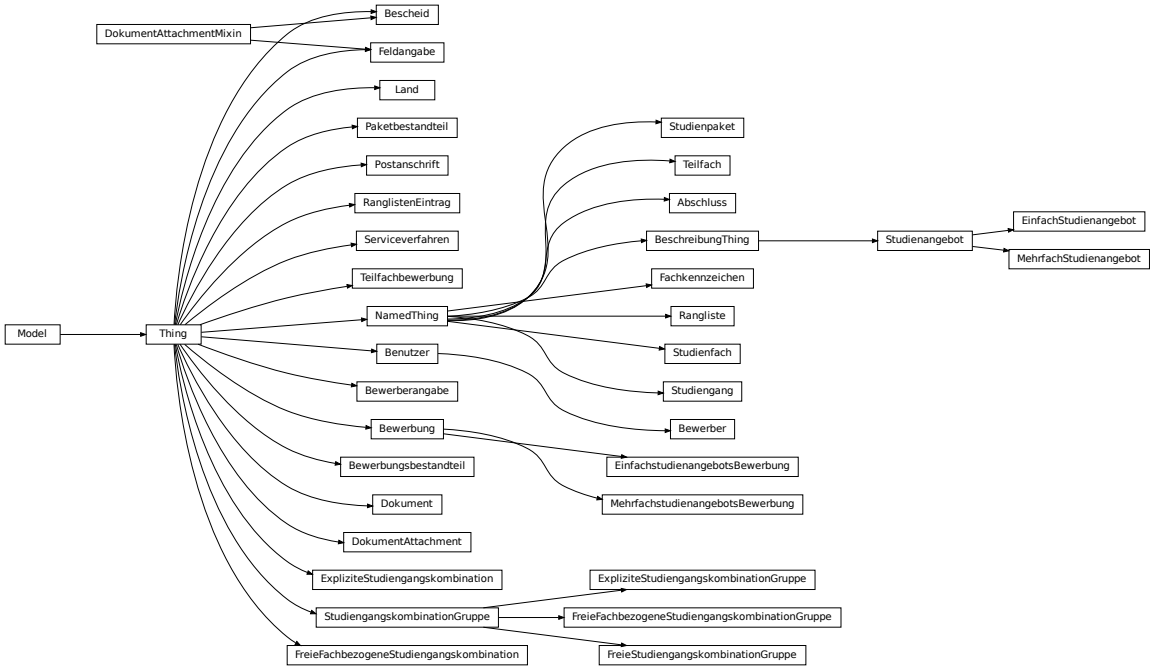
These models mirror the SfH data the university has access to. While it is meant for reading data from it, creating or updating entries is mainly up to the management command of *`download_sest`* (see *Usage*).

Warning: Don't make changes to this data as third party apps rely on it to be an exact mirror of the SfH's data. Changing it is up to the *`sestco.pysest.management.commands.download_sest`* management command. If you find some of the data not accurately mirrored, file a bug report for the *`sestco.pysest.management.commands.download_sest`* module instead of manipulating the data by yourself.

Furthermore, modular apps rely on the local database to be up to date and therefor query it instead of the SfH server via the web service. A modular app's documentation should always mention to run the *`download_sest`* management command before other commands involving the database.

The classes have various **dependencies** that must be considered when querying data from the database, visualized by the UML graph.

The **model inheritance** is implicitly shown in the UML graph, but extracted below solitly for an easier recognition:



Example queries

Get all `EinfachstudienangebotsBewerbung` items of a `Bewerber` object (`ForeignKey` relationship), and print them out one by one:

```
bewerber = Bewerber.get(bewerberid='B109064079213')
for bewerbung in bewerber.einfachstudienangebotsbewerbung_set():
    print(bewerbung)
```

The result set is equivalent to:

```
for bewerbung in EinfachstudienangebotsBewerbung.objects.filter(
        bewerber__bewerberid='B109064079213'):
    print(bewerbung)
```

Note that the objects know how to present themselves on print out.

Warning: The primary key of a `Studiengang` is not a compound natural key consisting of the fields "Abschlussschluessel", "Studienfachschluessel", and "Fachkennzeichenschluessel", but an atomic surrogate key `Studiengang.schluessel`. Code that refers to a `Bewerbung` or any other `Studiengang` related object should address only the atomic key for identification. Use helper functions to translate to university specific keys and vice versa, also before fetching Bewerbung items from the Sestco database.

While you might think that fetching directly via the fields of the university's compound primary key is possible, it is always much safer to stick to the actual primary key of the Servicestelle database structure.

Example:

```
def atomic_studiengang_pk(abschluss_schluessel, studienfach_schluessel,
        fachkennzeichen_schluessel=""):
    pk = u"{}-{}".format(abschluss_schluessel, studienfach_schluessel)
    if fachkennzeichen:
    pk += u"-{}".format(fachkennzeichen_schluessel)
    return pk

def compound_studiengang_pk(studiengang_schluessel):
    keys = ('abschluss_schluessel',
       'studienfach_schluessel',
       'fachkennzeichen_schluessel')
    values = studiengang_schluessel.split('-')
    return dict(zip(keys, values))

# Fetch a Bewerbung from the Sestco database:
bewerbung = EinfachstudienangebotsBewerbung.objects.get(
    studiengang__schluessel=atomic_studiengang_pk(
        university_bew.abschl,
        university_bew.fach,
        university_bew.fkz))

# Get university's compound primary key from Servicestelle key:
studiengang_schluessel = compound_studiengang_pk(
    einfachstudienangebotsbewerbung
      .einfachstudienangebot
      .studiengang
      .schluessel)
# Now go fetch data of university's database.
```

Get all `Bewerbung` items, access their child relation and print out its type:

```
for bewerbung in Bewerbung.objects.all():
    if hasattr(bewerbung, 'einfachstudienangebotsbewerbung'):
        bew = bewerbung.einfachstudienangebotsbewerbung
    elif hasattr(bewerbung, 'mehrfachstudienangebotsbewerbung'):
        bew = bewerbung.mehrfachstudienangebotsbewerbung
    else:
        bew = None                  # (It's neither False nor empty,
                                    # it's not known here.)
    if not bew is None:             # (Dealing with None, check for type
                                    # whenenever possible, not for boolean
                                    # expression using ``if not bew``.)
        print("Type is: %s" % type(bew))
    else:
        print("Couldn't detect Bewerbung type.")
```

Check if a Bewerbung is already present at the SfH server, and act according to it:

```
einfachstudienangebot = EinfachStudienangebot.objects.get(
        # (``r`` is a dictionary with Bewerbung data, and
        # The pk format ``xx-yy`` is defined by the university.)
        studiengang__schluessel="{}-{}".format(r['abschl'], r['fach']))
try:
    bewerbung = EinfachstudienangebotsBewerbung.objects.get(
            einfachstudienangebot=einfachstudienangebot,
            bewerber__bewerberid=r['sestBewerberID'])
except EinfachstudienangebotsBewerbung.DoesNotExist:
    # Do stuff for the case of it being not existent at SfH yet.
    pass
else:
    # Do stuff for the case of it being existent at SfH yet.
```

```
    pass
```

As it can be seen in this example, the local database copy is assumed to be up to date and therefor being requested instead of the webservice. This is why a modular app's documentation should always mention to run the `download_sest` management command first.

All classes come with the standard methods for the persistence layer and fetching objects from the database (See *Database storage*). Some have additional attributes.

class sestco.pysest.models.sest.**Bewerbung**(*id*, *created*, *updated*, *bewerber*, *eingangszeitpunkt*, *aktivitaetsstatus*, *bearbeitungsstatus*, *papierantragstatus*, *ort_des_eingangs*, *letzte_aenderung*, *version_sest*, *statusanmerkung*, *sonderantragsentscheidung*, *bewerber_email*)

can_sest_get_in(*overwrites=None*)
Return `True` if the *Bewerbung* is in a Bearbeitungsstatus the Servicestelle can change in to. The schema of status changes can be overwritten with *overwrites* with a dictionary of "bearbeitungsstatus" string (e.g.''in_Vorbereitung'') and tuples, having booleans or numbers evaluated like `(sest_in, sest_out, hs_in, hs_out)`.

can_sest_get_out(*overwrites=None*)
Return `True` if the *Bewerbung* is in a Bearbeitungsstatus the Servicestelle can change out from. The schema of status changes can be overwritten with *overwrites* with a dictionary of "bearbeitungsstatus" string (e.g. `in_Vorbereitung`) and tuples, having booleans or numbers evaluated like `(sest_in, sest_out, hs_in, hs_out)`.

can_uni_get_in(*overwrites=None*)
Return `True` if the *Bewerbung* is in a Bearbeitungsstatus the university can change in to. The schema of status changes can be overwritten with *overwrites* with a dictionary of "bearbeitungsstatus" string (e.g. `in_Vorbereitung`) and tuples, having booleans or numbers evaluated like `(sest_in, sest_out, hs_in, hs_out)`.

can_uni_get_out(*overwrites=None*)
Return `True` if the *Bewerbung* is in a Bearbeitungsstatus the university can change out from. The schema of status changes can be overwritten with *overwrites* with a dictionary of "bearbeitungsstatus" string (e.g. `in_Vorbereitung`) and tuples, having booleans or numbers evaluated like `(sest_in, sest_out, hs_in, hs_out)`.

