django-cas-server Documentation

Release 0.6.4

Valentin Samir

August 14, 2016

Contents

Contents:

CHAPTER 1

CAS Server

CAS Server is a Django application implementing the CAS Protocol 3.0 Specification.

By default, the authentication process use django internal users but you can easily use any sources (see auth classes in the auth.py file)

Table of Contents

1.1 Features

- Support CAS version 1.0, 2.0, 3.0
- Support Single Sign Out
- Configuration of services via the django Admin application
- Fine control on which user's attributes are passed to which service
- Possibility to rename/rewrite attributes per service

- Possibility to require some attribute values per service
- Federated mode between multiple CAS
- Supports Django 1.7, 1.8 and 1.9
- Supports Python 2.7, 3.x

1.2 Dependencies

`django-cas-server` depends on the following python packages:

- Django >= 1.7.1 < 1.10
- requests >= 2.4
- requests_futures >= 0.9.5
- lxml >= 3.4
- six >= 1.8

Minimal version of packages dependancy are just indicative and meens that `django-cas-server` has been tested with it. Previous versions of dependencies may or may not work.

Additionally, denpending of the authentication backend you plan to use, you may need the following python packages:

- ldap3
- psycopg2
- mysql-python

Here there is a table with the name of python packages and the corresponding packages providing them on debian like systems and centos like systems. You should try as much as possible to use system packages as there are automatically updated then you update your system. You can then install Not Available (N/A) packages on your system using pip inside a virtualenv as described in the *Installation* section. For use with python3, just replace python(2) in the table by python3.

python package	debian like systems	centos like systems
Django	python-django	python-django
requests	python-requests	python-requests
requests_futures	python-requests-futures	N/A
lxml	python-lxml	python-lxml
six	python-six	python-six
ldap3	python-ldap3	python-ldap3
psycopg2	python-psycopg2	python-psycopg2
mysql-python	python-mysqldb	python2-mysql

1.3 Installation

The recommended installation mode is to use a virtualenv with `--system-site-packages`

1. Make sure that python virtualenv is installed
2. Install python packages available via the system package manager:

   On debian like systems:

```
$ sudo apt-get install python-django python-requests python-six python-lxml python-requests-futu
```

On debian jessie, you can use the version of python-django available in the backports.

On centos like systems:

```
$ sudo yum install python-django python-requests python-six python-lxml
```

3. Create a virtualenv:

```
$ virtualenv --system-site-packages cas_venv
Running virtualenv with interpreter /var/www/html/cas-server/bin/python2
Using real prefix '/usr'
New python executable in cas/bin/python2
Also creating executable in cas/bin/python
Installing setuptools, pip...done.
```

4. And activate it:

```
$ cd cas_venv/; . bin/activate
```

5. Create a django project:

```
$ django-admin startproject cas_project
$ cd cas_project
```

6. Install *django-cas-server*. To use the last published release, run:

```
$ pip install django-cas-server
```

Alternatively if you want to use the version of the git repository, you can clone it:

```
$ git clone https://github.com/nitmir/django-cas-server
$ cd django-cas-server
$ pip install -r requirements.txt
```

Then, either run `make install` to create a python package using the sources of the repository and install it with pip, or place the `cas_server` directory into your PYTHONPATH (for instance by symlinking `cas_server` to the root of your django project).

7. Open `cas_project/settings.py` in you favourite editor and follow the quick start section.

1.4 Quick start

1. Add "cas_server" to your INSTALLED_APPS setting like this:

```
INSTALLED_APPS = (
    'django.contrib.admin',
    ...
    'cas_server',
)
```

For internationalization support, add "django.middleware.locale.LocaleMiddleware" to your MIDDLEWARE_CLASSES setting like this:

```
MIDDLEWARE_CLASSES = (
    ...
    'django.middleware.locale.LocaleMiddleware',
```

```
        ...
    )
```

2. Include the cas_server URLconf in your project urls.py like this:

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

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    ...
    url(r'^cas/', include('cas_server.urls', namespace="cas_server")),
]
```

3. Run `python manage.py migrate` to create the cas_server models.
4. You should add some management commands to a crontab: `clearsessions`, `cas_clean_tickets` and `cas_clean_sessions`.
   - `clearsessions`: please see Clearing the session store.
   - `cas_clean_tickets`: old tickets and timed-out tickets do not get purge from the database automatically. They are just marked as invalid. `cas_clean_tickets` is a clean-up management command for this purpose. It send SingleLogOut request to services with timed out tickets and delete them.
   - `cas_clean_sessions`: Logout and purge users (sending SLO requests) that are inactive since more than `SESSION_COOKIE_AGE`. The default value for is `1209600` seconds (2 weeks). You probably should reduce it to something like `86400` seconds (1 day).

   You could for example do as bellow :
5. Run `python manage.py createsuperuser` to create an administrator user.
6. Start the development server and visit http://127.0.0.1:8000/admin/ to add a first service allowed to authenticate user against the CAS (you'll need the Admin app enabled). See the *Service Patterns* section bellow.
7. Visit http://127.0.0.1:8000/cas/ to login with your django users.

1.5 Settings

All settings are optional. Add them to `settings.py` to customize `django-cas-server`:

1.5.1 Template settings

- `CAS_LOGO_URL`: URL to the logo showed in the up left corner on the default templates. Set it to `False` to disable it.
- `CAS_FAVICON_URL`: URL to the favicon (shortcut icon) used by the default templates. Default is a key icon. Set it to `False` to disable it.
- `CAS_SHOW_POWERED`: Set it to `False` to hide the powered by footer. The default is `True`.
- `CAS_COMPONENT_URLS`: URLs to css and javascript external components. It is a dictionnary and it must have the five following keys: `"bootstrap3_css"`, `"bootstrap3_js"`, `"html5shiv"`, `"respond"`, `"jquery"`. The default is:

```
{
    "bootstrap3_css": "//maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css",
    "bootstrap3_js": "//maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js",
    "html5shiv": "//oss.maxcdn.com/libs/html5shiv/3.7.0/html5shiv.js",
```

