Conference website Documentation

Release 1.0.1

Ulrich Goertz

October 29, 2011

CONTENTS

Contents:

CHAPTER

ONE

INTRODUCTION

A very brief guide to get started:

- Learn at least a little bit of Python: Tutorial
- Go through the Django tutorial, More tutorials
- Install the relevant software, see *Installation of the relevant software packages* (on your "home pc" - you do not need a server which is permanently connected to the internet, at this point)
- Download and install the cws source code: *Obtaining the source code*
- Go through the *Development/quick start* section
- Go through the *Customizing the web pages* chapter
- Go through *Exploring the test data*
- To go public with your CWS instance, see *Deployment on a "real" server*

CHAPTER

TWO

INSTALLING THE SYSTEM

2.1 Requirements

2.1.1 Test server

To test this system, you need a system that runs Django (and some of its extensions), so any common operating system will do. All tests I did were done on Ubuntu Linux systems, and the documentation is based on a Linux environment, as well.

2.1.2 Deployment

For real-life deployment, you will need a server connected to the internet which runs your django instance via a web server (such as Apache), and a mail server (typically running on the same machine as the web server).

2.1.3 User

The users of the web site will need a not-too-outdated web browser (even Internet Explorer 6 should do). Cookies and JavaScript must be enabled.

2.1.4 Admin

For the admin functionality, much heavier use of JavaScript is made. I tested this only on Firefox, and my recommendation is to use the current version of Firefox, or maybe of Google Chrome.

2.2 Installation of the relevant software packages

If you use pip, you can use `pip install -r requirements.txt` after downloading the source of CWS.

Python: The Python programming language. Version 2.7 is recommended. (Currently Django works with the 2.* branch, not the 3.* branch.)

Django The Django web framework. You need version 1.3.*. Download and then install it using `sudo python setup.py install`.

Django easy maps The Django eays maps package. Download and install using `sudo python setup.py install`.

Optional:

Werkzeug, django_extensions Extensions to allow for simple debugging. These are not necessary to run the application, but are highly recommended, if you want to dig deeper and make substantial changes to the code. Once you have installed them, activate them by uncommenting the relevant line in `INSTALLED_APPS` in `settings_local.py`.

IPython provides a better Python shell in combination with the django extensions mentioned above (try `./manage.py shell_plus` instead of `./manage.py shell`).

Ubuntu package: `ipython`.

debug_toolbar Another useful tool for debugging and profiling. After installing, activate it by uncommenting the relevant line in `INSTALLED_APPS` in `settings_local.py`.

2.3 Obtaining the source code

2.3.1 Download tar.gz archive

The easiest way is to download the current `.tar.gz` file from the BitBucket downloads site at

https://bitbucket.org/ugoertz/cws/downloads

(Use the `gz` link for the *tip* tag to obtain the newest version.) Extract the archive with:

```
tar xfz filename.tar.gz
```

2.3.2 hg repository

If you are familiar with the versioning system Mercurial, you can clone the hg repository via:

```
hg clone https://bitbucket.org/ugoertz/cws
```

2.4 Development/quick start

To just set up some working instance of this system, or to do development on it, you can run django on your desktop computer or laptop. Follow these steps:

Assumption: You can invoke a python interpreter and do `import django`. You have obtained the source code of this project (and extracted the archive somewhere). For the purpose of the documentation, the directory with the source code will be called `/home/ug/django/cws/`.

Before running the application, you need to adapt some settings. Copy the file `examples/settings_local.py` to the root directory of your installation:

```
ug@x220:~/django/cws$ cp examples/settings_local.py .
```

and edit the file `settings_local.py`. For the moment, it should suffice to set correct values for the variables:

```
SECRET_KEY, STATICFILES_DIRS, WEBSITE_URL, DEFAULT_SENDER,
SEND_MESSAGES_TO, SEND_BACKUPS_TO
```

(The `SECRET_KEY` should just be changed to some random string.)

Copy all template snippets from `examples/templates` to the `templates` directories:

```
ug@x220:~/django/cws$ cp examples/templates/* templates/
```

Synchronize the database:

```
./manage.py syncdb
```

(choose *no* when asked whether you want to create a superuser) and load some example data:

```
./manage.py loaddata fixtures/testdata.xml
```

This test data includes a superuser with user name `su@test.test` and password `superuser`.