4.2 The Servicestelle client: `sestclient`

Python wrapper for the "Dialogorientiertes Serviceverfahren für die Hochschulzulassung" API ("Schnittstelle Service–Stelle").

The Servicestelle web API specification can be retrieved from the Stiftung für Hochschulzulassung or T–Systems, but is restricted to internal use only.

The wrapper

- abstracts from the webservice interface to methods that make compulsory and optional arguments more intuitive,
- simplifies nested objects to simple python types, providing helper methods to create complex types for parameters,
- automatically instanciates clients for different services described by seperated WSDL files,
- comes with automated tests of the Servicestelle webservice interface.

Additionally, the wrapper allows to react on API changes easily on one place.

4.2.1 Usage example

Using the *ServiceStelle* class:

```
# Import class and instanciate an object.
from sestco.sestclient import ServiceStelle
sest = ServiceStelle('myusername', 'mypassword')
# Print out available types (and their namespace prefixes) for
# the ``StudiengaengeService`` service.
print(sest.studiengang)
# Request data.
response = sest.benutzer.abrufenStammdatenDurchHS('thebewerberid',
        'thebewerberBAN')
print(response)
```

Using the helper methods to create SOAP objects filled with data, the process to send data is straight forward:

```
from datetime import datetime
from sestco.sestclient import ServiceStelle

sest = ServiceStelle('myusername', 'mypassword')
bewerbungen = []

# One Bewerbung.
fach_englisch = ('025-007', '025', '007', None)
fach_franzoesisch = ('025-059', '025', '059', None)
teilfachbewerbungen = [
        sest.bewerbung.new_bewerbung_teilfach(
            fach_englisch,
            'thebewerberid',
            datetime(2011, 03, 6, 13, 03),
            'eingegangen',
            hauptfach=fach_englisch,
            aktivitaetsstatus='aktiv'),
        self.sest.bewerbung.new_bewerbung_teilfach(
            fach_franzoesisch,
            'thebewerberid',
            datetime(2011, 03, 6, 13, 03),
            'eingegangen',
            hauptfach=fach_franzoesisch,
            aktivitaetsstatus='aktiv'),
        ]
bewerbung = self.sest.bewerbung.new_bewerbung_mehrfach(
        teilfachbewerbungen,
        '025',
        'thebewerberid',
        datetime(2011, 03, 6, 13, 03),
        'eingegangen',
        'HS',
        bewerber_ban=self.bewerber_ban,
        aktivitaetsstatus='aktiv',
        )

# Join to "all" Bewerbungen (which is an empty list at this point).
bewerbungen.append(bewerbung)
# Send the request.
response = sest.bewerbung.uebermittelnNeueBewerbungenAnSeSt(
        '1810', 2012, 'WS', 'Koordinierung', bewerbungen)
print(response)
```

4.2.2 Response values

By using a method for **sending new or updated data**, the werbservice server returns a list of items, each one having an `ergebnisStatus` field and an additional field containing post related data. The response to **requesting data** is a list of items.

The `ergebnisStatus` can have the following values (For the whole list of status messages, see the webservice API description):

`zurueckgewiesen` ("Das angegebene [...] konnte nicht gefunden werden.")

- No item could be found under the given key.
- Not all sub items like `Studiengang` could be found, for example because not all of them have the status `oeffentlich_sichtbar` in the given *serviceverfahren*.

`zurueckgewiesen` ("Die angegebene Version der Bewerbung stimmt nicht mit der im System der Servicestelle gespeicherten Version überein.")

The version at the Servicestelle is higher, which indicates the object was updated in the meanwhile. Solution: Firstly retrieve it, and use this latest one to update its fields and send it back. You may want to resolve the conflict of different data before sending it.

`neu` When an item didn't exist before and now is successfully created.

`aktualisiert` The item has successfully been updated.

Examples

- `StudiengangClient.anlegenAendernStudienangeboteDurchHS()`

  New item:

```
[(StudienangebotErgebnis){
    ergebnisStatus = "neu"
    studienangebotsSchluessel =
        (MehrfachstudienangebotsSchluessel){
            abschlussSchluessel = "25"
        }
}]
```

  Rejected item (`abschlussSchluessel` not found):

```
[(StudienangebotErgebnis){
    ergebnisStatus = "zurueckgewiesen"
    grundZurueckweisung = "Ein Mehrfachstudienangebot muss einen
        Mehrfachabschluss haben. Typ: MehrfachStudienangebot,
        Feld: abschluss | "
    studienangebotsSchluessel = (MehrfachstudienangebotsSchluessel){
        abschlussSchluessel = "26"
    }
}]
```

  Exception (`hochschulnummer` not found):

```
WebFault: Server raised fault: 'Die Hochschule mit der Nummer 4411 wurde
    nicht gefunden. [Sitzungskennung: 110317-135637852_845@51]'
```

- *`BewerbungClient.uebermittelnNeueBewerbungenAnSeSt()`*:

```
[(BewerbungErgebnis){
   ergebnisStatus = "neu"
   bewerbungsSchluessel =
      (MehrfachstudienangebotsbewerbungsSchluessel){
         bewerberId = "B109064079213"
         abschlussSchluessel = "25"
```

```
        hauptfachSchluessel[] =
           (EinfachstudienangebotsSchluessel){
              studiengangSchluessel = "25-084"
              abschlussSchluessel = "25"
              studienfachSchluessel = "084"
           },
           (EinfachstudienangebotsSchluessel){
              studiengangSchluessel = "25-059"
              abschlussSchluessel = "25"
              studienfachSchluessel = "059"
           },
     }
}]
```

- *BenutzerClient.abrufenStammdatenDurchHS()*:

```
[(Bewerber){
   status = "aktiv"
   anrede = "Frau"
   vorname = "Johanna"
   weitereVornamen = None
   ...
   postanschrift =
      (Postanschrift){
         ...
         land =
            (Land){
               schluessel = "000"
               nameDe = "Deutschland"
            }
         region = None
      }
}]
```

4.2.3 Special methods to synchronize data changed by others

There are methods to get data and to post data via the client. However, when data is changed at the *Servicestelle* by another university that the own university is using as well, the *Servicestelle* would have to synchronize it, and to post the data to webservices of all universities. The implemented solution uses a different way to synchronize this, it can be triggered by the university clients and avoids the need of a webservice on the university side: A method can be used to regularly retrieve changed or new items. For performance issues, this is provided in two steps: The first request fetches temporary references of recently changed data sets, and a second method fetches ranges or full data of those sets. The full list of methods is:

- *BenutzerClient.anfragenStammdatenaenderungenDurchHS()*
- *BenutzerClient.uebermittelnStammdatenaenderungAnHS()*
- *BewerbungClient.anfragenNeueGeaenderteBewerbungenDurchHS()*
- *BewerbungClient.uebermittelnNeueGeaenderteBewerbungenAnHS()*
- *BewerberauswahlClient.anfragenGeaenderteRanglistenstatusDurchHS()*
- *BewerberauswahlClient.uebermittelnGeaenderteRanglistenstatusAnHS()*

4.2.4 The “Servicestelle” class

class sestco.sestclient.**ServiceStelle**(*username*, *password*, *location*, *proxy=None*)

Main class for accessing the webservices of the *Servicestelle für Hochschulzulassung*.

The client objects for each service can be accessed via the following attributes:

studiengang *StudiengangClient* object

benutzer *BenutzerClient* object

bewerbung *BewerbungClient* object

bewerberauswahl *BewerberauswahlClient* object

unterstuetzung *UnterstuetzungClient* object

See those classes for available methods.

Example:

```
from datetime import datetime
from sestco import sestclient

sest = sestclient.ServiceStelle('myusername', 'mypassword_+#9/')
response = sest.benutzer.anfragenStammdatenaenderungenDurchHS(
        '1810', '2012', 'SS', 'Koordinierung',
        datetime(2010, 12, 12, 12, 00))
print(response)
```

4.2.5 The client classes

class sestco.sestclient.**StudiengangClient**(*username*, *password*, *wsdl_file*, *service_path*, *servicestelle_version*, *location*, *proxy=None*)

Class to provide access to StudiengaengeService methods.

The class provides client methods to access the webservice server, and helper methods to construct SOAP objects easily that serve as a parameter. An instance of this class can be accessed via the attributes of *ServiceStelle* and in most cases may not be instantiated manually.

abrufenStudienangeboteDurchHS(*hochschulnummer*, *jahr*, *semester*, *vermittlungsprozess_typ*, *abschluss_schluessel=None*)

Return a list of Studienangebot objects.

class sestco.sestclient.**BenutzerClient**(*username*, *password*, *wsdl_file*, *service_path*, *servicestelle_version*, *location*, *proxy=None*)

Class to provide access to BenutzerService methods.