```
        "respond": "//oss.maxcdn.com/libs/respond.js/1.4.2/respond.min.js",
        "jquery": "//code.jquery.com/jquery.min.js",
    }
```

- `CAS_LOGIN_TEMPLATE`: Path to the template showed on `/login` then the user is not autenticated. The default is `"cas_server/login.html"`.
- `CAS_WARN_TEMPLATE`: Path to the template showed on `/login?service=...` then the user is authenticated and has asked to be warned before being connected to a service. The default is `"cas_server/warn.html"`.
- `CAS_LOGGED_TEMPLATE`: Path to the template showed on `/login` then to user is authenticated. The default is `"cas_server/logged.html"`.
- `CAS_LOGOUT_TEMPLATE`: Path to the template showed on `/logout` then to user is being disconnected. The default is `"cas_server/logout.html"`
- `CAS_REDIRECT_TO_LOGIN_AFTER_LOGOUT`: Should we redirect users to */login* after they logged out instead of displaying `CAS_LOGOUT_TEMPLATE`. The default is `False`.

1.5.2 Authentication settings

- `CAS_AUTH_CLASS`: A dotted path to a class or a class implementing `cas_server.auth.AuthUser`. The default is `"cas_server.auth.DjangoAuthUser"` Available classes bundled with `django-cas-server` are listed below in the *Authentication backend* section.
- `SESSION_COOKIE_AGE`: This is a django settings. Here, it control the delay in seconds after which inactive users are logged out. The default is `1209600` (2 weeks). You probably should reduce it to something like `86400` seconds (1 day).
- `CAS_PROXY_CA_CERTIFICATE_PATH`: Path to certificate authorities file. Usually on linux the local CAs are in `/etc/ssl/certs/ca-certificates.crt`. The default is `True` which tell requests to use its internal certificat authorities. Settings it to `False` should disable all x509 certificates validation and MUST not be done in production. x509 certificate validation is perform upon PGT issuance.
- `CAS_SLO_MAX_PARALLEL_REQUESTS`: Maximum number of parallel single log out requests send. If more requests need to be send, there are queued. The default is `10`.
- `CAS_SLO_TIMEOUT`: Timeout for a single SLO request in seconds. The default is `5`.

1.5.3 Federation settings

- `CAS_FEDERATE`: A boolean for activating the federated mode (see the *Federation mode* section below). The default is `False`.
- `CAS_FEDERATE_REMEMBER_TIMEOUT`: Time after witch the cookie use for "remember my identity provider" expire. The default is `604800`, one week. The cookie is called `_remember_provider`.

1.5.4 New version warnings settings

- `CAS_NEW_VERSION_HTML_WARNING`: A boolean for diplaying a warning on html pages then a new version of the application is avaible. Once closed by a user, it is not displayed to this user until the next new version. The default is `True`.
- `CAS_NEW_VERSION_EMAIL_WARNING`: A bolean sot sending a email to `settings.ADMINS` when a new version is available. The default is `True`.

1.5.5 Tickets validity settings

- `CAS_TICKET_VALIDITY`: Number of seconds the service tickets and proxy tickets are valid. This is the maximal time between ticket issuance by the CAS and ticket validation by an application. The default is `60`.
- `CAS_PGT_VALIDITY`: Number of seconds the proxy granting tickets are valid. The default is `3600` (1 hour).
- `CAS_TICKET_TIMEOUT`: Number of seconds a ticket is kept in the database before sending Single Log Out request and being cleared. The default is `86400` (24 hours).

1.5.6 Tickets miscellaneous settings

- `CAS_TICKET_LEN`: Default ticket length. All CAS implementation MUST support ST and PT up to 32 chars, PGT and PGTIOU up to 64 chars and it is RECOMMENDED that all tickets up to 256 chars are supports. Here the default is `64`.
- `CAS_LT_LEN`: Length of the login tickets. Login tickets are only processed by `django-cas-server` thus there is no length restriction on it. The default is `CAS_TICKET_LEN`.
- `CAS_ST_LEN`: Length of the service tickets. The default is `CAS_TICKET_LEN`. You may need to lower is to `32` if you use some old clients.
- `CAS_PT_LEN`: Length of the proxy tickets. The default is `CAS_TICKET_LEN`. This length should be the same as `CAS_ST_LEN`. You may need to lower is to `32` if you use some old clients.
- `CAS_PGT_LEN`: Length of the proxy granting tickets. The default is `CAS_TICKET_LEN`.
- `CAS_PGTIOU_LEN`: Length of the proxy granting tickets IOU. The default is `CAS_TICKET_LEN`.
- `CAS_LOGIN_TICKET_PREFIX`: Prefix of login tickets. The default is `"LT"`.
- `CAS_SERVICE_TICKET_PREFIX`: Prefix of service tickets. The default is `"ST"`. The CAS specification mandate that service tickets MUST begin with the characters ST so you should not change this.
- `CAS_PROXY_TICKET_PREFIX`: Prefix of proxy ticket. The default is `"PT"`.
- `CAS_PROXY_GRANTING_TICKET_PREFIX`: Prefix of proxy granting ticket. The default is `"PGT"`.
- `CAS_PROXY_GRANTING_TICKET_IOU_PREFIX`: Prefix of proxy granting ticket IOU. The default is `"PGTIOU"`.

1.5.7 Mysql backend settings

Deprecated, see the *Sql backend settings*. Only usefull if you are using the mysql authentication backend:

- `CAS_SQL_HOST`: Host for the SQL server. The default is `"localhost"`.
- `CAS_SQL_USERNAME`: Username for connecting to the SQL server.
- `CAS_SQL_PASSWORD`: Password for connecting to the SQL server.
- `CAS_SQL_DBNAME`: Database name.
- `CAS_SQL_DBCHARSET`: Database charset. The default is `"utf8"`
- `CAS_SQL_USER_QUERY`: The query performed upon user authentication. The username must be in field `username`, the password in `password`, additional fields are used as the user attributes. The default is `"SELECT user AS username, pass AS password, users.* FROM users WHERE user = %s"`
- `CAS_SQL_PASSWORD_CHECK`: The method used to check the user password. Must be one of the following:

  - `"crypt"` (see <https://en.wikipedia.org/wiki/Crypt_(C)>), the password in the database should begin this $
  - `"ldap"` (see https://tools.ietf.org/id/draft-stroeder-hashed-userpassword-values-01.html) the password in the database must begin with one of {MD5}, {SMD5}, {SHA}, {SSHA}, {SHA256}, {SSHA256}, {SHA384}, {SSHA384}, {SHA512}, {SSHA512}, {CRYPT}.
  - `"hex_HASH_NAME"` with `HASH_NAME` in md5, sha1, sha224, sha256, sha384, sha512. The hashed password in the database is compare to the hexadecimal digest of the clear password hashed with the corresponding algorithm.
  - `"plain"`, the password in the database must be in clear.

  The default is `"crypt"`.

1.5.8 Sql backend settings

Only usefull if you are using the sql authentication backend. You must add a `"cas_server"` database to settings.DATABASES as defined in the django documentation. It is then the database use by the sql backend.

- `CAS_SQL_USER_QUERY`: The query performed upon user authentication. The username must be in field `username`, the password in `password`, additional fields are used as the user attributes. The default is `"SELECT user AS username, pass AS password, users.* FROM users WHERE user = %s"`
- `CAS_SQL_PASSWORD_CHECK`: The method used to check the user password. Must be one of the following:
  - `"crypt"` (see <https://en.wikipedia.org/wiki/Crypt_(C)>), the password in the database should begin this $
  - `"ldap"` (see https://tools.ietf.org/id/draft-stroeder-hashed-userpassword-values-01.html) the password in the database must begin with one of {MD5}, {SMD5}, {SHA}, {SSHA}, {SHA256}, {SSHA256}, {SHA384}, {SSHA384}, {SHA512}, {SSHA512}, {CRYPT}.
  - `"hex_HASH_NAME"` with `HASH_NAME` in md5, sha1, sha224, sha256, sha384, sha512. The hashed password in the database is compare to the hexadecimal digest of the clear password hashed with the corresponding algorithm.
  - `"plain"`, the password in the database must be in clear.

  The default is `"crypt"`.
- `CAS_SQL_PASSWORD_CHARSET`: Charset the SQL users passwords was hash with. This is needed to encode the user sended password before hashing it for comparison. The default is `"utf-8"`.

1.5.9 Ldap backend settings

Only usefull if you are using the ldap authentication backend:

- `CAS_LDAP_SERVER`: Address of the LDAP server. The default is `"localhost"`.
- `CAS_LDAP_USER`: User bind address, for example `"cn=admin,dc=crans,dc=org"` for connecting to the LDAP server.
- `CAS_LDAP_PASSWORD`: Password for connecting to the LDAP server.
- `CAS_LDAP_BASE_DN`: LDAP search base DN, for example `"ou=data,dc=crans,dc=org"`.
- `CAS_LDAP_USER_QUERY`: Search filter for searching user by username. User inputed usernames are escaped using `ldap3.utils.conv.escape_bytes`. The default is `"(uid=%s)"`

- `CAS_LDAP_USERNAME_ATTR`: Attribute used for users usernames. The default is `"uid"`
- `CAS_LDAP_PASSWORD_ATTR`: Attribute used for users passwords. The default is `"userPassword"`
- `CAS_LDAP_PASSWORD_CHECK`: The method used to check the user password. Must be one of the following:
  - `"crypt"` (see <https://en.wikipedia.org/wiki/Crypt_(C)>), the password in the database should begin this $
  - `"ldap"` (see https://tools.ietf.org/id/draft-stroeder-hashed-userpassword-values-01.html) the password in the database must begin with one of {MD5}, {SMD5}, {SHA}, {SSHA}, {SHA256}, {SSHA256}, {SHA384}, {SSHA384}, {SHA512}, {SSHA512}, {CRYPT}.
  - `"hex_HASH_NAME"` with `HASH_NAME` in md5, sha1, sha224, sha256, sha384, sha512. The hashed password in the database is compare to the hexadecimal digest of the clear password hashed with the corresponding algorithm.
  - `"plain"`, the password in the database must be in clear.

  The default is `"ldap"`.
- `CAS_LDAP_PASSWORD_CHARSET`: Charset the LDAP users passwords was hash with. This is needed to encode the user sended password before hashing it for comparison. The default is `"utf-8"`.

1.5.10 Test backend settings

Only usefull if you are using the test authentication backend:

- `CAS_TEST_USER`: Username of the test user. The default is `"test"`.
- `CAS_TEST_PASSWORD`: Password of the test user. The default is `"test"`.
- `CAS_TEST_ATTRIBUTES`: Attributes of the test user. The default is `{'nom': 'Nymous', 'prenom': 'Ano', 'email': 'anonymous@example.net', 'alias': ['demo1', 'demo2']}`.

1.6 Authentication backend

`django-cas-server` comes with some authentication backends:

- dummy backend `cas_server.auth.DummyAuthUser`: all authentication attempt fails.
- test backend `cas_server.auth.TestAuthUser`: username, password and returned attributes for the user are defined by the `CAS_TEST_*` settings.
- django backend `cas_server.auth.DjangoAuthUser`: Users are authenticated against django users system. This is the default backend. The returned attributes are the fields available on the user model.
- mysql backend `cas_server.auth.MysqlAuthUser`: Deprecated, use the sql backend instead. see the *Mysql backend settings* section. The returned attributes are those return by sql query `CAS_SQL_USER_QUERY`.
- sql backend `cas_server.auth.SqlAuthUser`: see the *Sql backend settings* section. The returned attributes are those return by sql query `CAS_SQL_USER_QUERY`.
- ldap backend `cas_server.auth.LdapAuthUser`: see the *Ldap backend settings* section. The returned attributes are those of the ldap node returned by the query filter `CAS_LDAP_USER_QUERY`.
- federated backend `cas_server.auth.CASFederateAuth`: It is automatically used then `CAS_FEDERATE` is `True`. You should not set it manually without setting `CAS_FEDERATE` to `True`.

1.7 Logs

`django-cas-server` logs most of its actions. To enable login, you must set the `LOGGING` (https://docs.djangoproject.com/en/stable/topics/logging) variable in `settings.py`.

Users successful actions (login, logout) are logged with the level `INFO`, failures are logged with the level `WARNING` and user attributes transmitted to a service are logged with the level `DEBUG`.

For example to log to syslog you can use :

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'formatters': {
        'cas_syslog': {
            'format': 'cas: %(levelname)s %(message)s'
        },
    },
    'handlers': {
        'cas_syslog': {
            'level': 'INFO',
            'class': 'logging.handlers.SysLogHandler',
            'address': '/dev/log',
            'formatter': 'cas_syslog',
        },
    },
    'loggers': {
        'cas_server': {
            'handlers': ['cas_syslog'],
            'level': 'INFO',
            'propagate': True,
        },
    },
}
```

Or to log to a file:

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'formatters': {
        'cas_file': {
            'format': '%(asctime)s %(levelname)s %(message)s'
        },
    },
    'handlers': {
        'cas_file': {
            'level': 'INFO',
            'class': 'logging.FileHandler',
            'filename': '/tmp/cas_server.log',
            'formatter': 'cas_file',
        },
    },
    'loggers': {
        'cas_server': {
            'handlers': ['cas_file'],
            'level': 'INFO',
            'propagate': True,
        },
```