Create a link to the static files used in the django admin interface:

```
ln -s /home/ug/src-django/Django-1.3.1/django/contrib/admin/media static/admin
```

Finally, run the web server:

```
./manage.py runserver
```

If you have the django_extensions installed, you can replace the latter by:

```
./manage.py runserver_plus
```

You should now be able to load the main page by pointing your web browser at:

```
http://localhost:8000/
```

and you should be able to login using the credentials for the superuser given above. **Mail server.** The system assumes that it can send email by delivering it to a mail server running on localhost, listening at port 25 — the usual mail server setup. This means that if a mail server is running on your machine, you should be prepared that the application *will* send out messages on some occasions. On the other hand, for testing purposes, if you do not run a mail server, you can start a fake mail server in a terminal window:

```
sudo /usr/lib/python2.7/smtpd.py -n -c DebuggingServer localhost:25
```

This "server" will accept all email messages delivered to port 25 and simply print them out on the screen. In this way, you can watch which emails would be sent out.

If a mail server is running on your machine, but you do not want to send out messages, you could run the fake mail server at a different port and change the port accordingly in `settings.py`.

To further adapt the application to your conference, see the Chapter *Customizing the web pages*. For a more detailed description of the application in terms of the test data, see *Exploring the test data*.

2.5 Deployment on a "real" server

To deploy the application, you will want to use a *real* web server, such as Apache (with mod_wsgi to execute Python code). The relevant ubuntu packages are:

```
i   apache2-mpm-prefork            - Apache HTTP Server - traditional non-threa
i A apache2-utils                   - utility programs for webservers
i A apache2.2-bin                   - Apache HTTP Server common binary files
i A apache2.2-common                - Apache HTTP Server common files
i   libapache2-mod-wsgi             - Python WSGI adapter module for Apache
```

There are, of course, other options, see Deploying Django.

2.5.1 Apache

Create a directory `apache` in the main CWS directory, copy the file `examples/apache/django.wsgi` to the newly created directory, and edit it to adapt the paths to your setting.

Install Apache 2 and enable mod_wsgi.

An example apache configuration file can be found at. https://docs.djangoproject.com/en/1.3/howto/deployment/modwsgi/ (in our setup, the static files will be found at `/home/ug/django/cws/staticroot`, and there is not `media` directory).

Note that you must call `./manage.py collectstatic` to have all the static files copied to `staticroot/`. See Managing static files.

Note that you must restart (or at least reload) apache if you make changes to `*.py` files of the application. Changes to templates, on the other hand, are effective immediately.

2.5.2 Postgresql

Even if an Sqlite database will be sufficient for most conferences, a postgresql database might be more robust.

Relevant ubuntu packages:

```
i   postgresql-8.4                  - object-relational SQL database, version 8.
i   postgresql-client-8.4           - front-end programs for PostgreSQL 8.4
i A postgresql-client-common        - manager for multiple PostgreSQL client ver
i A postgresql-common               - PostgreSQL database-cluster manager
i   python-psycopg2                 - Python module for PostgreSQL
```

Installing postgresql:

- Install the postgresql packages
- Create a user (and set a password, because django will want to use password authentication):

```
root@e:~# su - postgres
postgres@e:~$ createuser -P
```

- Create a database:

```
postgres@e:~$ createdb django_db
```

- Grant rights to use the database to your user:

```
postgres@e:~$ psql
psql (8.4.8)
Type "help" for help.

postgres=# GRANT ALL PRIVILEGES ON django_db TO username;
```

- Allow password authentication to `username` by adding the following lines at the end of `/etc/postgresql/8.4/main/pg_hba.conf`:

```
host    django_db         username      127.0.0.1 255.255.255.255   password
local   django_db         username                                  password
```

- and restart the database daemon:

```
root@e:~# /etc/init.d/postgresql-8.4 restart
```

2.5.3 Setup

Setting up the application is similar to the quick start example above:

```
ug@x220:~/django/cws$ cp examples/settings_local.py .
```

Edit the file `settings_local.py` appropriately. In addition to the settings mentioned above, you will also have to

- Set `DEBUG=False`. **This is essential for security reasons.**
- Set `STATIC_URL` and `ADMIN_MEDIA_ROOT` to appropriate values (and arrange some way how static files should be served).

Create the relevant template files. To get started, you can simply:

```
ug@x220:~/django/cws$ cp examples/templates/* templates/
```

See Chapter *Customizing the web pages* for details.

Synchronize the database:

```
./manage.py syncdb
```

do not install the test data, but create a superuser:

```
./manage.py createsuperuser --username=joe@example.com --email=joe@example.com
```

You will be prompted for a password.