The class provides client methods to access the webservice server, and helper methods to construct SOAP objects easily that serve as a parameter. An instance of this class can be accessed via the attributes of *ServiceStelle* and in most cases may not be instantiated manually.

abrufenStammdatenDurchHS(*bewerber_id*, *bewerber_ban*)

Return applicant data for the user that authorized the university by his *bewerber_id* and *bewerber_ban*.

anfragenStammdatenaenderungenDurchHS(*hochschulnummer*, *jahr*, *semester*, *vermittlungsprozess_typ*, *start_date*)

Return a list of reference numbers for accounts whiches data has been changed since *start_date*. To retrieve full data sets, the *uebermittelnStammdatenaenderungAnHS()* method can be applied.

uebermittelnStammdatenaenderungAnHS(*hochschulnummer*, *jahr*, *semester*, *vermittlungsprozess_typ*, *referenzen*)

Return the full data sets for users referenced by a number in *referenz*, a list of references retrieved by *anfragenStammdatenaenderungenDurchHS()*.

abrufenStammdatenDurchHSohneAutorisierung(*hochschulnummer*, *jahr*, *semester*, *vermittlungsprozess_typ*, *bewerber_id=None*, *abschluss_schluessel=None*, *studienfach_schluessel=None*, *fachkennzeichen_schluessel=None*)

Return the applicant's data that match the given criteria of *bewerber_id*, *abschluss_schluessel*, *studienfach_schluessel*, or *fachkennzeichen_schluessel*. This can be done without its authorization when the user has submitted a complete application using his BAN to university *hochschulnummer*.

At least one of the following parameters must be given: `bewerber_id`, `abschluss_schluessel`, `studienfach_schluessel`, `fachkennzeichen_schluessel`.

class sestco.sestclient.**BewerbungClient**(*username*, *password*, *wsdl_file*, *service_path*, *servicestelle_version*, *location*, *proxy=None*)

Class to provide access to `BewerbungenService` methods.

The class provides client methods to access the webservice server, and helper methods to construct SOAP objects easily that serve as a parameter. An instance of this class can be accessed via the attributes of *ServiceStelle* and in most cases may not be instantiated manually.

uebermittelnNeueBewerbungenAnSeSt(*hochschulnummer*, *jahr*, *semester*, *vermittlungsprozess_typ*, *bewerbungen*)

Create the applications in *bewerbungen*, a list of `Bewerbung` objects as instantiated by *new_bewerbung_einfach()* or *new_bewerbung_mehrfach()*, or using the raw way with `factory.create('ns2:Mehrfachstudienangebotsbewerbung')`. Making use of helper functions as mentioned first is much more convenient and recommended as it is much less complex.

Note: Troubleshooting

- Never set the `fachkennzeichen_schluessel` for the `StudienangebotsSchluessel` to `''` as this turns into `None` in the XML file which will result in not finding the `Studienangebot`. For a not set field, use `None` in Python code instead or don't set the field at all, as both will not declare the field and won't brake the search.
- For the "Studienangebot" to be found, all the attributes of the keys must be set right, particularly the `fachkennzeichen_schluessel`, e.g. if it must be set at all for the particular `Studienangebot`.

`Bewerbung` must be a complex type that can either be created manually using the `factory.create` method or the more convenient helper methods that start with `new_`.

uebermittelnGeaenderteBewerbungenAnSeSt(*hochschulnummer*, *jahr*, *semester*, *vermittlungsprozess_typ*, *bewerbungen*)

Update the applications in *bewerbungen*, a list of `Bewerbung` items. See *uebermittelnNeueBewerbungenAnSeSt()* for more informations.

`Bewerbung` must be a complex type that can either be created manually using the `factory.create` method or the more convenient helper methods that start with `new_`.

anfragenNeueGeaenderteBewerbungenDurchHS(*hochschulnummer*, *jahr*, *semester*, *vermittlungsprozess_typ*, *start_date*)

Request a list of references for applications that have been changed since *start_date*. This method is used to get the list of references for *uebermittelnNeueGeaenderteBewerbungenAnHS()*.

uebermittelnNeueGeaenderteBewerbungenAnHS(*hochschulnummer*, *jahr*, *semester*, *vermittlungsprozess_typ*, *referenzen*)

Return a list of items for the applications referenced by *referenzen*, which are retrieved by *anfragenNeueGeaenderteBewerbungenDurchHS()*.

abrufenBewerbungenDurchHS(*hochschulnummer*, *jahr*, *semester*, *vermittlungsprozess_typ*, *bewerber_id=None*, *abschluss_schluessel=None*, *studienfach_schluessel=None*, *fachkennzeichen_schluessel=None*)

Return a list of applications that match the given criteria of *bewerber_id*, *abschluss_schluessel*, *studienfach_schluessel* and *fachkennzeichen_schluessel*.

At least one of the following parameters must be given: `bewerber_id`, `abschluss_schluessel`, `studienfach_schluessel`,

fachkennzeichen_schluessel.

new_bewerbung_einfach(*studiengang*, *bewerber_id*, *eingangszeitpunkt*, *bearbeitungsstatus*, *version_sest=None*, *bewerber_ban=None*, *aktivitaetsstatus=None*, *papierantragstatus=None*, *statusanmerkung=u"*, *sonderantragsentscheidung=u"*)

Return a Einfachstudienangebotsbewerbung, constructed from *studiengang*, which is a tuple like so:

```
(
    studiengang_schluessel,
    abschluss_schluessel,
    studienfach_schluessel,
    # optional:
    fachkennzeichen_schluessel
)
```

version_sest must be set for updating applications (e.g. when using *uebermittelnGeaenderteBewerbungenAnSeSt()*).

Helper method to avoid using the SOAP client's factory directly. Its return value is an argument for methods like *uebermittelnNeueBewerbungenAnSeSt()*, *uebermittelnGeaenderteBewerbungenAnSeSt()*.

new_bewerbung_mehrfach(*teilfachbewerbungen*, *abschluss_schluessel*, *bewerber_id*, *eingangszeitpunkt*, *bearbeitungsstatus*, *version_sest=None*, *bewerber_ban=None*, *aktivitaetsstatus=None*, *papierantragstatus=None*, *statusanmerkung=u"*, *sonderantragsentscheidung=u"*)

Return a Mehrfachstudienangebotsbewerbung, out of multiple *teilfachbewerbungen*, which is a list of at least two items returned by *new_bewerbung_teilfach()*.

Helper method to avoid using the SOAP client's factory directly. Its return value is an argument for methods like *uebermittelnNeueBewerbungenAnSeSt()*, *uebermittelnGeaenderteBewerbungenAnSeSt()*.

new_bewerbung_teilfach(*studiengang*, *bewerber_id*, *eingangszeitpunkt*, *bearbeitungsstatus*, *version_sest=None*, *hauptfach=None*, *ersatzfach=None*, *ist_ersatzfach_akzeptabel=False*, *ist_fachwechsel=False*, *aktivitaetsstatus=None*, *statusanmerkung=u"*, *sonderantragsentscheidung=u"*)

Return a Teilfachbewerbung, constructed from *studiengang*, which is a tuple like so:

```
(
    studiengang_schluessel,
    abschluss_schluessel,
    studienfach_schluessel,
    # optional:
    fachkennzeichen_schluessel
)
```

letzte_aenderung and *version_sest* must be set for updating applications (e.g. when using *uebermittelnGeaenderteBewerbungenAnSeSt()*). *hauptfach* and *ersatzfach* are tuples of the same structure as *studiengang*.

Helper method to avoid using the SOAP client's factory directly. Its return value is an argument for methods like *new_bewerbung_mehrfach()*.

class sestco.sestclient.**BewerberauswahlClient**(*username*, *password*, *wsdl_file*, *service_path*, *servicestelle_version*, *location*, *proxy=None*)

Class to provide access to BewerberauswahlService methods.

The class provides client methods to access the webservice server, and helper methods to construct SOAP objects easily that serve as a parameter. An instance of this class can be accessed via the attributes of

ServiceStelle and in most cases may not be instantiated manually.

anlegenAendernStudienpaketeDurchHS()
Not implemented yet.

uebermittelnRanglistenAnSeSt(*hochschulnummer*, *jahr*, *semester*, *vermittlungsprozess_typ*, *rangliste_set*)
Send `Rangliste` items of *rangliste_set* and their entries to Servicestelle.

`Rangliste` must be a complex type that can either be created manually using the `factory.create` method or the more convenient helper methods that start with `new_`.

anfragenGeaenderteRanglistenstatusDurchHS(*hochschulnummer*, *jahr*, *semester*, *vermittlungsprozess_typ*, *start_date*)
Return a list of reference numbers for Rangliste items whiches data has been changed since *start_date*. To retrieve full data sets, the *uebermittelnGeaenderteRanglistenstatusAnHS()* method can be applied.

uebermittelnGeaenderteRanglistenstatusAnHS(*hochschulnummer*, *jahr*, *semester*, *vermittlungsprozess_typ*, *referenzen*)
Return the full data sets for Rangliste items referenced by a number in *referenz*, a list of references retrieved by *anfragenGeaenderteRanglistenstatusDurchHS()*.

abrufenRanglistenstatusDurchHS()
Not implemented yet.

ausloesenFruehzeitigerZulassungsangeboteDurchHS()
Not implemented yet.

abrufenStudienpaketeDurchHS(*hochschulnummer*, *jahr*, *semester*, *vermittlungsprozess_typ*)
Return all `Studienpaket` items of the current process.