```
    },
}
```

1.8 Service Patterns

In a CAS context, `Service` refers to the application the client is trying to access. By extension we use `service` for the URL of such an application.

By default, `django-cas-server` do not allow any service to use the CAS to authenticate users. In order to allow services, you need to connect to the django admin interface using a django superuser, and add a first service pattern.

A service pattern comes with 9 fields:

- `Position`: an integer used to change the order in which services are matched against service patterns.
- `Name`: the name of the service pattern. It will be displayed to the users asking for a ticket for a service matching this service pattern on the login page.
- `Pattern`: a regular expression used to match services.
- `User field`: the user attribute to use as username for services matching this service pattern. Leave it empty to use the login name.
- `Restrict username`: if checked, only login name defined below are allowed to get tickets for services matching this service pattern.
- `Proxy`: if checked, allow the creation of Proxy Ticket for services matching this service pattern. Otherwise, only Service Ticket will be created.
- `Proxy callback`: if checked, services matching this service pattern are allowed to retrieve Proxy Granting Ticket. A service with a Proxy Granting Ticket can get Proxy Ticket for other services. Hence you must only check this for trusted services that need it. (For instance, a webmail needs Proxy Ticket to authenticate himself as the user to the imap server).
- `Single log out`: Check it to send Single Log Out requests to authenticated services matching this service pattern. SLO requests are send to all services the user is authenticated to then the user disconnect.
- `Single log out callback`: The http(s) URL to POST the SLO requests. If empty, the service URL is used. This field is useful to allow non http services (imap, smtp, ftp) to handle SLO requests.

A service pattern has 4 associated models:

- `Usernames`: a list of username associated with the `Restrict username` field
- `Replace attribut names`: a list of user attributes to send to the service. Choose the name used for sending the attribute by setting `Remplacement` or leave it empty to leave it unchanged.
- `Replace attribut values`: a list of sent user attributes for which value needs to be tweak. Replace the attribute value by the string obtained by replacing the leftmost non-overlapping occurrences of `pattern` in string by `replace`. In `replace` backslash escapes are processed. Matched groups are captures by 1, 2, etc.
- `Filter attribut values`: a list of user attributes for which value needs to match a regular expression. For instance, service A may need an email address, and you only want user with an email address to connect to it. To do so, put `email` in `Attribute` and `.*` in `pattern`.

Then a user ask a ticket for a service, the service URL is compare against each service patterns sorted by *position*. The first service pattern that matches the service URL is chosen. Hence, you should give low *position* to very specific patterns like `^https://www\.example\.com(/.*)?$` and higher *position* to generic patterns like `^https://.*`. So the service URL *https://www.examle.com* will use the service pattern for `^https://www\.example\.com(/.*)?$` and not the one for `^https://.*`.

1.9 Federation mode

`django-cas-server` comes with a federation mode. Then `CAS_FEDERATE` is `True`, user are invited to choose an identity provider on the login page, then, they are redirected to the provider CAS to authenticate. This provider transmit to `django-cas-server` the user username and attributes. The user is now logged in on `django-cas-server` and can use services using `django-cas-server` as CAS.

The list of allowed identity providers is defined using the django admin application. With the development server started, visit http://127.0.0.1:8000/admin/ to add identity providers.

An identity provider comes with 5 fields:

- `Position`: an integer used to tweak the order in which identity providers are displayed on the login page. Identity providers are sorted using position first, then, on equal position, using `verbose name` and then, on equal `verbose name`, using `suffix`.
- `Suffix`: the suffix that will be append to the username returned by the identity provider. It must be unique.
- `Server url`: the URL to the identity provider CAS. For instance, if you are using `https://cas.example.org/login` to authenticate on the CAS, the *server url* is `https://cas.example.org`
- `CAS protocol version`: the version of the CAS protocol to use to contact the identity provider. The default is version 3.
- `Verbose name`: the name used on the login page to display the identity provider.
- `Display`: a boolean controlling the display of the identity provider on the login page. Beware that this do not disable the identity provider, it just hide it on the login page. User will always be able to log in using this provider by fetching `/federate/provider_suffix`.

In federation mode, `django-cas-server` build user's username as follow: `provider_returned_username@provider_suffix`. Choose the provider returned username for `django-cas-server` and the provider suffix in order to make sense, as this built username is likely to be displayed to end users in applications.

Then using federate mode, you should add one command to a daily crontab: `cas_clean_federate`. This command clean the local cache of federated user from old unused users.

You could for example do as bellow :

CHAPTER 2

cas_server package

2.1 Subpackages

2.1.1 cas_server.templatetags package

Submodules

cas_server.templatetags.cas_server module

template tags for the app

cas_server.templatetags.cas_server.**is_checkbox**(*field*)

check if a form bound field is a checkbox

Parameters **field** (*django.forms.BoundField*) – A bound field

Returns `True` if the field is a checkbox, `False` otherwise.

Return type bool

cas_server.templatetags.cas_server.**is_hidden**(*field*)

check if a form bound field is hidden

Parameters **field** (*django.forms.BoundField*) – A bound field

Returns `True` if the field is hidden, `False` otherwise.

Return type bool

Module contents

2.2 Submodules

2.2.1 cas_server.admin module

module for the admin interface of the app

class cas_server.admin.**BaseInlines**(*parent_model*, *admin_site*)
Bases: django.contrib.admin.TabularInline

Base class for inlines in the admin interface.

extra = 0
This controls the number of extra forms the formset will display in addition to the initial forms.

media

class cas_server.admin.**UserAdminInlines**(*parent_model*, *admin_site*)
Bases: *BaseInlines*

Base class for inlines in *UserAdmin* interface

form
The form *TicketForm* used to display tickets.

alias of TicketForm

readonly_fields = ('validate', 'service', 'service_pattern', 'creation', 'renew', 'single_log_out', 'value')
Fields to display on a object that are read only (not editable).

fields = ('validate', 'service', 'service_pattern', 'creation', 'renew', 'single_log_out')
Fields to display on a object.

media

class cas_server.admin.**ServiceTicketInline**(*parent_model*, *admin_site*)
Bases: *UserAdminInlines*

ServiceTicket in admin interface

model
The model which the inline is using.

alias of ServiceTicket

media

class cas_server.admin.**ProxyTicketInline**(*parent_model*, *admin_site*)
Bases: *UserAdminInlines*

ProxyTicket in admin interface

model
The model which the inline is using.

alias of ProxyTicket

media

class cas_server.admin.**ProxyGrantingInline**(*parent_model*, *admin_site*)
Bases: *UserAdminInlines*

ProxyGrantingTicket in admin interface

model
The model which the inline is using.

alias of ProxyGrantingTicket

media

class cas_server.admin.**UserAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.ModelAdmin

User in admin interface

inlines = (<class 'cas_server.admin.ServiceTicketInline'>, <class 'cas_server.admin.ProxyTicketInline'>, <class 'cas_se
See *ServiceTicketInline*, *ProxyTicketInline*, *ProxyGrantingInline* objects below the *UserAdmin* fields.

readonly_fields = ('username', 'date', 'session_key')
Fields to display on a object that are read only (not editable).

fields = ('username', 'date', 'session_key')
Fields to display on a object.

list_display = ('username', 'date', 'session_key')
Fields to display on the list of class:*UserAdmin* objects.

media

class cas_server.admin.**UsernamesInline**(*parent_model*, *admin_site*)
Bases: *BaseInlines*

Username in admin interface

model
The model which the inline is using.

alias of Username

media

class cas_server.admin.**ReplaceAttributNameInline**(*parent_model*, *admin_site*)
Bases: *BaseInlines*

ReplaceAttributName in admin interface

model
The model which the inline is using.

alias of ReplaceAttributName

media

class cas_server.admin.**ReplaceAttributValueInline**(*parent_model*, *admin_site*)
Bases: *BaseInlines*

ReplaceAttributValue in admin interface

model
The model which the inline is using.

alias of ReplaceAttributValue

media

class cas_server.admin.**FilterAttributValueInline**(*parent_model*, *admin_site*)
Bases: *BaseInlines*

FilterAttributValue in admin interface

model
The model which the inline is using.

alias of FilterAttributValue

media

class cas_server.admin.**ServicePatternAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.ModelAdmin

ServicePattern in admin interface

inlines = (<class 'cas_server.admin.UsernamesInline'>, <class 'cas_server.admin.ReplaceAttributNameInline'>, <class
See *UsernamesInline*, *ReplaceAttributNameInline*, *ReplaceAttributValueInline*, *FilterAttributValueInline* objects below the *ServicePatternAdmin* fields.

list_display = ('pos', 'name', 'pattern', 'proxy', 'single_log_out', 'proxy_callback', 'restrict_users')
Fields to display on the list of class:*ServicePatternAdmin* objects.

media

class cas_server.admin.**FederatedIendityProviderAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.ModelAdmin

FederatedIendityProvider in admin interface

fields = ('pos', 'suffix', 'server_url', 'cas_protocol_version', 'verbose_name', 'display')
Fields to display on a object.

list_display = ('verbose_name', 'suffix', 'display')
Fields to display on the list of class:*FederatedIendityProviderAdmin* objects.

media

2.2.2 cas_server.apps module

django config module

class cas_server.apps.**CasAppConfig**(*app_name*, *app_module*)
Bases: django.apps.AppConfig

django CAS application config class

name = 'cas_server'
Full Python path to the application. It must be unique across a Django project.

verbose_name = <django.utils.functional.__proxy__ object>
Human-readable name for the application.

2.2.3 cas_server.auth module

Some authentication classes for the CAS

class cas_server.auth.**AuthUser**(*username*)
Bases: object

Authentication base class

Parameters **username** (*unicode*) – A username, stored in the *username* class attribute.

username = None
username used to instanciate the current object

test_password(*password*)
Tests password agains the user password.

Raises **NotImplementedError** – always. The method need to be implemented by sub-classes

attributs()
The user attributes.

raises NotImplementedError: always. The method need to be implemented by subclasses

class cas_server.auth.**DummyAuthUser**(*username*)
Bases: *cas_server.auth.AuthUser*

A Dummy authentication class. Authentication always fails

Parameters **username** (*unicode*) – A username, stored in the *username* class attribute. There is no valid value for this attribute here.

test_password(*password*)
Tests password agains the user password.

Parameters **password** (*unicode*) – a clear text password as submited by the user.

Returns always False

Return type bool

attributs()
The user attributes.

Returns en empty dict.

Return type dict

class cas_server.auth.**TestAuthUser**(*username*)
Bases: *cas_server.auth.AuthUser*

A test authentication class only working for one unique user.

Parameters **username** (*unicode*) – A username, stored in the *username* class attribute. The uniq valid value is settings.CAS_TEST_USER.

test_password(*password*)
Tests password agains the user password.

Parameters **password** (*unicode*) – a clear text password as submited by the user.

Returns True if *username* is valid and password is equal to settings.CAS_TEST_PASSWORD, False otherwise.

Return type bool

attributs()
The user attributes.

Returns the settings.CAS_TEST_ATTRIBUTES dict if *username* is valid, an empty dict otherwise.