Create a link to the static files used in the django admin interface:

```
ln -s /home/ug/src-django/Django-1.3.1/django/contrib/admin/media static/admin
```

2.6 Backup

To regularly save a backup of the database, you can use Django's dumpdata command: Create a directory `db-backup` and a file `backup-db`:

```
#!/bin/bash
/home/ug/django/cws/manage.py dumpdata --format=xml > /home/ug/django/cws/db-backup/db`date +%m%d%H`
```

and invoke it regularly by setting up a cron job, e.g.:

```
39 15,21,3,9 * * * /home/ug/django/cws/backup-db
```

Another option is to store backups of the database, e.g., using Postgresql:

```
su postgres -c "pg_dumpall --clean | gzip > /var/lib/postgresql/pg-backup/backup-`date +%d%H`.gz"
```

(this creates a backup of all databases).

2.7 List of software packages used

The following packages are included in the `/static/` directory:

- Jquery
- JqueryUI

- Slickgrid
- Pines notify
- Blueprint CSS
- Silk icon package

CHAPTER

THREE

CUSTOMIZING THE WEB PAGES

3.1 Overview

The following are the adaptations that are absolutely necessary if you want to set up a new instance of the system.

We assume that you have installed this application and that `./manage.py runserver` delivers pages (see the Chapter *Installing the system*). If you want to play around with the application in order to get to know how it works, install the test data by:

```
./manage.py loaddate fixtures/testdata.xml
```

As described in loc.cit. If you want to set up the site for your own conference, omit this step. In order to remove the test data and revert the system to its initial state, you can delete the database (in the case of Sqlite, just delete the database file), and after:

```
./manage.py syncdb
```

create a superuser with:

```
./manage.py createsuperuser --username=joe@example.com --email=joe@example.com
```

Files to be edited (see below for details):

```
settings_local.py,

templates/about, templates/base, templates/contact, templates/create,
templates/home, templates/local1, templates/local2,
templates/schedule, templates/talks, templates/title
templates/travel,
```

for deployment via an Apache web server (see *Deployment on a "real" server*):

```
apache/django.wsgi
```

and optionally:

```
doc/conf.py
```

Create `static/admin` link. (Also: *collect* static files?!)

Objects to be created/edited in the admin interface (at `http://your.url/admin/`) (see below for examples):

```
social events
standard messages
hotels
```