new_rangliste_with_kopfdaten(*schluessel*, *studienpaket_schluessel*, *ebene*, *status*, *abarbeitungsposition*, *plaetze_type*, *plaetze*, *ist_vorwegzulasserrangliste*, *ist_chancenrangliste*, *name_de=None*, *name_en=None*, *klauseln=None*, *eintrag_set=None*, *statusanmerkung=None*)
Return a `Rangliste` item, containing *eintrag_set*, a list of `(bewerber_id, rang)` tuples. *status* is one of `'eingerichtet'`, `'befuellt'`, `'freigegegben'`, and *ebene* is one of `'Vorabquoten'`, `'geminderte_Vorabquoten'`, `'Hauptquoten'`. *plaetze_type* is one of `'prozentual'` or `'absolut'`. See also: *new_rangliste_without_kopfdaten()*. *klauseln* is a dictionary containing the Klausel name as the key and Klausel specific configuration values as a subdictionary, or only `True` if no configuration values are set. Example:

```
{
    # The entrie's value of a *Klausel* is ``True`` or a
    # dictionary of subfields.
    'Binnenquotenklausel': {
        'binnenquotenkennung': u'my_kennung',
        'als_binnenquotendachrangliste': True,
        },
    'Mindestplatzklausel': {
        'grenzkapazitaet': 20,
        },
    # If not set, the value defaults to ``False``.
    #'Limitierungsklausel': True,
    'Unterbelegungsklausel': {
        # ID name of *Ebene* or ID number of *Rangliste* type,
        # e.g. 'Hauptquoten' or 66.
        'platzuebergabe': 'Hauptquoten',
        },
    'Nichtauffuellklausel': True,
    },
```

At least one of the following parameters must be given: `name_de`, `name_en`.

Helper method to avoid using the SOAP client's factory directly. Its return value is an argument for methods like *uebermittelnRanglistenAnSeSt()*.

new_rangliste_without_kopfdaten(*schluessel*, *studienpaket_schluessel*, *ebene*, *status*, *eintrag_set*, *statusanmerkung=None*)

Return a `Rangliste` item, without the need to give the configuration parameters (*kopfdaten-Beibehalten*). See *new_rangliste_with_kopfdaten()* for parameter explanation. See also: *new_rangliste_with_kopfdaten()*.

Helper method to avoid using the SOAP client's factory directly. Its return value is an argument for methods like *uebermittelnRanglistenAnSeSt()*.

class sestco.sestclient.**UnterstuetzungClient**(*username*, *password*, *wsdl_file*, *service_path*, *servicestelle_version*, *location*, *proxy=None*)

Class to provide access to `UnterstuetzungsService` methods.

The class provides client methods to access the webservice server, and helper methods to construct SOAP objects easily that serve as a parameter. An instance of this class can be accessed via the attributes of *ServiceStelle* and in most cases may not be instantiated manually.

abrufenDokumenteDurchHS(*hochschulnummer*, *dokumentids*)

Retrieve and return documents of the corresponding *dokumentids* from the Servicestelle.

dokumentids is an iterable of ID strings.

4.3 Utils

class sestco.pysest.utils.**LeftTrack**(*item_amount*)

Track of time left to reach an amount of items *item_amount* to zero.

Usage example:

```
items = (1, 2, 3, 4, 5, 6, 7, 8, 9)
lt = LeftTrack(len(items))
for i in items:
    lt.track_item()
    print(lt)
    time.sleep(1)
```

The counter is not accurate yet, but does the job of an estimation.

track_item()

Set a track point. Use this within a loop.

class sestco.pysest.utils.**Timestamp**(*app*, *name*, *default_get=None*)

Handle saving `Variable` objects specially for saving timestamps. If *default_get* is `None`, the start datetime of the current vermittlungsprozess is set as default value for *get()*.

When saving, the value will be cached, as well as when loading, so no database access is done afterwards for *load()*.

Use >= for refering to timestamp, not >.

Note: >= is safier not to loose data items in case of the process being interrupted with items having the same timestamp. To avoid *always* having the last updated one to be updated again, the timestamp is raised by one second before saving in case of success flag is not set to `False`.

---

set(*timestamp=None*)

Set the object attribute to datetime *timestamp*, without storing it to the database. Use the *save()* method for persisting it. If timestamp is `None`, the current time is taken.

get()
Return the last by *set()* set timestamp. This may differ from the one returned by *load()*.

save(*successful=True*)
Save timestamp set by *set()* timestamp.

load()
Retrieve latest saved timestamp from database, or if not in database, return the default value, and if not set, return the start datetime of the current vermittlungsprozess.

load_str()
Same as *load()*, but returns the timestamp as a formatted string.

delete()
Delete timestamp object from database.

sestco.pysest.utils.**gives_permission**(*message*, *default='yes'*)
Prompt for permission on command line, asking *message*, and to continue.

sestco.pysest.utils.**str2bool**(*value*)
Convert *value* (which can be "yes", "no" and others, see source code) to a boolean.

class sestco.pysest.utils.**LeftTrack**(*item_amount*)
Track of time left to reach an amount of items *item_amount* to zero.

Usage example:

```
items = (1, 2, 3, 4, 5, 6, 7, 8, 9)
lt = LeftTrack(len(items))
for i in items:
    lt.track_item()
    print(lt)
    time.sleep(1)
```

The counter is not accurate yet, but does the job of an estimation.

track_item()
Set a track point. Use this within a loop.

class sestco.pysest.utils.**Timestamp**(*app*, *name*, *default_get=None*)
Handle saving `Variable` objects specially for saving timestamps. If *default_get* is `None`, the start datetime of the current vermittlungsprozess is set as default value for *get()*.

When saving, the value will be cached, as well as when loading, so no database access is done afterwards for *load()*.

Use >= for refering to timestamp, not >.

Note: >= is safier not to loose data items in case of the process being interrupted with items having the same timestamp. To avoid *always* having the last updated one to be updated again, the timestamp is raised by one second before saving in case of success flag is not set to `False`.

---

set(*timestamp=None*)
Set the object attribute to datetime *timestamp*, without storing it to the database. Use the *save()* method for persisting it. If timestamp is `None`, the current time is taken.

get()
Return the last by *set()* set timestamp. This may differ from the one returned by *load()*.

save(*successful=True*)
Save timestamp set by *set()* timestamp.

load()
Retrieve latest saved timestamp from database, or if not in database, return the default value, and if not set, return the start datetime of the current vermittlungsprozess.

load_str()
Same as *load()*, but returns the timestamp as a formatted string.

delete()
Delete timestamp object from database.

class sestco.pysest.models.utils.**Variable**(**args*, ***kwargs*)
Class for storing persistent values.

classmethod set(*app*, *name*, *value*)
Store a *value* for variable *name* of *app* to database.

classmethod get(*app*, *name*, *default=None*)
Get a value for variable *name* of *app*. If a variable of this name can't be found, *default* is returned.

classmethod remove(*app*, *name*)
Delete the variable from database.

get_value()
Return the stored value. This is used in the admin interface, for example.

class sestco.pysest.models.utils.**UploadLog**(**args*, ***kwargs*)
A log entry about an attemp of sending an item to the Servicestelle, saving fields like the used request method, request value, response value and more. The entries are linked to content type and therefor can be listed together across models or by model: It makes use of a generic relation of the Django content type API.

Example:

```
my_content_type_object = MyObject.objects.get(
        # ...
        )
logentry = UploadLog(
        sent=my_send_date,
        status=my_item_webservice.ergebnisStatus,
        request_method=my_content_type_object.sest_request_method,
        request=my_content_type_object.sest_request,
        response=my_item_webservice,
        content_object=my_content_type_object)
# (Set None to overwrite an obsolete one.)
logentry.reason = None
if hasattr(my_item_webservice, 'grundZurueckweisung'):
    logentry.reason = unicode(my_item_webservice.grundZurueckweisung)
    my_content_type_object.sest_reason = unicode(
            my_item_webservice.grundZurueckweisung)
logentry.full_clean()
logentry.save()
```

4.4 Django Framework API

Regarding the *Django* part, knowing only the database API is sufficient. Depending on what functionalities the custom module implements, further fields have to be considered. The *Django* framework is powerful but simple. The following is just a short overview though for starting up easier, referencing to the more detailled resources. This section focuses on the functionalities mostly needed when writing a modular app for the purpose given here, so some are not introduced here, like the signal system, support for internalization, caching, email and more. For a full overview of the Django's API see the official documentation.

4.4.1 Database storage

Loading an object is either done by `get()`, `all()`, or `filter()` and `exclude()`. The `get()` and `all()` methods differ from how many objects are being fetched and exceptions being raised (e.g., `get()` raises an

exception if there is not excactly one object found). Slicing a result set is possible, and `get()` can be invoked after `filter()`/`exclude()` on a query object. Knowing the model dependencies, the *Django* database API allows attribute like queries:

```
strasse = bewerbung.bewerber.postanschrift.strasse

# Or for more complex spanning relationships queries:
bewerber = Bewerber.objects.filter(
        postanschrift__strasse__name=u"Hafenstraße",
        familienname=u"Weber",
        bewerbung_set__bearbeitunsstatus=u'eingegangen')
```

Note: The following examples run the `full_clean` method that validates the model before saving, as *Django* doesn't run all validations on saving by its own. To make it more save, the models of the Servicestelle invoke this method automatically on saving and therefore it is actually not necessary to run it explicitly for them (but it is necessary for any other ones like an app's custom ones that don't implement the automation). It may still be done though for the Servicestelle ones as well, e.g. for catching the exception (which is rarely necessary).