Return type dict

class cas_server.auth.**DBAuthUser**(*username*)
Bases: *cas_server.auth.AuthUser*

base class for databate based auth classes

user = None
DB user attributes as a dict if the username is found in the database.

attributs()
The user attributes.

Returns a `dict` with the user attributes. Attributes may be `unicode()` or `list` of `unicode()`. If the user do not exists, the returned `dict` is empty.

Return type dict

class `cas_server.auth.`**`MysqlAuthUser`**(*username*)
Bases: *cas_server.auth.DBAuthUser*

DEPRECATED, use *SqlAuthUser* instead.

A mysql authentication class: authenticate user agains a mysql database

Parameters **`username`** (*unicode*) – A username, stored in the *username* class attribute. Valid value are fetched from the MySQL database set with `settings.CAS_SQL_*` settings parameters using the query `settings.CAS_SQL_USER_QUERY`.

`test_password`(*password*)
Tests `password` agains the user password.

Parameters **`password`** (*unicode*) – a clear text password as submited by the user.

Returns `True` if *username* is valid and `password` is correct, `False` otherwise.

Return type bool

class `cas_server.auth.`**`SqlAuthUser`**(*username*)
Bases: *cas_server.auth.DBAuthUser*

A SQL authentication class: authenticate user agains a SQL database. The SQL database must be configures in settings.py as `settings.DATABASES['cas_server']`.

Parameters **`username`** (*unicode*) – A username, stored in the *username* class attribute. Valid value are fetched from the MySQL database set with `settings.CAS_SQL_*` settings parameters using the query `settings.CAS_SQL_USER_QUERY`.

`test_password`(*password*)
Tests `password` agains the user password.

Parameters **`password`** (*unicode*) – a clear text password as submited by the user.

Returns `True` if *username* is valid and `password` is correct, `False` otherwise.

Return type bool

class `cas_server.auth.`**`LdapAuthUser`**(*username*)
Bases: *cas_server.auth.DBAuthUser*

A ldap authentication class: authenticate user against a ldap database

Parameters **`username`** (*unicode*) – A username, stored in the *username* class attribute. Valid value are fetched from the ldap database set with `settings.CAS_LDAP_*` settings parameters.

classmethod **`get_conn`**()
Return a connection object to the ldap database

`test_password`(*password*)
Tests `password` agains the user password.

Parameters **`password`** (*unicode*) – a clear text password as submited by the user.

Returns `True` if *username* is valid and `password` is correct, `False` otherwise.

Return type bool

class cas_server.auth.**DjangoAuthUser**(*username*)
Bases: *cas_server.auth.AuthUser*

A django auth class: authenticate user against django internal users

Parameters **username** (*unicode*) – A username, stored in the *username* class attribute. Valid value are usernames of django internal users.

user = None
a django user object if the username is found. The user model is retreived using `django.contrib.auth.get_user_model()`.

test_password(*password*)
Tests `password` agains the user password.

Parameters **password** (*unicode*) – a clear text password as submited by the user.

Returns `True` if *user* is valid and `password` is correct, `False` otherwise.

Return type bool

attributs()
The user attributes, defined as the fields on the *user* object.

Returns a `dict` with the *user* object fields. Attributes may be If the user do not exists, the returned `dict` is empty.

Return type dict

class cas_server.auth.**CASFederateAuth**(*username*)
Bases: *cas_server.auth.AuthUser*

Authentication class used then CAS_FEDERATE is True

Parameters **username** (*unicode*) – A username, stored in the *username* class attribute. Valid value are usernames of *FederatedUser* object. *FederatedUser* object are created on CAS backends successful ticket validation.

user = None
a :class‘FederatedUser<cas_server.models.FederatedUser>‘ object if `username` is found.

test_password(*ticket*)
Tests `password` agains the user password.

Parameters **password** (*unicode*) – The CAS tickets just used to validate the user authentication against its CAS backend.

Returns `True` if *user* is valid and `password` is a ticket validated less than `settings.CAS_TICKET_VALIDITY` secondes and has not being previously used for authenticated this *FederatedUser*. `False` otherwise.

Return type bool

attributs()
The user attributes, as returned by the CAS backend.

Returns `FederatedUser.attributs`. If the user do not exists, the returned `dict` is empty.

Return type dict

2.2.4 cas_server.cas module

exception cas_server.cas.**CASError**
Bases: exceptions.ValueError

class cas_server.cas.**ReturnUnicode**
Bases: object

static u(*string*, *charset*)

class cas_server.cas.**SingleLogoutMixin**
Bases: object

classmethod get_saml_slos(*logout_request*)
returns saml logout ticket info

class cas_server.cas.**CASClient**
Bases: object

class cas_server.cas.**CASClientBase**(*service_url=None*, *server_url=None*, *extra_login_params=None*, *renew=False*, *username_attribute=None*)
Bases: object

logout_redirect_param_name = 'service'

verify_ticket(*ticket*)
must return a triple

get_login_url()
Generates CAS login URL

get_logout_url(*redirect_url=None*)
Generates CAS logout URL

get_proxy_url(*pgt*)
Returns proxy url, given the proxy granting ticket

get_proxy_ticket(*pgt*)
Returns proxy ticket given the proxy granting ticket

static get_page_charset(*page*, *default='utf-8'*)

class cas_server.cas.**CASClientV1**(*service_url=None*, *server_url=None*, *extra_login_params=None*, *renew=False*, *username_attribute=None*)
Bases: *cas_server.cas.CASClientBase*, *cas_server.cas.ReturnUnicode*

CAS Client Version 1

logout_redirect_param_name = 'url'

verify_ticket(*ticket*)
Verifies CAS 1.0 authentication ticket.

Returns username on success and None on failure.

class cas_server.cas.**CASClientV2**(*proxy_callback=None*, **args*, ***kwargs*)
Bases: *cas_server.cas.CASClientBase*, *cas_server.cas.ReturnUnicode*

CAS Client Version 2

url_suffix = 'serviceValidate'

logout_redirect_param_name = 'url'

verify_ticket(*ticket*)
Verifies CAS 2.0+/3.0+ XML-based authentication ticket and returns extended attributes

get_verification_response(*ticket*)

classmethod parse_attributes_xml_element(*element*, *charset*)

classmethod verify_response(*response*, *charset*)

classmethod parse_response_xml(*response*, *charset*)

class cas_server.cas.**CASClientV3**(*proxy_callback=None*, **args*, ***kwargs*)
Bases: *cas_server.cas.CASClientV2*, *cas_server.cas.SingleLogoutMixin*

CAS Client Version 3

url_suffix = 'serviceValidate'

logout_redirect_param_name = 'service'

classmethod parse_attributes_xml_element(*element*, *charset*)

classmethod verify_response(*response*, *charset*)

class cas_server.cas.**CASClientWithSAMLV1**(*proxy_callback=None*, **args*, ***kwargs*)
Bases: *cas_server.cas.CASClientV2*, *cas_server.cas.SingleLogoutMixin*

CASClient 3.0+ with SAML

verify_ticket(*ticket*, ***kwargs*)
Verifies CAS 3.0+ XML-based authentication ticket and returns extended attributes.

@date: 2011-11-30 @author: Carlos Gonzalez Vila <carlewis@gmail.com>

Returns username and attributes on success and None,None on failure.

fetch_saml_validation(*ticket*)

classmethod get_saml_assertion(*ticket*)
http://www.jasig.org/cas/protocol#samlvalidate-cas-3.0

SAML request values:

RequestID [REQUIRED]: unique identifier for the request

IssueInstant [REQUIRED]: timestamp of the request

samlp:AssertionArtifact [REQUIRED]: the valid CAS Service Ticket obtained as a response parameter at login.

2.2.5 cas_server.default_settings module

Default values for the app's settings

cas_server.default_settings.**CAS_LOGO_URL = '/static/cas_server/logo.png'**
URL to the logo showed in the up left corner on the default templates.

cas_server.default_settings.**CAS_FAVICON_URL = '/static/cas_server/favicon.ico'**
URL to the favicon (shortcut icon) used by the default templates. Default is a key icon.

cas_server.default_settings.**CAS_SHOW_POWERED = True**
Show the powered by footer if set to `True`

cas_server.default_settings.**CAS_COMPONENT_URLS = {'bootstrap3_js': '//maxcdn.bootstrapcdn.com/bootstrap/3.3**
URLs to css and javascript external components.

cas_server.default_settings.**CAS_LOGIN_TEMPLATE = 'cas_server/login.html'**
Path to the template showed on /login then the user is not autenticated.

cas_server.default_settings.**CAS_WARN_TEMPLATE = 'cas_server/warn.html'**
Path to the template showed on /login?service=... then the user is authenticated and has asked to be warned before being connected to a service.

cas_server.default_settings.**CAS_LOGGED_TEMPLATE = 'cas_server/logged.html'**
Path to the template showed on /login then to user is authenticated.

cas_server.default_settings.**CAS_LOGOUT_TEMPLATE = 'cas_server/logout.html'**
Path to the template showed on /logout then to user is being disconnected.

cas_server.default_settings.**CAS_REDIRECT_TO_LOGIN_AFTER_LOGOUT = False**
Should we redirect users to /login after they logged out instead of displaying *CAS_LOGOUT_TEMPLATE*.

cas_server.default_settings.**CAS_AUTH_CLASS = 'cas_server.auth.DjangoAuthUser'**
A dotted path to a class or a class implementing cas_server.auth.AuthUser.

cas_server.default_settings.**CAS_PROXY_CA_CERTIFICATE_PATH = True**
Path to certificate authorities file. Usually on linux the local CAs are in /etc/ssl/certs/ca-certificates.crt. `True` tell requests to use its internal certificat authorities.

cas_server.default_settings.**CAS_SLO_MAX_PARALLEL_REQUESTS = 10**
Maximum number of parallel single log out requests send if more requests need to be send, there are queued

cas_server.default_settings.**CAS_SLO_TIMEOUT = 5**
Timeout for a single SLO request in seconds.

cas_server.default_settings.**CAS_AUTH_SHARED_SECRET = ''**
Shared to transmit then using the view *cas_server.views.Auth*

cas_server.default_settings.**CAS_TICKET_VALIDITY = 60**
Number of seconds the service tickets and proxy tickets are valid. This is the maximal time between ticket issuance by the CAS and ticket validation by an application.

cas_server.default_settings.**CAS_PGT_VALIDITY = 3600**
Number of seconds the proxy granting tickets are valid.

cas_server.default_settings.**CAS_TICKET_TIMEOUT = 86400**
Number of seconds a ticket is kept in the database before sending Single Log Out request and being cleared.

cas_server.default_settings.**CAS_TICKET_LEN = 64**
All CAS implementation MUST support ST and PT up to 32 chars, PGT and PGTIOU up to 64 chars and it is RECOMMENDED that all tickets up to 256 chars are supports so we use 64 for the default len.

cas_server.default_settings.**CAS_LT_LEN = 64**
alias of `settings.CAS_TICKET_LEN`

cas_server.default_settings.**CAS_ST_LEN = 64**
alias of `settings.CAS_TICKET_LEN` Services MUST be able to accept service tickets of up to 32 characters in length.

cas_server.default_settings.**CAS_PT_LEN = 64**
alias of `settings.CAS_TICKET_LEN` Back-end services MUST be able to accept proxy tickets of up to 32 characters.

cas_server.default_settings.**CAS_PGT_LEN = 64**
alias of `settings.CAS_TICKET_LEN` Services MUST be able to handle proxy-granting tickets of up to 64

cas_server.default_settings.**CAS_PGTIOU_LEN = 64**
alias of `settings.CAS_TICKET_LEN` Services MUST be able to handle PGTIOUs of up to 64 characters in length.

cas_server.default_settings.**CAS_LOGIN_TICKET_PREFIX = u'LT'**
Prefix of login tickets.

cas_server.default_settings.**CAS_SERVICE_TICKET_PREFIX = u'ST'**
Prefix of service tickets. Service tickets MUST begin with the characters ST so you should not change this.

cas_server.default_settings.**CAS_PROXY_TICKET_PREFIX = u'PT'**
Prefix of proxy ticket. Proxy tickets SHOULD begin with the characters, PT.

cas_server.default_settings.**CAS_PROXY_GRANTING_TICKET_PREFIX = u'PGT'**
Prefix of proxy granting ticket. Proxy-granting tickets SHOULD begin with the characters PGT.

cas_server.default_settings.**CAS_PROXY_GRANTING_TICKET_IOU_PREFIX = u'PGTIOU'**
Prefix of proxy granting ticket IOU. Proxy-granting ticket IOUs SHOULD begin with the characters PGTIOU.

cas_server.default_settings.**CAS_SQL_HOST = 'localhost'**
Host for the SQL server.

cas_server.default_settings.**CAS_SQL_USERNAME = ''**
Username for connecting to the SQL server.

cas_server.default_settings.**CAS_SQL_PASSWORD = ''**
Password for connecting to the SQL server.

cas_server.default_settings.**CAS_SQL_DBNAME = ''**
Database name.

cas_server.default_settings.**CAS_SQL_DBCHARSET = 'utf8'**
Database charset.