3.2 settings_local.py

Example file for a testing environment:

```
# -*- coding: utf-8 -*-

import os

# Set DEBUG to True while you are testing changes during the development
# process, and in any case set it to False as soon as your site is publically
# accessible
DEBUG = True
TEMPLATE_DEBUG = DEBUG


ADMINS = (
    # ('Your name', 'your@email.test'),
)

MANAGERS = ADMINS


# The database that django should use.
DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.sqlite3', # Add 'postgresql_psycopg2', 'postgresql', 'mysq
            'NAME': './django.db',                      # Or path to database file if using sqlite3.
            'USER': '',                      # Not used with sqlite3.
            'PASSWORD': '',                  # Not used with sqlite3.
            'HOST': '',                      # Set to empty string for localhost. Not used with sqli
            'PORT': '',                      # Set to empty string for default. Not used with sqlite
            }
        }


# Make this unique, and don't share it with anybody.
SECRET_KEY = '4dr+webXXXX0(7dh/nk4sb!8sw*+z=f+*o-3m3t$a&$^lx#@r='

# Absolute path to the directory that holds media.
# Example: "/home/media/media.lawrence.com/"
MEDIA_ROOT = os.path.join(os.path.dirname(__file__), 'media')

# URL that handles the media served from MEDIA_ROOT. Make sure to use a
# trailing slash if there is a path component (optional in other cases).
# Examples: "http://media.lawrence.com", "http://example.com/media/"
MEDIA_URL = '/media/'

# URL prefix for admin media -- CSS, JavaScript and images. Make sure to use a
# trailing slash.
# Examples: "http://foo.com/media/", "/media/".
ADMIN_MEDIA_PREFIX = '/static/admin/'

# Static files,
# see https://docs.djangoproject.com/en/1.3/howto/static-files/
STATICFILES_DIRS = ( "/home/ug/django/cws/static/",)
STATIC_ROOT = "/home/ug/django/cws/staticroot/"
STATIC_URL = "/static/"
```

```
MIDDLEWARE_CLASSES = (
    'django.middleware.common.CommonMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.flatpages.middleware.FlatpageFallbackMiddleware',
    # 'debug_toolbar.middleware.DebugToolbarMiddleware',
)


PROJECT_NAME = os.path.split(os.path.dirname(__file__))[1]

INTERNAL_IPS = ('127.0.0.1',)

DEBUG_TOOLBAR_CONFIG = {'INTERCEPT_REDIRECTS': False}

INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.admin',
    'django.contrib.flatpages',
    'django.contrib.staticfiles',
    'django.contrib.messages',
    'easy_maps',
    'conference',
    'speaker',
    'news',
    'messages',
    # 'django_extensions',
    # 'debug_toolbar',
)


WEBSITE_URL = 'http://your.url.goes.here/'

DEFAULT_SENDER = 'su@test.test'
SEND_MESSAGES_TO = ['su@test.test']
SEND_BACKUPS_TO = ['su@test.test']
CONF_NAME = 'Testing 2011'
FULL_CONF_NAME = 'Test conference, 2011'

START_DAY = "2012-01-02"
END_DAY = "2012-01-08"

# START_DAY_HOTEL and END_DAY_HOTEL are the dates shown as default arrival and
# departure days in the registration form. Typically, START_DAY_HOTEL is one
# day before the conference starts, and END_DAY_HOTEL is the last day of the
# conference. END_DAY_HOTEL_PLUS_ONE is used in the hotel administration page:
# departure days before/after this day are colored differently.
START_DAY_HOTEL = "2012-01-01"
END_DAY_HOTEL = "2012-01-08"
END_DAY_HOTEL_PLUS_ONE = "2012-01-09"

# Should speakers automatically be registered for the conference dinner? If you
# want to use this, then you have to create a SocialEvent tagged "dinner"
# before you enter any speakers.
```

```
REGISTER_SPEAKERS_FOR_DINNER = True

# The Google API key to be used for the map on the travel page. The key is
# bound to an URL, so you have to get your own key at
# http://code.google.com/intl/de/apis/maps/signup.html
# Usually you will not need one, but in particular for development you might
# have to use an API key for "localhost".
# EASY_MAPS_GOOGLE_KEY = ''

# To display the conference venue on the map, we need its name (including an
# address that Google maps can use).
CONFERENCE_VENUE = 'Conference venue,<br>University Essen'
```

Before you expose your application to the public, some changes **must** be made (in particular, you must set `DEBUG=False`!), see *Setup*.

3.3 Templates

As a rule, the files in the `examples/templates` directory should be copied to `templates` and then adapted appropriately. It is no problem, if some of these files are empty, but the files have to be present. Of course, sometimes you might want to dig deeper and also edit the `*.html` files or other files in the template directory.

3.3.1 base

This file concerns the **links shown in the left hand side column**. You can disable some of the links, and you can also add further links. New links could either show to pages on a different web site, or to pages from your django instance. In the latter case you have to make sure that django knows how to serve these pages - either you have to edit `urls.py` and provide a template (and possibly a *view*) for your page, or use the django flatpage framework.

3.3.2 Title

The name of your conference. This will be used as the default *title* in the `<head>` section of the HTML files.

3.3.3 About

Give some information about the person(s) responsible for the site. Depending on your use case, you might need/want to put an *Impressum* according to German regulations here.

3.3.4 Local information

Typically, here you will provide a map, and further local information. Django provides a list of the hotels. For participants with *final* hotel reservation (who are logged in), the hotel they stay in is marked. The template file `local1` contains the content shown above the list of hotels, the file `local2` contains the content shown below.

3.3.5 Schedule

To create the table with the schedule, one possibility is the following.

This is currently very awkward.

Create a ReST table like this:

```
+------------------------------------------------------------------------+
|    Monday                     |    Tuesday                             |
+------------------------------------------------------------------------+
|   Speaker 1                   |    Speaker 2                           |
+-------------------------------+----------------------------------------+
```

(a more complete example is included as `examples/schedule.rst`) and use

```
rst2html.py schedule.rst schedule
```

to convert the file into a HTML file `schedule`. Now delete the `<head>...</head>` section of this file, as well as the `<body>` tag and the `<div class="document">` line, delete the last three lines (`</div>`, `</body>`, `</html>`), and, in the remaining HTML table, delete all occurrences of the word `container` (this is necessary because of a naming conflict with the BlueprintCSS framework).

Finally, copy the file `schedule` so obtained to the `templates` directory.