To **update** its attributes, assign new values and save it. Note that the attributes follow the python coding guidelines, having underscores instead of capital letters to seperate words:

```
bewerber = Bewerber.objects.get(bewerber_ban='B1234567')
bewerber.familienname = u"Weber"
# A try--catch clause can be used to catch a ValidationError
# exception. In most cases here, it is for no use, though.
try:
    bewerber.full_clean()
except ValidationError:
    # Do some stuff ...
    print("Validation error")
    raise
bewerber.save()

# It is also possible to supply an object as parameter:
anschrift = Postanschrift.objects.get(bewerber=bewerber)
```

Creating an object is straight forward:

```
bewerber = Bewerber(
    bewerber_ban='B1234567',
    familienname=u"Müller",
    # ...
    )
# Validate and save to database.
bewerber.full_clean()
bewerber.save()
```

To **delete** an object, call its `delete()` method:

```
bewerber.delete()
```

For more detailed informations, check the Django QuerySet API reference.

4.4.2 Transactions

In some rare cases it is necessary to automatically or manually rollback database transactions, e.g. if one object should only be stored in case of saving another object is successful. The *Django* database layer provides the possibility to set default transaction or to change the behaviour on a per function or per code block level. See the according manual about managing database transactions.

4.4.3 Management commands

The management command API can be used to create console commands that have access to the database backend and the whole framework API. Here is a fictive example:

```
from django.core.management.base import BaseCommand, CommandError
from django.conf import settings

from sestco.sestclient import ServiceStelle
from sestco.pysest.models import MyModel, AnotherModel
from sestco.pysest import get_vermittlungsprozess


class Command(BaseCommand):
    help = "Retrieve data from the SfH and write it to local database"

    def handle(self, *args, **options):
        # Do some stuff here ...
        pass
```

This command can be used from the command line like this:

```
python manage.py download_sest
```

The API has support for parameters and complex option sets. For more detailed informations, check the Django Management Command API reference.

For this interface, commands *must never rely on states of the current database only*. However it is possible for efficiency purposes, but in this case they need to be able to restore the current database state relevant for synchronization even on an empty database. They may loose log data, but should never be unable to continue synchronization smoothly after a full database loss. It must be able to collect all the missing data relevant for the synchronization process again. This is a design decision to ease administration as much as possible when having to get back from an unconsistent state to a consistent state.

> **Warning:** If the rule of state independence is not accomplished in a modular app, it can result in data loss, as the user will expect it to be state independent.

The management should also be implemented as an action controller and view, firing methods and handling user input, while the operations on data is outsourced to the model classes, handler classes or others that are the right choice to operate on data. This seperation of data operations makes them accessible from other workflows like a view preparing and handling data for a server side HTML or RESTful interface serving XML or JSON formatted data.

4.4.4 Logging

Django provides a wrapper for the Python standard logging API, which is mature and allows fine graned logging messages and configuration. Any modular app should make use of it to provide debug, general and error informations and warnings. There is a definition for the severity levels defined, which is more specific than the *Django* one, and is explained in the *Sestco Settings* section.

Implementing logging in a modular app is fairly simple:

```
from logging import getLogger
# Set module name here or, for implementing a non app specific logger,
# the name of the logger, e.g. `'django.request'`.
logger = getLogger(__name__)


logger.debug("This is a debug data text.")
```

```
logger.info("This is a general information text.")
logger.warning("This is a warning description.")
logger.error("This is an error description.")
```

It is recommended to use this system from the beginning instead of standard prints to the console. A logger can also be configured in a start up script, either by the Python logging `DictConf` format or (not recommended) by methods of logger objects.

For detailed informations about logging see the Django documentation about logging.

4.4.5 Admin panel

While the user frontend is supposed to be used to manage the application from an end user's perspective, *Django* allows to configure an admin panel for managing the project on a meta level, e.g. unpublishing unappropriate comments. The admin panel is meant for a *trusted user circle*, e.g. website administrators. It makes heavy use of automatically generated forms, considering the dependencies between models. The customization of this backend is somehow limited due to it's automatic generation, but is still highly customably in sort of

- fields to be shown
- filters that can be applied simultanously
- search capability
- showing and editing inline child objects
- defining lists

and more. It is *not meant* to be used by users of the application, as it is not designed for non trustful users and doesn't contain any workflow that reduces and sorts the available steps to a workflow.

See the admin panel documentation for detailed informations.

4.4.6 Web–GUI

The Web–GUI API consists of a URL, a method and a template system. It may access a webservice, internal methods basically used for web service requests, or simple so called view methods that process data and supply the context variables for the templates. See the Django documentation for how to use it.

4.4.7 Authentication

Django comes with a user authentication system. It handles user accounts, groups, permissions and cookie-based user sessions, if needed. The official documentation can be found at https://docs.djangoproject.com/en/dev/topics/auth/.

4.4.8 Webservice API

There is the possibility to give RESTful webservice capability to the framework. This can be useful if the amount of external applications is high or if the data should not be sent and retrieved actively by the interface, but by the external university applications itselfs, or for providing JSON formatted data for an AJAX client based GUI.

It is never a good idea to map all RESTful resources strictly to database models, though in many cases it is the right way. This is why a for the RESTful implementation a framework is recommended and being used for the core applications that its main purpose is to provide resources, which can retrieve data from any source like file system, but has full support for *Django* models and optionally direct mapping as well. This results in a high level of flexibility and modeling resources as they are actually needed for a specific use case, e.g. when a resource handles several database classes due to dependencies between them, or when an artificial resource must be designed to handle the limited verbs of a RESTful implementation.

4.5 Sestco Deprecation Timeline

This document outlines when various pieces of Sestco will be removed or altered in a backward incompatible way, following their deprecation, as per the Django deprecation policy. More details about each item can often be found in the release notes of two versions prior.

4.5.1 0.3

See the *Sestco 0.1 release notes* for more details on these changes.

4.5.2 0.4

See the *Sestco 0.2 release notes* for more details on these changes.

CHAPTER 5

Contributing to Core

5.1 Working with Mercurial and the upstream repository

5.1.1 Installing Mercurial

Install Mercurial with your operating system's package manager. On Windows, download mercurial from Mercurial's downloads page, and in the install wizard remove the translations from installation to save disk space.

After installing Mercurial the first thing you should do is setup your name, email, and extensions in `~/.hgrc`:

```
[ui]
username = John Doe <john@example.com>

[extensions]
rebase =
color =
```

Note that `username` should be your real name, not a nick.

5.1.2 Setting up local repository

- Create your Bitbucket account, e.g. with the nick "jdoe".
- Fork Sestco's repository to your own account.
- Create a local copy of your fork:

  ```
  hg clone https://jdoe@bitbucket.org/jdoe/sestco
  ```

This will create a new directory "sestco", containing a clone of your Bitbucket repository.

5.1.3 Working on a ticket

You might first of all want to set the ticket status to "accepted" in the online issue tracker, to show others that somebody is working on that ticket, if it is not "accepted" by somebody else yet.

When working on a ticket base that work on master, and create a new branch for the work:

```
hg update -C master
hg bookmark ticket_xxxxx
```

The bookmark command creates a new bookmark that you can use to address later. Don't hesitate to create new branches even for the smallest things - that's what they are there for.

If instead you were working for a fix on the 1.4 branch, you would do:

```
hg update 1_4
hg bookmark ticket_xxxxx_1_4  # If you receive the message that the bookmark
                              # already exists, run ``hg update ticket_xxxxx_1_4``.
```

Assume the work is carried on `ticket_xxxxx` branch. Make some changes and commit them:

```
hg status   # Optional, but recommended: Check file status.
hg diff     # Optional, but recommended: Check file changes.
hg commit
```

If you need to do additional work on your branch, commit as often as necessary:

```
hg commit -m "Added two more tests for edge cases"
```

If you want a view on the history graph, use:

```
hg log -G
```

After upstream has changed

When upstream has changed, you can just pull.[2] Note the different user names in the URL: the first one is yours, the second of the repo owner:

```
# Pull in the latest master from the main repository (not your fork).
hg pull -B master https://jdoe@bitbucket.org/cschur/sestco
hg rebase -d master   # Rebase active branch on master.
```

The work is automatically rebased.[3] The rebase command removes all your local commits temporarily, applies the upstream commits, and then applies your local commits again on the work. If there are merge conflicts you will need to resolve them.

Publishing work at your upstream repository

You can publish your work on Bitbucket just by doing:

[2] If you manage your own, seperate repository to publish your current state in case you don't have write access to the main one, telling Mercurial the main repository is necessary. Your repo is connected to your own (fork) repository, thus you need to call other repositories explicitly to interact with.

[3] It is preferable to *rebase* on upstream, not *merge* the upstream.

The reason for this is that by rebasing, your commits will always be *on top of* the upstream's work, not *mixed in with* the changes in the upstream. This way your branch will contain only commits related to its topic, which makes squashing easier. Never do the following as long as a difficult history doesn't make it unevitable:

```
hg pull -B master
# Don't do this:
merge master
hg commit
```

Merging is good when managing branches across features, e.g. for getting bug fixes from a freeze branch into the default (master, development) branch.