cas_server.default_settings.**CAS_SQL_USER_QUERY = 'SELECT user AS username, pass AS password, users.* FR**
The query performed upon user authentication.

cas_server.default_settings.**CAS_SQL_PASSWORD_CHECK = 'crypt'**
The method used to check the user password. Must be one of `"crypt"`, `"ldap"`, `"hex_md5"`, `"hex_sha1"`, `"hex_sha224"`, `"hex_sha256"`, `"hex_sha384"`, `"hex_sha512"`, `"plain"`.

cas_server.default_settings.**CAS_SQL_PASSWORD_CHARSET = 'utf-8'**
charset the SQL users passwords was hash with

cas_server.default_settings.**CAS_LDAP_SERVER = 'localhost'**
Address of the LDAP server

cas_server.default_settings.**CAS_LDAP_USER = None**
LDAP user bind address, for example `"cn=admin,dc=crans,dc=org"` for connecting to the LDAP server.

cas_server.default_settings.**CAS_LDAP_PASSWORD = None**
LDAP connection password

cas_server.default_settings.**CAS_LDAP_BASE_DN = None**
LDAP seach base DN, for example `"ou=data,dc=crans,dc=org"`.

cas_server.default_settings.**CAS_LDAP_USER_QUERY = '(uid=%s)'**
LDAP search filter for searching user by username. User inputed usernames are escaped using `ldap3.utils.conv.escape_bytes()`.

cas_server.default_settings.**CAS_LDAP_USERNAME_ATTR = 'uid'**
LDAP attribute used for users usernames

cas_server.default_settings.**CAS_LDAP_PASSWORD_ATTR = 'userPassword'**
LDAP attribute used for users passwords

cas_server.default_settings.**CAS_LDAP_PASSWORD_CHECK = 'ldap'**
The method used to check the user password. Must be one of `"crypt"`, `"ldap"`, `"hex_md5"`, `"hex_sha1"`, `"hex_sha224"`, `"hex_sha256"`, `"hex_sha384"`, `"hex_sha512"`, `"plain"`.

cas_server.default_settings.**CAS_LDAP_PASSWORD_CHARSET = 'utf-8'**
charset the LDAP users passwords was hash with

cas_server.default_settings.**CAS_TEST_USER = 'test'**
Username of the test user.

cas_server.default_settings.**CAS_TEST_PASSWORD = 'test'**
Password of the test user.

cas_server.default_settings.**CAS_TEST_ATTRIBUTES = {'nom': 'Nymous', 'alias': ['demo1', 'demo2'], 'prenom': '**
Attributes of the test user.

cas_server.default_settings.**CAS_ENABLE_AJAX_AUTH = False**
A `bool` for activatinc the hability to fetch tickets using javascript.

cas_server.default_settings.**CAS_FEDERATE = False**
A `bool` for activating the federated mode

cas_server.default_settings.**CAS_FEDERATE_REMEMBER_TIMEOUT = 604800**
Time after witch the cookie use for "remember my identity provider" expire (one week).

cas_server.default_settings.**CAS_NEW_VERSION_HTML_WARNING = True**
A `bool` for diplaying a warning on html pages then a new version of the application is avaible. Once closed by a user, it is not displayed to this user until the next new version.

cas_server.default_settings.**CAS_NEW_VERSION_EMAIL_WARNING = True**
A `bool` for sending emails to `settings.ADMINS` when a new version is available.

cas_server.default_settings.**CAS_NEW_VERSION_JSON_URL = 'https://pypi.python.org/pypi/django-cas-server/jso**
URL to the pypi json of the application. Used to retreive the version number of the last version. You should not change it.

class cas_server.default_settings.**SessionStore**(*session_key=None*)
Bases: `django.contrib.sessions.backends.base.SessionBase`

SessionStore class depending of `SESSION_ENGINE`

classmethod clear_expired()

create()

create_model_instance(*data*)
Return a new instance of the session model object, which represents the current session state. Intended to be used for saving the session data to the database.

delete(*session_key=None*)

exists(*session_key*)

classmethod get_model_class()

load()

model

save(*must_create=False*)
Saves the current session data to the database. If 'must_create' is True, a database error will be raised if the saving operation doesn't create a *new* entry (as opposed to possibly updating an existing entry).

2.2.6 cas_server.federate module

federated mode helper classes

cas_server.federate.**logger = <logging.Logger object>**
logger facility

class cas_server.federate.**CASFederateValidateUser**(*provider*, *service_url*, *renew=False*)
Bases: object

Class CAS client used to authenticate the user again a CAS provider

Parameters

- **provider** (cas_server.models.FederatedIendityProvider) – The provider to use for authenticate the user.
- **service_url** (*unicode*) – The service url to transmit to the provider.

username = None
the provider returned username

attributs = {}
the provider returned attributes

federated_username = None
the provider returned username this the provider suffix appended

provider = None
the identity provider

client = None
the CAS client instance

get_login_url()

Returns the CAS provider login url

Return type unicode

get_logout_url(*redirect_url=None*)

Parameters **redirect_url** (unicode or NoneType) – The url to redirect to after logout from the provider, if provided.

Returns the CAS provider logout url

Return type unicode

verify_ticket(*ticket*)
test ticket agains the CAS provider, if valid, create a *FederatedUser* matching provider returned username and attributes.

Parameters **ticket** (*unicode*) – The ticket to validate against the provider CAS

Returns True if the validation succeed, else False.

Return type bool

static register_slo(*username*, *session_key*, *ticket*)
association a ticket with a (username, session_key) for processing later SLO request by creating a *cas_server.models.FederateSLO* object.

Parameters

- **username** (*unicode*) – A logged user username, with the @ component.

- **session_key** (*unicode*) – A logged user session_key matching `username`.
- **ticket** (*unicode*) – A ticket used to authentication `username` for the session `session_key`.

clean_sessions(*logout_request*)
process a SLO request: Search for ticket values in `logout_request`. For each ticket value matching a *cas_server.models.FederateSLO*, disconnect the corresponding user.

Parameters **logout_request** (*unicode*) – An XML document contening one or more Single Log Out requests.

2.2.7 cas_server.forms module

forms for the app

class cas_server.forms.**BootsrapForm**(**args*, ***kwargs*)
Bases: `django.forms.Form`

Form base class to use boostrap then rendering the form fields

class cas_server.forms.**BaseLogin**(**args*, ***kwargs*)
Bases: *BootsrapForm*

Base form with all field possibly hidden on the login pages

service = None
The service url for which the user want a ticket

lt = None
A valid LoginTicket to prevent POST replay

renew = None
Is the service asking the authentication renewal ?

gateway = None
Url to redirect to if the authentication fail (user not authenticated or bad service)

class cas_server.forms.**WarnForm**(**args*, ***kwargs*)
Bases: *BaseLogin*

Form used on warn page before emiting a ticket

warned = None
`True` if the user has been warned of the ticket emission

class cas_server.forms.**FederateSelect**(**args*, ***kwargs*)
Bases: *BaseLogin*

Form used on the login page when `settings.CAS_FEDERATE` is `True` allowing the user to choose an identity provider.

provider = None
The providers the user can choose to be used as authentication backend

warn = None
A checkbox to ask to be warn before emiting a ticket for another service

remember = None
A checkbox to remember the user choices of *provider*

class cas_server.forms.**UserCredential**(**args*, ***kwargs*)
Bases: *BaseLogin*

Form used on the login page to retrive user credentials

username = None
The user username

password = None
The user password

warn = None
A checkbox to ask to be warn before emiting a ticket for another service

clean()
Validate that the submited *username* and *password* are valid

Raises **django.forms.ValidationError** – if the *username* and *password* are not valid.

Returns The cleaned POST data

Return type dict

class cas_server.forms.**FederateUserCredential**(**args*, ***kwargs*)
Bases: *UserCredential*

Form used on a auto submited page for linking the views *FederateAuth* and *LoginView*.

On successful authentication on a provider, in the view *FederateAuth* a *FederatedUser* is created by *cas_server.federate.CASFederateValidateUser.verify_ticket()* and the user is redirected to *LoginView*. This form is then automatically filled with infos matching the created *FederatedUser* using the `ticket` as one time password and submited using javascript. If javascript is not enabled, a connect button is displayed.

This stub authentication form, allow to implement the federated mode with very few modificatons to the *LoginView* view.

clean()
Validate that the submited `username` and `password` are valid using the *CASFederateAuth* auth class.

Raises **django.forms.ValidationError** – if the `username` and `password` do not correspond to a *FederatedUser*.

Returns The cleaned POST data

Return type dict

class cas_server.forms.**TicketForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*)
Bases: `django.forms.ModelForm`

Form for Tickets in the admin interface

2.2.8 cas_server.models module

models for the app

cas_server.models.**logger = <logging.Logger object>**
logger facility

class cas_server.models.**JsonAttributes**(**args*, ***kwargs*)
Bases: django.db.models.Model

A base class for models storing attributes as a json

class Meta

abstract = False

JsonAttributes.**attributs**
The attributes

class cas_server.models.**FederatedIendityProvider**(**args*, ***kwargs*)
Bases: django.db.models.Model

An identity provider for the federated mode

suffix = None
Suffix append to backend CAS returned username: `returned_username @ suffix`. it must be unique.

server_url = None
URL to the root of the CAS server application. If login page is https://cas.example.net/cas/login then *server_url* should be https://cas.example.net/cas/

cas_protocol_version = None
Version of the CAS protocol to use when sending requests the the backend CAS.

verbose_name = None
Name for this identity provider displayed on the login page.

pos = None
Position of the identity provider on the login page. Identity provider are sorted using the (*pos*, *verbose_name*, *suffix*) attributes.

display = None
Display the provider on the login page. Beware that this do not disable the identity provider, it just hide it on the login page. User will always be able to log in using this provider by fetching `/federate/suffix`.

static build_username_from_suffix(*username*, *suffix*)
Transform backend username into federated username using `suffix`

Parameters

- **username** (*unicode*) – A CAS backend returned username
- **suffix** (*unicode*) – A suffix identifying the CAS backend

Returns The federated username: `username @ suffix`.

Return type unicode

build_username(*username*)
Transform backend username into federated username

Parameters **username** (*unicode*) – A CAS backend returned username

Returns The federated username: `username @ suffix`.

Return type unicode

exception DoesNotExist

exception FederatedIendityProvider.**MultipleObjectsReturned**

FederatedIendityProvider.**federateduser_set**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

FederatedIendityProvider.**get_cas_protocol_version_display**(**moreargs*, ***morekwargs*)

FederatedIendityProvider.**objects = <django.db.models.manager.Manager object>**

class cas_server.models.**FederatedUser**(**args*, ***kwargs*)
Bases: *JsonAttributes*

A federated user as returner by a CAS provider (username and attributes)

username = None
The user username returned by the CAS backend on successful ticket validation

provider
A foreign key to *FederatedIendityProvider*

ticket = None
The last ticket used to authenticate *username* against *provider*

last_update = None
Last update timespampt. Usually, the last time *ticket* has been set.

federated_username
The federated username with a suffix for the current *FederatedUser*.

classmethod get_from_federated_username(*username*)

Returns A *FederatedUser* object from a federated `username`

Return type *FederatedUser*

classmethod clean_old_entries()
remove old unused *FederatedUser*

exception DoesNotExist

exception FederatedUser.**MultipleObjectsReturned**

FederatedUser.**get_next_by_last_update**(**moreargs*, ***morekwargs*)

FederatedUser.**get_previous_by_last_update**(**moreargs*, ***morekwargs*)

FederatedUser.**objects = <django.db.models.manager.Manager object>**

class cas_server.models.**FederateSLO**(**args*, ***kwargs*)
Bases: django.db.models.Model

An association between a CAS provider ticket and a (username, session) for processing SLO

username = None
the federated username with the ‘‘@‘‘component

session_key = None
the session key for the session *username* has been authenticated using *ticket*