> **Warning:** **Never use merging during feature development**. Use rebasing instead.

```
hg push -B ticket_xxxxx          # Push changed ticket_xxxxx upstream.
```

When you go to *your* Bitbucket page you will notice a new branch has been created.

If you are working on a Trac ticket, you should mention in the ticket that your work is available from branch ticket_xxxxx of your Bitbucket repo. Include a link to your branch.

Note that the above branch is called a "topic branch". You are free to rewrite the history of this branch, by using `hg rebase` for example. Other people shouldn't base their work on such a branch, because their clone would become corrupt when you edit commits.

There are also "public branches". These are branches other people are supposed to fork, so the history of these branches should never change. Good examples of public branches are the master and stable/A.B.x branches in the main repository.

Creating feature or bugfix changesets

When you think your work is ready to be pulled into Sestco, you should create a changeset that contains all the minor commits of this feature or bug fix.

A good changeset means:

1. The work includes documentation and tests, if needed – actually tests are always needed, except for documentation changes.
2. Following the *coding style*[1].
3. `flake8` must pass.
4. The documentation must build without warnings.
5. The test suite must pass.
6. A commit with one logical change.

   In the example above you created two commits working on one feature, the "Fixed ticket_xxxxx" commit and "Added two more tests" commit.

   We do not want to have the entire history of your working process in your repository. Your commit "Added two more tests" would be unhelpful noise. Instead, we would rather only have one commit containing all your work.

   (a) To rework the history of your feature or bugfix branch you can squash the commits into one for a feature or bugfix by using rebase:

   ```
   # Pull in the latest master. Be aware of the different user names in
   # the URL, and replace ``jdoe`` by your one.
   hg pull -B master https://jdoe@bitbucket.org/cschur/sestco
   hg update ticket_xxxxx          # Switch to ticket branch.
   hg rebase -d master --collapse  # Rebase it on master, and
                                 # collapse its descendants.
   hg bookmark master              # Move master bookmark to top changeset.
   hg bookmark -d ticket_xxxxx     # Delete now obsolete bookmark.
   ```

   This will keep the first commit of the `ticket_xxxxx` branch, squash the afterwards commits into the first one, and rebase it on `master`. An editor window should open, so you can reword the commit message for the commit now that it includes several steps.[8]

[1] https://docs.djangoproject.com/en/dev/internals/contr ibuting/writing-code/coding-style/

[8] There is also a more explicit way of doing this, which allows to rebase on the current destination, so actually a collapse without rebasing:

```
hg rebase -s START_REF -d BEFORE_START_REF --collapse # Collapse descendants
                                                      # into START_REF.
```

`START_REF` is the root revision of the issue branch. This will keep the `START_REF` commit, and squash the afterwards commits into the first one. An editor window should open, so you can reword the commit message for the commit now that it includes several steps.

To *remove* experimental lightweight branches, use `hg strip`.

Don't use the `hg histedit` or `hg collapse` command, as they don't preserve file renames.

(b) Well-formed messages for each commit: a summary line of maximum 50 characters and then paragraphs wrapped at 72 characters thereafter[4,7]. Start with these past term verbs: "**Added**" (new feature), "**Fixed**" (bug fix), "**Changed**" (task), "**Updated**" (task, due to changes in third-party code). Note that the summary line is *less* precise than the ticket title in the issue tracker.

Examples:

```
#1234 -- Added diff merging for user data.
#5678 -- Fixed diff merging on special characters.
#9012 -- Changed default language from german to english.
#3456 -- Updated ez_setup.py from 0.6a7 to 0.6a9.
```

General format:

```
#ISSUEID -- VERB WHAT.
```

Committing to the main repository

When you created a feature or bugfix changeset that you think is ready to be pulled into Sestco, you should create a pull request at Bitbucket.

Once you have created your pull request, you should add a comment in the related Trac ticket explaining what you've done. In particular you should note the environment in which you ran the tests, for instance: "all tests pass under SQLite and MySQL".

After the changeset has been published, go to the online issue tracker and close the ticket with a message that links to the Mercurial revision (assume `2a1c6bd2bb37` is the Mercurial revision id):

```
In [2a1c6bd2bb37]
```

[4] https://docs.djangoproject.com/en/dev/intern als/contributing/committing-code/#committing-guidlines
[7] http://stopwritingramblingcommitmessages. com/, http://stackoverflow.com/q/43598

CHAPTER 6

Release notes

Release notes for the official Sestco releases. Each release note will tell you what's new in each version, and will also describe any backwards-incompatible changes made in that version.

For those upgrading to a new version of Sestco, you will need to check all the backwards-incompatible changes and *deprecated features* for each 'final' release from the one after your current Sestco version, up to and including the new version. If a release includes an upgrade to a new Django version, check the Django release notes for its backwards-incompatible changes as well.

6.1 Final releases

6.1.1 Pre-1.0 releases

In contrast to Post-1.0 releases, Pre-1.0 releases don't come with a **deprecation path**, because they are meant for faster development in the growth phase of a project. This means backwards-incompatible changes might be introduced from one version to another, without deprecating them two versions in advance. They are still announced, and there's just less time without overlapping of the old and the new implementation way.

A (database) **migration path** is provided though for those releases as well as for Post-1.0-releases.

Sestco 0.3 release notes

Welcome to Sestco 0.3!

These release notes cover the *new features*, as well as some *backwards incompatible changes* you'll want to be aware of when upgrading from Sestco 0.2 or older versions. We've also dropped some features, which are detailed in *our deprecation plan*, and we've *begun the deprecation process for some features*.

What's new in Sestco 0.3

For end users

Introduced setting to reduce downloaded data

The *SECO_STUDIENANGEBOT_DOWNLOAD* setting was introduced to be able to reduce the amount of downloaded data on each download cycle.

Added download of `FreieFachbezogeneStudiengangskombination`

The download of `Studienangebot` items now supports the `FreieFachbezogeneStudiengangskombination` type.

Added support for Servicestelle version 3

Support for Servicestelle version 3.x has been added. The default webservice endpoint is now:

```
https://dosv.hochschulstart.de/hochschule/webservice/3/
```

In case the endpoint is defined explicitly in the settings, it must be updated to the new endpoint of Servicestelle version 3.

Field `location` of setting `SECO_WEBSERVICE` has become mandatory

The item `location` of setting *SECO_WEBSERVICE* has become mandatory. Relying on a default value here has proven to be prone to errors in shifts from productive setups to testing environments.

Replaced `sest-admin` and built-in project scaffolding by external templates

External Secsto project templates are used instead of a built-in solution. Setting up a project is explained in the installation instructions.

For programers

`settings.py`

- BASE_DIR should be better defined as:

```
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
```

- Build paths inside the project like this: `os.path.join(BASE_DIR, ...)`.
- Remove `TEMPLATE_DEBUG` setting.

Backwards incompatible changes in 0.3

Changed primary key of `Studiengang`

The primary key of a `Studiengang` has changed from a compound natural key consisting of the fields `abschluss_schluessel`, `studienfach_schluessel`, and `fachkennzeichen_schluessel`, to an atomic surrogate key `studiengang_schluessel`. Code that refers to a `Bewerbung` or any other `Studiengang` related object needs to be changed to use only the atomic key for identification.

For historic reasons, there's still a compound field containing the atomic key `studiengang_schluessel`, and additional fields `abschluss_schluessel`, `studienfach_schluessel`, and `fachkennzeichen_schluessel`. Even though only the first field `studiengang_schluessel`

serves as the identifying atomic key, the Servicestelle API preserved the compound structure for the reason of refactoring costs.

Changed several function signatures of `sestco.sestclient`

The following public API method signatures of the *sestclient* module changed:

- *ServiceStelle*'s parameter `location` is no longer optional. The signature changed from:

  __init__(*self*, *username*, *password*, *location='https://dosv.hochschulstart.de/hochschule/webservice'*, *proxy=None*, *ciphers=None*)

  to:

  __init__(*self*, *username*, *password*, *location*, *proxy=None*, *ciphers=None*)

- *new_bewerbung_einfach()*, from

  new_bewerbung_einfach(*self*, *studiengang*, *bewerber_id*, *eingangszeitpunkt*, *bearbeitungsstatus*, *ort_des_eingangs='HS'*, *version_sest=None*, *bewerber_ban=None*, *letzte_aenderung=None*, *aktivitaetsstatus=None*, *statusanmerkung=''*, *sonderantragsentscheidung=''*)

  to

  new_bewerbung_einfach(*self*, *studiengang*, *bewerber_id*, *eingangszeitpunkt*, *bearbeitungsstatus*, *version_sest=None*, *bewerber_ban=None*, *aktivitaetsstatus=None*, *statusanmerkung=''*, *sonderantragsentscheidung=''*)

  The tuple of parameter `studiengang` now contains `studiengang_schluessel`, the atomic primary key of `Studiengang`:

```
(
    studiengang_schluessel,
    abschluss_schluessel,
    studienfach_schluessel,
    fachkennzeichen_schluessel
)
```

- *new_bewerbung_teilfach()*, from:

  new_bewerbung_teilfach(*self*, *studiengang*, *bewerber_id*, *eingangszeitpunkt*, *bearbeitungsstatus*, *ort_des_eingangs='HS'*, *version_sest=None*, *hauptfach=None*, *ersatzfach=None*, *ist_ersatzfach_akzeptabel=False*, *ist_fachwechsel=False*, *letzte_aenderung=None*, *aktivitaetsstatus=None*, *statusanmerkung=''*, *sonderantragsentscheidung=''*)

  to

  new_bewerbung_teilfach(*self*, *studiengang*, *bewerber_id*, *eingangszeitpunkt*, *bearbeitungsstatus*, *version_sest=None*, *hauptfach=None*, *ersatzfach=None*, *ist_ersatzfach_akzeptabel=False*, *ist_fachwechsel=False*, *aktivitaetsstatus=None*, *statusanmerkung=''*, *sonderantragsentscheidung=''*)

  The tuple of parameter `studiengang` now contains `studiengang_schluessel`, the atomic primary key of `Studiengang`:

```
(
    studiengang_schluessel,
    abschluss_schluessel,
    studienfach_schluessel,
    fachkennzeichen_schluessel
)
```

- *new_bewerbung_mehrfach()*, from

  new_bewerbung_mehrfach(*self*, *teilfachbewerbungen*, *abschluss_schluessel*, *bewerber_id*, *eingangszeitpunkt*, *bearbeitungsstatus*, *ort_des_eingangs='HS'*, *version_sest=None*, *bewerber_ban=None*, *letzte_aenderung=None*, *aktivitaetsstatus=None*, *statusanmerkung=''*, *sonderantragsentscheidung=''*)

  to

  new_bewerbung_mehrfach(*self*, *teilfachbewerbungen*, *abschluss_schluessel*, *bewerber_id*, *eingangszeitpunkt*, *bearbeitungsstatus*, *version_sest=None*, *bewerber_ban=None*, *aktivitaetsstatus=None*, *statusanmerkung=''*, *sonderantragsentscheidung=''*)

Updated to Django 1.11

Updated from Django 1.7 to Django 1.11. Skipping several feature versions comes with a number of backwards incompatible changes that need adjustments in university specific modules.

settings.py

- All sequences are now lists (not tuples).
- Add middleware:

```
'django.middleware.security.SecurityMiddleware',
```

- Add TEMPLATE setting:

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

- Add password validation setting:

```
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.
↪UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator
↪',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.
↪CommonPasswordValidator',
    },
```

```
    {
        'NAME': 'django.contrib.auth.password_validation.
↪NumericPasswordValidator',
    },
]
```

admin.py

Import of `GenericTabularInline` changed from:

```
from django.contrib.contenttypes.generic import GenericTabularInline
```

to:

```
from django.contrib.contenttypes.admin import GenericTabularInline
```

urls.py

- Replace import:

  ```
  from django.conf.urls import patterns, include, url
  ```

  by:

  ```
  from django.conf.urls import url
  ```

- Replace URL definition:

  ```
  urlpatterns = patterns(
      # ...
  )
  ```

  by:

  ```
  urlpatterns = [
      # ...
  ]
  ```

models.py

- Import of `GenericForeignKey` changed from:

  ```
  from django.contrib.contenttypes.generic import GenericForeignKey
  ```

  to:

  ```
  from django.contrib.contenttypes.fields import GenericForeignKey
  ```

management commands

The parse library changed.

Instead of an `option_list` attribute, use `add_arguments(self, parser)` and `parser.add_argument()`. See the Django 1.11 docs on management commands.

Note that in `handle()`, positional arguments are now contained in `**options` together with the optional arguments.

`handlers.py`

There might be several changes of Django that affect your university specific `handlers.py`. It is your responsibility to check Django's release notes for the 1.8, 1.9, 1.10 and 1.11 versions and test your app.

`from sestco.conf import *` in settings

It is now necessary to add `from sestco.conf import *` to your project's `settings.py` due to the new feature of providing centralized default settings. Other ways of providing centralized default settings were discussed but were rated as coming with more severe compromises.

Features deprecated in 0.3

None so far.

Features removed in 0.3

- Support for Servicestelle version 1.x has been dropped in favor of Servicestelle version 3. The default webservice endpoint is now:

```
https://dosv.hochschulstart.de/hochschule/webservice/3/
```

Sestco 0.2 release notes

August 11, 2014

Welcome to Sestco 0.2!

These release notes cover the *new features*, as well as some *backwards incompatible changes* you'll want to be aware of when upgrading from Sestco 0.1 or older versions. We've also dropped some features, which are detailed in *our deprecation plan*.

What's new in Sestco 0.2

For end users

Option to select a cipher for SSL

Added option to select a cipher for SSL in case the operating system doesn't select one of accepted by the Servicestelle. See `SECO_CIPHERS`.

Support for "Explizite Studiengangskombination"

Support has been added for the "Explizite Studiengangskombination" type of the `MehrfachstudienangebotsBewerbung`, especially regarding to download and the database model.

Support for `Bewerbungsbestandteile`

`Bewerbungsbestandteile` has been added to the model and is now downloaded from the Servicestelle. If one of the content is a document, a link to a `Dokument` object is set, so it can be downloaded together with other `Dokument` items in a separate step.

Download of Dokument items

Documents are downloaded now. Two settings variables have been introduced for that, *`SECO_DOKUMENT_ROOT`* and *`SECO_DOKUMENT_DOWNLOAD`*. Furthermore, the *`sestco.pysest.handlers.BaseUnterstuetzungHandler`* has been introduced, and must be subclassed in the university specific handler, but no methods need to be implemented.

Due to a bug at the Servicestelle webservice interface, the download is only included on a full update. See *Option to update all data*.

Support for Bescheid

Bescheid type is now part of the local Servicestelle model, and attached documents are downloaded.

Option to update all data

The `download_sest()` command now has a `--all` option to update not only the data changed since the last download, but all data.

For programers

PEP 8 compliance

Made all the code PEP 8 compliant. This makes it much easier to keep it consistent between programmers and to check via Flake8 before committing to the main line.

Backwards incompatible changes in 0.2

Changed various names

- Module `pysest` has been renamed to `sestco`, and `sestco.sfhai` to `sestco.pysest`.
- Management command `download_sfh` has been renamed to *`download_sest`*, and `sync_bews` has been renamed to *`sync_uni`*.
- Renamed `sfhai_xxx` to `sest_xxx`.
- Module `sestco.pysest.models.sfh` has been renamed to *`sestco.pysest.models.sest`*.
- *`sestco.pysest.models.sest.Bewerbung`* methods `can_hs_get_xxx` have been renamed to `can_uni_get_xxx`.
- Renamed settings variables:
  - `SFH_xxx` to `SECO_xxx`
  - `HS_xxx` to `SECO_xxx`

Updated to Django 1.7

Version 1.7 of Django is required. Check the Django release notes for its backwards-incompatible changes as well.

Introduced `BaseUnterstuetzungHandler`

The *sestco.pysest.handlers.BaseUnterstuetzungHandler* has been introduced, and must be subclassed in the university specific handler, but no methods need to be implemented.

Sestco 0.1 release notes

Changes for end users:

- pysest: Web–GUI based admin interface for browsing and filtering data of local Servicestelle copy.
- pysest: Shell management command for retrieving data from SfH via webservice, converting it to the local database schema and saving it. For now, Bewerber data and Einfachstudienganege are supported.
- Support for Rangliste.
- New management command: *sestco.pysest.management.commands.rang_upload_schema*.
- New management command: *sestco.pysest.management.commands.rang_upload_eintraege*.
- Admin pages for Rangliste process (upload and download to/from Servicestelle).
- Renamed management commands (now `download_sest`).
- Download Mehrfachstudiengaenge from Servicestelle and show them in the admin interface.
- Logging. Administrators can now define of which severity level messages should be shown on console, logged to a file or sent by mail. Around 200 log messages cover the whole code.
- sosposgx: Command line interface.
- sosposgx: Command for exporting data to the text format the GX application requires and to hundreds of different encodings.
- Raise custom Command errors instead of "internal errors" when the error for sure comes from an external reason.

Changes for administrators:

- pysest: Introduced several setting constants.
- Corrected `SECO_WEBSERVICE` settings check bug.

Changes for API users:

- sestclient: SfH SOAP webservice specific client wrapper.
- pysest: Database model to save a copy of all university relevant SfH data.
- Support for Bewerberauswahl (SOAP client *sestco.sestclient*).
- Support for Mehrfachstudiengaenge (SOAP client *sestco.sestclient*).
- Download Mehrfachstudiengaenge from Servicestelle.
- `delete` method for *sestco.pysest.models.utils.Variable* and *sestco.pysest.utils.Timestamp*.
- Support for Ranglisten: fetching Servicestelle and saving to local database.

- Added `sestco.pysest.commands.BaseSestCommand` class for easy implementation of Servicestelle communicating console commands.
- `sestco.pysest.commands.BaseRangCommand` class for implementing console commands to upload Ranglisten data easily.
- Refactored whole code for providing an interface that custom modules can plug in easily and have access to all the general functionalities. All management commands are placed in the core module now, and custom modules only need to implement handler classes that provide methods with custom algorithms.
- Introduced a configurable schema with defaults that allows easily checking if a Bewerbung can get out or in from the current status to another status.

Changes for core developers

- Changed to a better test structure.
- Added test API and integration test for download_sest.

Bugfixes

- Fixed BID clean up method to match right entries.

6.2 Security releases

Whenever a security issue is disclosed, appropriate release notes are now added to all affected release series.