ticket = None
The ticket used to authenticate *username*

classmethod clean_deleted_sessions()
remove old *FederateSLO* object for which the session do not exists anymore

exception DoesNotExist

exception FederateSLO.MultipleObjectsReturned

FederateSLO.objects = <django.db.models.manager.Manager object>

class cas_server.models.**User**(**args*, ***kwargs*)
Bases: django.db.models.Model

A user logged into the CAS

session_key = None
The session key of the current authenticated user

username = None
The username of the current authenticated user

date = None
Last time the authenticated user has do something (auth, fetch ticket, etc...)

delete(**args*, ***kwargs*)
Remove the current *User*. If settings.CAS_FEDERATE is True, also delete the corresponding *FederateSLO* object.

classmethod clean_old_entries()
Remove *User* objects inactive since more that SESSION_COOKIE_AGE and send corresponding SingleLogOut requests.

classmethod clean_deleted_sessions()
Remove *User* objects where the corresponding session do not exists anymore.

attributs
Property. A fresh dict for the user attributes, using settings.CAS_AUTH_CLASS

logout(*request=None*)
Send SLO requests to all services the user is logged in.

Parameters **request** (django.http.HttpRequest or NoneType) – The current django HttpRequest to display possible failure to the user.

get_ticket(*ticket_class*, *service*, *service_pattern*, *renew*)
Generate a ticket using ticket_class for the service service matching service_pattern and asking or not for authentication renewal with renew

Parameters

- **ticket_class** (*type*) – *ServiceTicket* or *ProxyTicket* or *ProxyGrantingTicket*.
- **service** (*unicode*) – The service url for which we want a ticket.
- **service_pattern** (ServicePattern) – The service pattern matching service. Beware that service must match *ServicePattern.pattern* and the current *User* must pass *ServicePattern.check_user()*. These checks are not done here and you must perform them before calling this method.
- **renew** (*bool*) – Should be True if authentication has been renewed. Must be False otherwise.

Returns A *Ticket* object.

Return type *ServiceTicket* or *ProxyTicket* or *ProxyGrantingTicket*.

get_service_url(*service*, *service_pattern*, *renew*)
Return the url to which the user must be redirected to after a Service Ticket has been generated

Parameters

- **service** (*unicode*) – The service url for which we want a ticket.
- **service_pattern** (ServicePattern) – The service pattern matching `service`. Beware that `service` must match *ServicePattern.pattern* and the current *User* must pass *ServicePattern.check_user()*. These checks are not done here and you must perform them before calling this method.
- **renew** (*bool*) – Should be `True` if authentication has been renewed. Must be `False` otherwise.

Return unicode The service url with the ticket GET param added.

Return type unicode

exception DoesNotExist

exception User.**MultipleObjectsReturned**

User.**get_next_by_date**(**moreargs*, ***morekwargs*)

User.**get_previous_by_date**(**moreargs*, ***morekwargs*)

User.**objects = <django.db.models.manager.Manager object>**

User.**proxygrantingticket**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

User.**proxyticket**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

User.**serviceticket**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

exception cas_server.models.**ServicePatternException**
Bases: exceptions.Exception

Base exception of exceptions raised in the ServicePattern model

exception cas_server.models.**BadUsername**
Bases: *ServicePatternException*

Exception raised then an non allowed username try to get a ticket for a service

exception cas_server.models.**BadFilter**
Bases: *ServicePatternException*

Exception raised then a user try to get a ticket for a service and do not reach a condition

exception cas_server.models.**UserFieldNotDefined**
Bases: *ServicePatternException*

Exception raised then a user try to get a ticket for a service using as username an attribut not present on this user

class cas_server.models.**ServicePattern**(**args*, ***kwargs*)
Bases: django.db.models.Model

Allowed services pattern agains services are tested to

pos = None
service patterns are sorted using the *pos* attribute

name = None
A name for the service (this can bedisplayed to the user on the login page)

pattern = None
A regular expression matching services. "Will usually looks like '^https://some\.server\.com/path/.*$'. As it is a regular expression, special character must be escaped with a '\'.

user_field = None
Name of the attribute to transmit as username, if empty the user login is used

restrict_users = None
A boolean allowing to limit username allowed to connect to *usernames*.

proxy = None
A boolean allowing to deliver *ProxyTicket* to the service.

proxy_callback = None
A boolean allowing the service to be used as a proxy callback (via the pgtUrl GET param) to deliver *ProxyGrantingTicket*.

single_log_out = None
Enable SingleLogOut for the service. Old validaed tickets for the service will be kept until settings.CAS_TICKET_TIMEOUT after what a SLO request is send to the service and the ticket is purged from database. A SLO can be send earlier if the user log-out.

single_log_out_callback = None
An URL where the SLO request will be POST. If empty the service url will be used. This is usefull for non HTTP proxied services like smtp or imap.

check_user(*user*)
Check if `user` if allowed to use theses services. If `user` is not allowed, raises one of *BadFilter*, *UserFieldNotDefined*, *BadUsername*

Parameters **user** (*User*) – a *User* object

Raises

- **BadUsername** – if *restrict_users* if `True` and *User.username* is not within *usernames*.
- **BadFilter** – if a *FilterAttributValue* condition of *filters* connot be verified.
- **UserFieldNotDefined** – if *user_field* is defined and its value is not within *User.attributs*.

Returns `True`

Return type bool

classmethod validate(*service*)
Get a *ServicePattern* intance from a service url.

Parameters **service** (*unicode*) – A service url

Returns A *ServicePattern* instance matching `service`.

Return type *ServicePattern*

Raises **ServicePattern.DoesNotExist** – if no *ServicePattern* is matching `service`.

exception DoesNotExist

exception ServicePattern.**MultipleObjectsReturned**

ServicePattern.**attributs**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

ServicePattern.**filters**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

ServicePattern.**objects = <django.db.models.manager.Manager object>**

ServicePattern.**proxygrantingticket**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

ServicePattern.**proxyticket**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

ServicePattern.**replacements**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

ServicePattern.**serviceticket**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

ServicePattern.**usernames**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

class cas_server.models.**Username**(**args*, ***kwargs*)
Bases: django.db.models.Model

A list of allowed usernames on a *ServicePattern*

value = None
username allowed to connect to the service

service_pattern
ForeignKey to a *ServicePattern*. *Username* instances for a *ServicePattern* are accessible thought its *ServicePattern.usernames* attribute.

exception DoesNotExist

exception Username.**MultipleObjectsReturned**

Username.**objects = <django.db.models.manager.Manager object>**

class cas_server.models.**ReplaceAttributName**(**args*, ***kwargs*)
Bases: django.db.models.Model

A replacement of an attribute name for a *ServicePattern*. It also tell to transmit an attribute of *User.attributs* to the service. An empty *replace* mean to use the original attribute name.

name = None
Name the attribute: a key of *User.attributs*

replace = None
The name of the attribute to transmit to the service. If empty, the value of *name* is used.

service_pattern
ForeignKey to a *ServicePattern*. *ReplaceAttributName* instances for a *ServicePattern* are accessible thought its *ServicePattern.attributs* attribute.

exception DoesNotExist

exception ReplaceAttributName.**MultipleObjectsReturned**

ReplaceAttributName.**objects = <django.db.models.manager.Manager object>**

class cas_server.models.**FilterAttributValue**(**args*, ***kwargs*)
Bases: django.db.models.Model

A filter on *User.attributs* for a *ServicePattern*. If a *User* do not have an attribute *attribut* or its value do not match *pattern*, then *ServicePattern.check_user()* will raises *BadFilter* if called with that user.

attribut = None
The name of a user attribute

pattern = None
A regular expression the attribute *attribut* value must verify. If *attribut* if a list, only one of the list values needs to match.

service_pattern
ForeignKey to a *ServicePattern*. *FilterAttributValue* instances for a *ServicePattern* are accessible thought its *ServicePattern.filters* attribute.

exception DoesNotExist

exception FilterAttributValue.**MultipleObjectsReturned**

FilterAttributValue.**objects = <django.db.models.manager.Manager object>**

class cas_server.models.**ReplaceAttributValue**(**args*, ***kwargs*)
Bases: django.db.models.Model

A replacement (using a regular expression) of an attribute value for a *ServicePattern*.

attribut = None
Name the attribute: a key of *User.attributs*

pattern = None
A regular expression matching the part of the attribute value that need to be changed

replace = None
The replacement to what is mached by *pattern*. groups are capture by \1, \2 ...

service_pattern
ForeignKey to a *ServicePattern*. *ReplaceAttributValue* instances for a *ServicePattern* are accessible thought its *ServicePattern.replacements* attribute.

exception DoesNotExist

exception ReplaceAttributValue.**MultipleObjectsReturned**

ReplaceAttributValue.**objects = <django.db.models.manager.Manager object>**

class cas_server.models.**Ticket**(**args*, ***kwargs*)
Bases: *JsonAttributes*

Generic class for a Ticket

class Meta

abstract = False

Ticket.**user**
ForeignKey to a *User*.

Ticket.**validate = None**
A boolean. True if the ticket has been validated

Ticket.**service = None**
The service url for the ticket

Ticket.**service_pattern**
ForeignKey to a *ServicePattern*. The *ServicePattern* corresponding to *service*. Use *ServicePattern.validate()* to find it.

Ticket.**creation = None**
Date of the ticket creation

Ticket.**renew = None**
A boolean. True if the user has just renew his authentication

Ticket.**single_log_out = None**
A boolean. Set to *service_pattern* attribute *ServicePattern.single_log_out* value.

Ticket.**VALIDITY = 60**
Max duration between ticket creation and its validation. Any validation attempt for the ticket after *creation* + VALIDITY will fail as if the ticket do not exists.

Ticket.**TIMEOUT = 86400**
Time we keep ticket with *single_log_out* set to True before sending SingleLogOut requests.

exception Ticket.**DoesNotExist**
raised in *Ticket.get()* then ticket prefix and ticket classes mismatch

static Ticket.**send_slos**(*queryset_list*)
Send SLO requests to each ticket of each queryset of queryset_list

Parameters **queryset_list** (*list*) – A list a *Ticket* queryset

Returns A list of possibly encoutered Exception

Return type list

classmethod Ticket.**clean_old_entries**()
Remove old ticket and send SLO to timed-out services

Ticket.**logout**(*session*, *async_list=None*)
Send a SLO request to the ticket service

static Ticket.**get_class**(*ticket*, *classes=None*)
Return the ticket class of ticket

Parameters

- **ticket** (*unicode*) – A ticket
- **classes** (*list*) – Optinal arguement. A list of possible *Ticket* subclasses

Returns The class corresponding to ticket (*ServiceTicket* or *ProxyTicket* or *ProxyGrantingTicket*) if found among classes, ''None otherwise.

Return type type or NoneType

Ticket.**username**()
The username to send on ticket validation

Returns The value of the corresponding user attribute if *service_pattern*.user_field is set, the user username otherwise.

Ticket.**attributs_flat**()
generate attributes list for template rendering

Returns An list of (attribute name, attribute value) of all user attributes flatened (no nested list)

Return type list of tuple of unicode

classmethod Ticket.**get**(*ticket*, *renew=False*, *service=None*)

Search the database for a valid ticket with provided arguments

Parameters

- **ticket** (*unicode*) – A ticket value
- **renew** (*bool*) – Is authentication renewal needed