Additionally, *an archive of disclosed security issues* is maintained.

6.3 Development releases

These notes are retained for historical purposes. If you are upgrading between formal releases, you don't need to worry about these notes.

6.3.1 Archive of security issues

Sestco's development team is strongly committed to responsible reporting and disclosure of security-related issues.

As part of that commitment, we maintain the following historical list of issues which have been fixed and disclosed. For each issue, the list below includes the date, a brief description, the CVE identifier if applicable, a list of affected versions, a link to the full disclosure and links to the appropriate patch(es).

Until now, no security fixes had to be made.

6.3.2 Sestco 0.1a5 release notes

April 24, 2012

Changes for end users:

- pysest: Web–GUI based admin interface for browsing and filtering data of local Servicestelle copy.
- pysest: Shell management command for retrieving data from SfH via webservice, converting it to the local database schema and saving it. For now, Bewerber data and Einfachstudienganege are supported.
- Support for Rangliste.
- New managment command: `pysest.management.commands.rang_upload_schema`.
- New managment command: `pysest.management.commands.rang_upload_entries`.
- Admin pages for Rangliste process (upload and download to/from Servicestelle).

- Renamed management commands (now `download_sest`).
- Download Mehrfachstudiengaenge from Servicestelle and show them in the admin interface.
- Logging. Administrators can now define of which severity level messages should be shown on console, logged to a file or sent by mail. Around 200 log messages cover the whole code.
- sosposgx: Command line interface.
- sosposgx: Command for exporting data to the text format the GX application requires and to hundreds of different encodings.
- Raise custom Command errors instead of "internal errors" when the error for sure comes from an external reason.
- Added support for field `sonderantragsentscheidung`.

Changes for administrators:

- pysest: Introduced several setting constants.
- Corrected `SECO_WEBSERVICE` settings check bug.

Changes for API users:

- sestclient: SfH SOAP webservice specific client wrapper.
- pysest: Database model to save a copy of all university relevant SfH data.
- Support for Bewerberauswahl (SOAP client *`sestco.sestclient`*).
- Support for Mehrfachstudiengaenge (SOAP client *`sestco.sestclient`*).
- Download Mehrfachstudiengaenge from Servicestelle.
- `delete` method for *`sestco.pysest.models.utils.Variable`* and *`sestco.pysest.utils.Timestamp`*.
- Support for Ranglisten: fetching Servicestelle and saving to local database.
- `sestco.pysest.management.commands.rang_upload_data.BaseRangCommand` class for implementing console commands to upload Ranglisten data easily.
- Refactored whole code for providing an interface that custom modules can plug in easily and have access to all the general functionalities. All management commands are placed in the core module now, and custom modules only need to implement handler classes that provide methods with custom algorithms.
- Introduced a configurable schema with defaults that allows easily checking if a Bewerbung can get out or in from the current status to another status.

6.3.3 Sestco 0.1a4 release notes

April 14, 2012

Changes for end users:

- pysest: Web–GUI based admin interface for browsing and filtering data of local Servicestelle copy.
- pysest: Shell management command for retrieving data from SfH via webservice, converting it to the local database schema and saving it. For now, Bewerber data and Einfachstudienganege are supported.
- Support for Rangliste.
- New managment command: `pysest.management.commands.rang_upload_schema`.
- New managment command: `pysest.management.commands.rang_upload_entries`.
- Admin pages for Rangliste process (upload and download to/from Servicestelle).
- Renamed management commands (now `download_sest`).
- Download Mehrfachstudiengaenge from Servicestelle and show them in the admin interface.

- Logging. Administrators can now define of which severity level messages should be shown on console, logged to a file or sent by mail. Around 200 log messages cover the whole code.
- sosposgx: Command line interface.
- sosposgx: Command for exporting data to the text format the GX application requires and to hundreds of different encodings.

Changes for administrators:

- pysest: Introduced several setting constants.

Changes for API users:

- sestclient: SfH SOAP webservice specific client wrapper.
- pysest: Database model to save a copy of all university relevant SfH data.
- Support for Bewerberauswahl (SOAP client *sestco.sestclient*).
- Support for Mehrfachstudiengaenge (SOAP client *sestco.sestclient*).
- Download Mehrfachstudiengaenge from Servicestelle.
- `delete` method for *sestco.pysest.models.utils.Variable* and *sestco.pysest.utils.Timestamp*.
- Support for Ranglisten: fetching Servicestelle and saving to local database.
- `sestco.pysest.management.commands.rang_upload_data.BaseRangCommand` class for implementing console commands to upload Ranglisten data easily.
- Refactored whole code for providing an interface that custom modules can plug in easily and have access to all the general functionalities. All management commands are placed in the core module now, and custom modules only need to implement handler classes that provide methods with custom algorithms.
- Introduced a configurable schema with defaults that allows easily checking if a Bewerbung can get out or in from the current status to another status.

6.3.4 Sestco 0.1a3 release notes

March 2, 2012

Mostly corrected minor bugs in `pysest` to make the code run reliably.

Changes for end users:

- pysest: Web–GUI based admin interface for browsing and filtering data of local Servicestelle copy.
- pysest: Shell management command for retrieving data from SfH via webservice, converting it to the local database schema and saving it. For now, Bewerber data and Einfachstudienganege are supported.
- Support for Rangliste.
- New managment command: `pysest.management.commands.rang_upload_schema`.
- New managment command: `pysest.management.commands.rang_upload_entries`.
- Admin pages for Rangliste process (upload and download to/from Servicestelle).
- Renamed management commands (now `download_sest`).
- Download Mehrfachstudiengaenge from Servicestelle and show them in the admin interface.
- Logging. Administrators can now define of which severity level messages should be shown on console, logged to a file or sent by mail. Around 200 log messages cover the whole code.

Changes for administrators:

- pysest: Introduced several setting constants.

Changes for API users:

- sestclient: SfH SOAP webservice specific client wrapper.
- pysest: Database model to save a copy of all university relevant SfH data.
- Support for Bewerberauswahl (SOAP client *sestco.sestclient*).
- Support for Mehrfachstudiengaenge (SOAP client *sestco.sestclient*).
- Download Mehrfachstudiengaenge from Servicestelle.
- `delete` method for *sestco.pysest.models.utils.Variable* and *sestco.pysest.utils.Timestamp*.
- Support for Ranglisten: fetching Servicestelle and saving to local database.
- `sestco.pysest.management.commands.rang_upload_data.BaseRangCommand` class for implementing console commands to upload Ranglisten data easily.
- Refactored whole code for providing an interface that custom modules can plug in easily and have access to all the general functionalities. All management commands are placed in the core module now, and custom modules only need to implement handler classes that provide methods with custom algorithms.
- Introduced a configurable schema with defaults that allows easily checking if a Bewerbung can get out or in from the current status to another status.

6.3.5 Sestco 0.1a2 release notes

January 10, 2012

Changes for end users:

- pysest: Web–GUI based admin interface for browsing and filtering data of local Servicestelle copy.
- pysest: Shell management command for retrieving data from SfH via webservice, converting it to the local database schema and saving it. For now, Bewerber data and Einfachstudienganege are supported.
- Support for Rangliste.
- New managment command: `pysest.management.commands.rang_upload_schema`.
- New managment command: `pysest.management.commands.rang_upload_entries`.
- Admin pages for Rangliste process (upload and download to/from Servicestelle).
- Renamed management commands (now `download_sest`).
- Download Mehrfachstudiengaenge from Servicestelle and show them in the admin interface.
- Logging. Administrators can now define of which severity level messages should be shown on console, logged to a file or sent by mail. Around 200 log messages cover the whole code.

Changes for administrators:

- pysest: Introduced several setting constants.

Changes for API users:

- sestclient: SfH SOAP webservice specific client wrapper.
- pysest: Database model to save a copy of all university relevant SfH data.
- Support for Bewerberauswahl (SOAP client *sestco.sestclient*).
- Support for Mehrfachstudiengaenge (SOAP client *sestco.sestclient*).
- Download Mehrfachstudiengaenge from Servicestelle.
- `delete` method for *sestco.pysest.models.utils.Variable* and *sestco.pysest.utils.Timestamp*.
- Support for Ranglisten: fetching Servicestelle and saving to local database.

- `sestco.pysest.management.commands.rang_upload_data.BaseRangCommand` class for implementing console commands to upload Ranglisten data easily.
- Refactored whole code for providing an interface that custom modules can plug in easily and have access to all the general functionalities. All management commands are placed in the core module now, and custom modules only need to implement handler classes that provide methods with custom algorithms.
- Introduced a configurable schema with defaults that allows easily checking if a Bewerbung can get out or in from the current status to another status.

6.3.6 Sestco 0.1a1 release notes

August 20, 2011

Changes for end users:

- pysest: Web–GUI based admin interface for browsing and filtering data of local Servicestelle copy.
- pysest: Shell management command for retrieving data from SfH via webservice, converting it to the local database schema and saving it. For now, Bewerber data and Einfachstudienganege are supported.

Changes for administrators:

- pysest: Introduced several setting constants.

Changes for API users:

- sestclient: SfH SOAP webservice specific client wrapper.
- pysest: Database model to save a copy of all university relevant SfH data.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

Symbols

A

B

C

D

F

G

H

I

L

T

U

V

W