- **service** (*unicode*) – Optional argument. The ticket service

Raises

- **Ticket.DoesNotExist** – if no class is found for the ticket prefix
- **cls.DoesNotExist** – if ticket value is not found in th database

Returns a *Ticket* instance

Return type *Ticket*

Ticket.**get_next_by_creation**(**moreargs*, ***morekwargs*)

Ticket.**get_previous_by_creation**(**moreargs*, ***morekwargs*)

class cas_server.models.**ServiceTicket**(**args*, ***kwargs*)
Bases: *Ticket*

A Service Ticket

PREFIX = u'ST'
The ticket prefix used to differentiate it from other tickets types

value = None
The ticket value

exception DoesNotExist

exception ServiceTicket.**MultipleObjectsReturned**

ServiceTicket.**get_next_by_creation**(**moreargs*, ***morekwargs*)

ServiceTicket.**get_previous_by_creation**(**moreargs*, ***morekwargs*)

ServiceTicket.**objects = <django.db.models.manager.Manager object>**

ServiceTicket.**service_pattern**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

ServiceTicket.**user**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

class cas_server.models.**ProxyTicket**(**args*, ***kwargs*)
Bases: *Ticket*

A Proxy Ticket

PREFIX = u'PT'
The ticket prefix used to differentiate it from other tickets types

value = None
The ticket value

exception DoesNotExist

exception ProxyTicket.**MultipleObjectsReturned**

ProxyTicket.**get_next_by_creation**(**moreargs*, ***morekwargs*)

ProxyTicket.**get_previous_by_creation**(**moreargs*, ***morekwargs*)

ProxyTicket.**objects = <django.db.models.manager.Manager object>**

ProxyTicket.**proxies**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`ProxyTicket.`**`service_pattern`**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

`ProxyTicket.`**`user`**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

class `cas_server.models.`**`ProxyGrantingTicket`**(**args*, ***kwargs*)
Bases: *Ticket*

A Proxy Granting Ticket

`PREFIX = u'PGT'`
The ticket prefix used to differentiate it from other tickets types

`VALIDITY = 3600`
ProxyGranting ticket are never validated. However, they can be used during *VALIDITY* to get *ProxyTicket* for *user*

`value = None`
The ticket value

exception `DoesNotExist`

exception `ProxyGrantingTicket.`**`MultipleObjectsReturned`**

`ProxyGrantingTicket.`**`get_next_by_creation`**(**moreargs*, ***morekwargs*)

`ProxyGrantingTicket.`**`get_previous_by_creation`**(**moreargs*, ***morekwargs*)

`ProxyGrantingTicket.`**`objects = <django.db.models.manager.Manager object>`**

`ProxyGrantingTicket.`**`service_pattern`**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

ProxyGrantingTicket.**user**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

class cas_server.models.**Proxy**(**args*, ***kwargs*)
Bases: django.db.models.Model

A list of proxies on *ProxyTicket*

url = None
Service url of the PGT used for getting the associated *ProxyTicket*

proxy_ticket
ForeignKey to a *ProxyTicket*. *Proxy* instances for a *ProxyTicket* are accessible thought its *ProxyTicket.proxies* attribute.

exception DoesNotExist

exception Proxy.**MultipleObjectsReturned**

Proxy.**objects = <django.db.models.manager.Manager object>**

class cas_server.models.**NewVersionWarning**(**args*, ***kwargs*)
Bases: django.db.models.Model

The last new version available version sent

exception DoesNotExist

exception NewVersionWarning.**MultipleObjectsReturned**

NewVersionWarning.**objects = <django.db.models.manager.Manager object>**

classmethod NewVersionWarning.**send_mails**()
For each new django-cas-server version, if the current instance is not up to date send one mail to `settings.ADMINS`.

2.2.9 cas_server.urls module

urls for the app

2.2.10 cas_server.utils module

Some util function for the app

cas_server.utils.**logger = <logging.Logger object>**
logger facility

cas_server.utils.**json_encode**(*obj*)
Encode a python object to json

cas_server.utils.**context**(*params*)
Function that add somes variable to the context before template rendering

Parameters **params** (*dict*) – The context dictionary used to render templates.

Returns The `params` dictionary with the key `settings` set to `django.conf.settings`.

Return type dict

`cas_server.utils.`**`json_response`**(*request*, *data*)
Wrapper dumping *data* to a json and sending it to the user with an HttpResponse

Parameters

- **`request`** (*`django.http.HttpRequest`*) – The request object used to generate this response.
- **`data`** (*`dict`*) – The python dictionnary to return as a json

Returns The content of `data` serialized in json

Return type django.http.HttpResponse

`cas_server.utils.`**`import_attr`**(*path*)
transform a python dotted path to the attr

Parameters **`path`** (`unicode` or anything) – A dotted path to a python object or a python object

Returns The python object pointed by the dotted path or the python object unchanged

`cas_server.utils.`**`redirect_params`**(*url_name*, *params=None*)
Redirect to `url_name` with `params` as querystring

Parameters

- **`url_name`** (*`unicode`*) – a URL pattern name
- **`params`** (`dict` or `NoneType`) – Some parameter to append to the reversed URL

Returns A redirection to the URL with name `url_name` with `params` as querystring.

Return type django.http.HttpResponseRedirect

`cas_server.utils.`**`reverse_params`**(*url_name*, *params=None*, ***kwargs*)
compute the reverse url of `url_name` and add to it parameters from `params` as querystring

Parameters

- **`url_name`** (*`unicode`*) – a URL pattern name
- **`params`** (`dict` or `NoneType`) – Some parameter to append to the reversed URL
- **`**kwargs`** – additional parameters needed to compure the reverse URL

Returns The computed reverse URL of `url_name` with possible querystring from `params`

Return type unicode

`cas_server.utils.`**`copy_params`**(*get_or_post_params*, *ignore=None*)
copy a `django.http.QueryDict` in a `dict` ignoring keys in the set `ignore`

Parameters

- **`get_or_post_params`** (*`django.http.QueryDict`*) – A GET or POST `QueryDict`
- **`ignore`** (*`set`*) – An optinal set of keys to ignore during the copy

Returns A copy of get_or_post_params

Return type dict

`cas_server.utils.`**`set_cookie`**(*response*, *key*, *value*, *max_age*)
Set the cookie `key` on `response` with value `value` valid for `max_age` secondes

Parameters

- **response** (*django.http.HttpResponse*) – a django response where to set the cookie
- **key** (*unicode*) – the cookie key
- **value** (*unicode*) – the cookie value
- **max_age** (*int*) – the maximum validity age of the cookie

cas_server.utils.**get_current_url**(*request*, *ignore_params=None*)
Giving a django request, return the current http url, possibly ignoring some GET parameters

Parameters

- **request** (*django.http.HttpRequest*) – The current request object.
- **ignore_params** (*set*) – An optional set of GET parameters to ignore

Returns The URL of the current page, possibly omitting some parameters from `ignore_params` in the querystring.

Return type unicode

cas_server.utils.**update_url**(*url*, *params*)
update parameters using `params` in the `url` query string

Parameters

- **url** (`unicode` or `str`) – An URL possibily with a querystring
- **params** (*dict*) – A dictionary of parameters for updating the url querystring

Returns The URL with an updated querystring

Return type unicode

cas_server.utils.**unpack_nested_exception**(*error*)
If exception are stacked, return the first one

Parameters **error** – A python exception with possible exception embeded within

Returns A python exception with no exception embeded within

cas_server.utils.**gen_lt**()
Generate a Login Ticket

Returns A ticket with prefix `settings.CAS_LOGIN_TICKET_PREFIX` and length `settings.CAS_LT_LEN`

Return type unicode

cas_server.utils.**gen_st**()
Generate a Service Ticket

Returns A ticket with prefix `settings.CAS_SERVICE_TICKET_PREFIX` and length `settings.CAS_ST_LEN`

Return type unicode

cas_server.utils.**gen_pt**()
Generate a Proxy Ticket

Returns A ticket with prefix `settings.CAS_PROXY_TICKET_PREFIX` and length `settings.CAS_PT_LEN`

Return type unicode

cas_server.utils.**gen_pgt**()
Generate a Proxy Granting Ticket

Returns A ticket with prefix `settings.CAS_PROXY_GRANTING_TICKET_PREFIX` and length `settings.CAS_PGT_LEN`

Return type unicode

cas_server.utils.**gen_pgtiou**()
Generate a Proxy Granting Ticket IOU

Returns A ticket with prefix `settings.CAS_PROXY_GRANTING_TICKET_IOU_PREFIX` and length `settings.CAS_PGTIOU_LEN`

Return type unicode

cas_server.utils.**gen_saml_id**()
Generate an saml id

Returns A random id of length `settings.CAS_TICKET_LEN`

Return type unicode

cas_server.utils.**get_tuple**(*nuplet*, *index*, *default=None*)

Parameters

- **nuplet** (*tuple*) – A tuple
- **index** (*int*) – An index
- **default** – An optional default value

Returns `nuplet[index]` if defined, else `default` (possibly `None`)

cas_server.utils.**crypt_salt_is_valid**(*salt*)
Validate a salt as crypt salt

Parameters **salt** (*str*) – a password salt

Returns `True` if `salt` is a valid crypt salt on this system, `False` otherwise

Return type bool

class cas_server.utils.**LdapHashUserPassword**
Bases: `object`

Class to deal with hashed password as defined at https://tools.ietf.org/id/draft-stroeder-hashed-userpassword-values-01.html

schemes_salt = set(['{SSHA512}', '{SSHA384}', '{CRYPT}', '{SMD5}', '{SSHA}', '{SSHA256}'])
valide schemes that require a salt

schemes_nosalt = set(['{SHA}', '{SHA512}', '{SHA256}', '{MD5}', '{SHA384}'])
valide sschemes that require no slat

exception BadScheme
Bases: `exceptions.ValueError`

Error raised then the hash scheme is not in *LdapHashUserPassword.schemes_salt* + *LdapHashUserPassword.schemes_nosalt*

exception LdapHashUserPassword.**BadHash**
Bases: `exceptions.ValueError`

Error raised then the hash is too short

exception LdapHashUserPassword.**BadSalt**
Bases: exceptions.ValueError

Error raised then, with the scheme {CRYPT}, the salt is invalid

classmethod LdapHashUserPassword.**hash**(*scheme*, *password*, *salt=None*, *charset='utf8'*)
Hash password with scheme using salt. This three variable beeing encoded in charset.

Parameters

- **scheme** (*bytes*) – A valid scheme
- **password** (*bytes*) – A byte string to hash using scheme
- **salt** (*bytes*) – An optional salt to use if scheme requires any
- **charset** (*str*) – The encoding of scheme, password and salt

Returns The hashed password encoded with charset

Return type bytes

classmethod LdapHashUserPassword.**get_scheme**(*hashed_passord*)
Return the scheme of hashed_passord or raise *BadHash*

Parameters **hashed_passord** (*bytes*) – A hashed password

Returns The scheme used by the hashed password

Return type bytes

Raises **BadHash** – if no valid scheme is found within hashed_passord

classmethod LdapHashUserPassword.**get_salt**(*hashed_passord*)
Return the salt of hashed_passord possibly empty

Parameters **hashed_passord** (*bytes*) – A hashed password

Returns The salt used by the hashed password (empty if no salt is used)

Return type bytes

Raises **BadHash** – if no valid scheme is found within hashed_passord or if the hashed password is too short for the scheme found.

cas_server.utils.**check_password**(*method*, *password*, *hashed_password*, *charset*)
Check that password match *hashed_password* using method, assuming the encoding is charset.

Parameters

- **method** (*str*) – on of "crypt", "ldap", "hex_md5", "hex_sha1", "hex_sha224", "hex_sha256", "hex_sha384", "hex_sha512", "plain"
- **password** (str or unicode) – The user inputed password
- **hashed_password** (str or unicode) – The hashed password as stored in the database
- **charset** (*str*) – The used char encoding (also used internally, so it must be valid for the charset used by password when it was initially)

Returns True if password match hashed_password using method, False otherwise

Return type bool

cas_server.utils.**decode_version**(*version*)
decode a version string following version semantic http://semver.org/ input a tuple of int

Parameters **version** (*unicode*) – A dotted version

Returns A tuple a int

Return type tuple

`cas_server.utils.`**`last_version`**`()`
Fetch the last version from pypi and return it. On successful fetch from pypi, the response is cached 24h, on error, it is cached 10 min.

Returns the last django-cas-server version

Return type unicode

`cas_server.utils.`**`dictfetchall`**`(`*cursor*`)`
Return all rows from a django cursor as a dict

`cas_server.utils.`**`logout_request`**`(`*ticket*`)`
Forge a SLO logout request

Parameters **`ticket`** (*unicode*) – A ticket value

Returns A SLO XML body request

Return type unicode

2.2.11 cas_server.views module

views for the app

class `cas_server.views.`**`LogoutMixin`**
Bases: `object`

destroy CAS session utils

`logout``(`*all_session=False*`)`
effectively destroy a CAS session

Parameters **`all_session`** (*boolean*) – If `True` destroy all the user sessions, otherwise destroy the current user session.

Returns The number of destroyed sessions

Return type int

class `cas_server.views.`**`CsrfExemptView`**`(`***kwargs*`)`
Bases: `django.views.generic.base.View`

base class for csrf exempt class views

`dispatch``(`**args*, ***kwargs*`)`
dispatch different http request to the methods of the same name

Parameters **`request`** (*django.http.HttpRequest*) – The current request object

class `cas_server.views.`**`LogoutView`**`(`***kwargs*`)`
Bases: `django.views.generic.base.View`, *cas_server.views.LogoutMixin*

destroy CAS session (logout) view

`request` **= None**
current `django.http.HttpRequest` object

`service` **= None**
service GET parameter

url = None
url GET paramet

ajax = None
`True` if the HTTP_X_AJAX http header is sent and `settings.CAS_ENABLE_AJAX_AUTH` is `True`, `False` otherwise.

init_get(*request*)
Initialize the *LogoutView* attributes on GET request

Parameters **request** (*django.http.HttpRequest*) – The current request object

get(*request*, **args*, ***kwargs*)
methode called on GET request on this view

Parameters **request** (*django.http.HttpRequest*) – The current request object

class cas_server.views.**FederateAuth**(***kwargs*)
Bases: *cas_server.views.CsrfExemptView*

view to authenticated user agains a backend CAS then CAS_FEDERATE is True

csrf is disabled for allowing SLO requests reception.

service_url = None
current URL used as service URL by the CAS client

get_cas_client(*request*, *provider*, *renew=False*)
return a CAS client object matching provider

Parameters

- **request** (*django.http.HttpRequest*) – The current request object
- **provider** (cas_server.models.FederatedIendityProvider) – the user identity provider

Returns The user CAS client object

Return type *federate.CASFederateValidateUser*

post(*request*, *provider=None*)
method called on POST request

Parameters

- **request** (*django.http.HttpRequest*) – The current request object
- **provider** (*unicode*) – Optional parameter. The user provider suffix.

get(*request*, *provider=None*)
method called on GET request

Parameters

- **request** (*django.http.HttpRequestself.*) – The current request object
- **provider** (*unicode*) – Optional parameter. The user provider suffix.

class cas_server.views.**LoginView**(***kwargs*)
Bases: django.views.generic.base.View, *cas_server.views.LogoutMixin*

credential requestor / acceptor

user = None
The current *models.User* object

form = None
The form to display to the user

request = None
current `django.http.HttpRequest` object

service = None
service GET/POST parameter

renew = None
`True` if renew GET/POST parameter is present and not "False"

warn = None
the warn GET/POST parameter

gateway = None
the gateway GET/POST parameter

method = None
the method GET/POST parameter

ajax = None
`True` if the HTTP_X_AJAX http header is sent and `settings.CAS_ENABLE_AJAX_AUTH` is `True`, `False` otherwise.

renewed = False
`True` if the user has just authenticated

warned = False
`True` if renew GET/POST parameter is present and not "False"

username = None
The *FederateAuth* transmited username (only used if `settings.CAS_FEDERATE` is `True`)

ticket = None
The *FederateAuth* transmited ticket (only used if `settings.CAS_FEDERATE` is `True`)

INVALID_LOGIN_TICKET = 1

USER_LOGIN_OK = 2

USER_LOGIN_FAILURE = 3

USER_ALREADY_LOGGED = 4

USER_AUTHENTICATED = 5

USER_NOT_AUTHENTICATED = 6

init_post(*request*)
Initialize POST received parameters

Parameters **request** (*django.http.HttpRequest*) – The current request object

gen_lt()
Generate a new LoginTicket and add it to the list of valid LT for the user

check_lt()
Check is the POSTed LoginTicket is valid, if yes invalide it

Returns `True` if the LoginTicket is valid, `False` otherwise

Return type bool

post(*request*, **args*, ***kwargs*)
methode called on POST request on this view

Parameters **request** (*django.http.HttpRequest*) – The current request object

process_post()
Analyse the POST request:

•check that the LoginTicket is valid

•check that the user sumited credentials are valid

Returns

- *INVALID_LOGIN_TICKET* if the POSTed LoginTicket is not valid
- *USER_ALREADY_LOGGED* if the user is already logged and do no request reauthentication.
- *USER_LOGIN_FAILURE* if the user is not logged or request for reauthentication and his credentials are not valid
- *USER_LOGIN_OK* if the user is not logged or request for reauthentication and his credentials are valid

Return type int

init_get(*request*)
Initialize GET received parameters

Parameters **request** (*django.http.HttpRequest*) – The current request object

get(*request*, **args*, ***kwargs*)
methode called on GET request on this view

Parameters **request** (*django.http.HttpRequest*) – The current request object

process_get()
Analyse the GET request

Returns

- *USER_NOT_AUTHENTICATED* if the user is not authenticated or is requesting for authentication renewal
- *USER_AUTHENTICATED* if the user is authenticated and is not requesting for authentication renewal

Return type int

init_form(*values=None*)
Initialization of the good form depending of POST and GET parameters

Parameters **values** (*django.http.QueryDict*) – A POST or GET QueryDict

service_login()
Perform login agains a service

Returns

- The rendering of the `settings.CAS_WARN_TEMPLATE` if the user asked to be warned before ticket emission and has not yep been warned.
- The redirection to the service URL with a ticket GET parameter
- The redirection to the service URL without a ticket if ticket generation failed and the *gateway* attribute is set

- The rendering of the `settings.CAS_LOGGED_TEMPLATE` template with some error messages if the ticket generation failed (e.g: user not allowed).

Return type django.http.HttpResponse

authenticated()
Processing authenticated users

Returns

- The returned value of *service_login()* if *service* is defined
- The rendering of `settings.CAS_LOGGED_TEMPLATE` otherwise

Return type django.http.HttpResponse

not_authenticated()
Processing non authenticated users

Returns

- The rendering of `settings.CAS_LOGIN_TEMPLATE` with various messages depending of GET/POST parameters
- The redirection to *FederateAuth* if `settings.CAS_FEDERATE` is `True` and the "remember my identity provider" cookie is found

Return type django.http.HttpResponse

common()
Common part execute uppon GET and POST request

Returns

- The returned value of *authenticated()* if the user is authenticated and not requesting for authentication or if the authentication has just been renewed
- The returned value of *not_authenticated()* otherwise

Return type django.http.HttpResponse

class cas_server.views.**Auth**(***kwargs*)
Bases: *cas_server.views.CsrfExemptView*

A simple view to validate username/password/service tuple

csrf is disable as it is intended to be used by programs. Security is assured by a shared secret between the programs dans django-cas-server.

static post(*request*)
methode called on POST request on this view

Parameters **request** (*django.http.HttpRequest*) – The current request object

Returns `HttpResponse(u"yes\n")` if the POSTed tuple (username, password, service) if valid (i.e. (username, password) is valid dans username is allowed on service). `HttpResponse(u"no\n...")` otherwise, with possibly an error message on the second line.

Return type django.http.HttpResponse

class cas_server.views.**Validate**(***kwargs*)
Bases: `django.views.generic.base.View`

service ticket validation

static get(*request*)
methode called on GET request on this view

Parameters **request** (*django.http.HttpRequest*) – The current request object

Returns

- `HttpResponse("yes\nusername")` if submited (service, ticket) is valid
- else `HttpResponse("no\n")`

Return type django.http.HttpResponse

exception cas_server.views.**ValidationBaseError**(*code*, *msg=''*)
Bases: exceptions.Exception

Base class for both saml and cas validation error

code = None
The error code

msg = None
The error message

render(*request*)
render the error template for the exception

Parameters **request** (*django.http.HttpRequest*) – The current request object:

Returns the rendered `cas_server/serviceValidateError.xml` template

Return type django.http.HttpResponse

exception cas_server.views.**ValidateError**(*code*, *msg=''*)
Bases: *cas_server.views.ValidationBaseError*

handle service validation error

template = 'cas_server/serviceValidateError.xml'
template to be render for the error

context()
content to use to render *template*

Returns A dictionary to contextualize *template*

Return type dict

class cas_server.views.**ValidateService**(***kwargs*)
Bases: django.views.generic.base.View

service ticket validation [CAS 2.0] and [CAS 3.0]

request = None
Current `django.http.HttpRequest` object

service = None
The service GET parameter

ticket = None
the ticket GET parameter

pgt_url = None
the pgtUrl GET parameter

renew = None
the renew GET parameter

allow_proxy_ticket = False
specify if ProxyTicket are allowed by the view. Hence we user the same view for `/serviceValidate` and `/proxyValidate` juste changing the parameter.

get(*request*)
methode called on GET request on this view

Parameters **request** (*django.http.HttpRequest*) – The current request object:

Returns The rendering of `cas_server/serviceValidate.xml` if no errors is raised, the rendering or `cas_server/serviceValidateError.xml` otherwise.

Return type django.http.HttpResponse

process_ticket()
fetch the ticket against the database and check its validity

Raises **ValidateError** – if the ticket is not found or not valid, potentially for that service

Returns A couple (ticket, proxies list)

Return type `tuple`

process_pgturl(*params*)
Handle PGT request

Parameters **params** (*dict*) – A template context dict

Raises **ValidateError** – if pgtUrl is invalid or if TLS validation of the pgtUrl fails

Returns The rendering of `cas_server/serviceValidate.xml`, using `params`

Return type django.http.HttpResponse

class `cas_server.views.`**Proxy**(***kwargs*)
Bases: `django.views.generic.base.View`

proxy ticket service

request = None
Current `django.http.HttpRequest` object

pgt = None
A ProxyGrantingTicket from the pgt GET parameter

target_service = None
the targetService GET parameter

get(*request*)
methode called on GET request on this view

Parameters **request** (*django.http.HttpRequest*) – The current request object:

Returns The returned value of *process_proxy()* if no error is raised, else the rendering of `cas_server/serviceValidateError.xml`.

Return type django.http.HttpResponse

process_proxy()
handle PT request

Raises **ValidateError** – if the PGT is not found, or the target service not allowed or the user not allowed on the tardet service.

Returns The rendering of `cas_server/proxy.xml`

Return type django.http.HttpResponse

exception cas_server.views.**SamlValidateError**(*code*, *msg=''*)
Bases: *cas_server.views.ValidationBaseError*

handle saml validation error

template = 'cas_server/samlValidateError.xml'
template to be render for the error

context()

Returns A dictionary to contextualize *template*

Return type dict

class cas_server.views.**SamlValidate**(***kwargs*)
Bases: *cas_server.views.CsrfExemptView*

SAML ticket validation

request = None

target = None

ticket = None

root = None

post(*request*)
methode called on POST request on this view

Parameters **request** (*django.http.HttpRequest*) – The current request object

Returns the rendering of cas_server/samlValidate.xml if no error is raised, else the rendering of cas_server/samlValidateError.xml.

Return type django.http.HttpResponse

process_ticket()
validate ticket from SAML XML body

Raises SamlValidateError: if the ticket is not found or not valid, or if we fail to parse the posted XML.

Returns a ticket object

Return type *models.Ticket*

2.3 Module contents

A django CAS server application

cas_server.**VERSION = '0.6.4'**
version of the application

cas_server.**default_app_config = 'cas_server.apps.CasAppConfig'**
path the the application configuration class

CHAPTER 3

Indices and tables

- genindex

C

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W