3.3.6 Travel

In the file `templates/travel` you can put some information on how to reach the location of your conference.

Google map on the *travel* page

A google map will be shown on the travel page. If needed, you can specify a Google maps API key in the `settings_local.py` file, see above. If you want to change the list of airports shown (currently Düsseldorf, Köln/Bonn, Frankfurt), you have to edit the file `templates/map.html`.

3.4 Social events

Add your social events (conference dinner, excursions) in the admin interface at `http://your.url/admin`, e.g.:

```
Tag: dinner

Title: Conference dinner

Description: This is a <b>html</b> description of the dinner. Images can be included
using the img tag, with a full URL, i.e. you have to upload the image to
a location form where static files are served: either the /static/ or
/media/ directories of your django installation, or some independent web
server.
```

The `dinner` tag is special in the sense that all speakers added via the *Add speaker* link are automatically registered for the dinner. (This behavior can be changed in `settings_local.py`.

3.5 Messages

To send *standard messages* (upon registration, to communicate hotel details, to send further information shortly before the conference starts, etc.), you need to *activate* and edit the `StdMessage` objects in the django admin interface. See Chapter *Sending messages to participants*

3.6 Displaying individualized information to participants

To display information (e.g. about hotel details, or your decision about a funding request) to participants in an *Important information* tab on the *Edit my data* page, *activate* (and possibly edit) the `ParticipantOrgComment` objects in the django admin interface.

3.7 News

To add items to the *News* list on the home page, create `News` objects in the django admin interface.

3.8 Hotels

For each hotel that should appear on the site (in the Google map, in the list of hotels on the travel and local info pages, on the pages for administering the hotel details of each participant)

Create the hotel objects in the django admin interface. Most of the fields should be self explanatory. Some comments:

Address should be written as plain text, without HTML markup

URL including http://

Info Here you can use HTML tags such as bold, paragraph .

Price per night dz dz stands for double room

Price remarks Currently not used, but you could put text here and refer to it in the StdMessages or ParticantOrgComments.

The application will try to find the latidude/longitude coordinates of the hotel location from the given address using the Google maps geocoding interface. If this fails, then the hotel will not be shown on the map on the *Travel* page. In this case, find the coordinates yourself from http://maps.google.de/ and enter them in the corresponding Address object in the django admin interface.

3.9 HotelFeature

To get started, you do not have to do anything about this.

`HotelFeature` objects allow you to collect information from participants concerning details of their hotel reservation. Typical cases are:

double The participant requests a double room

late Late arrival - the participant will arrive at the hotel after some specified time (default is 6pm). So, if necessary, you can alert the hotel in advance, or notify the participant about how he can obtain his key.

special A text field for special wishes concerning the hotel reservation.

`HotelFeature` objects for these three cases are created from the initial data which is automatically loaded upon `./manage.py syncdb`. Their values are displayed in the tables on the `/adm/hotel/` and `/adm/table/` pages.

To explain the general mechanism, we go through the fields of a HotelFeature:

Tag some unique identifier of the feature

Description The participant is shown this text together with a checkbox (which by default is not checked). In this way you can collect yes/no information from participants.

Help text If this field is non-empty, then upon selecting the check box, an additional text field is displayed. In this way, you can collect further information in the *yes* case. (Example: if *late arrival* is selected, then the participant can enter more detailed information, such as *at 8pm at the main station*.

3.10 Other things

3.10.1 CSS

The project uses the Blueprint CSS framework, with the following modification:

- Line 65 of `static/blueprint/blueprint/screen.css` was commented out.

3.10.2 Documentation

The documentation was written using Sphinx. To recompile the documentation, install Sphinx,

- run `make html` and open the `index.html` file in `doc/_build/html` in your web browser, or
- run `make latexpdf` and open the pdf file in `doc/_build/latex`

If you want to make changes to the documentation, edit the `*.rst` files (which use the *ReStructured Text* format; see the Sphinx documentation).

CHAPTER

FOUR

ADMINISTRATION

There are two places where administrative tasks can be/have to be taken care of:

Django admin interface. You can access the django admin interface at http://www.your.url/admin/ It allows you to view and edit all the relevant objects in the django database.

Admin interface of the conference site. Some functionality to deal with administrative tasks is offered by CWS. These pages can be accessed via the links on the left hand side which appear once you login as a superuser.

4.1 Exploring the test data

- Install the test data

  ```
  ./manage.py loaddate fixtures/testdata.xml
  ```

  and start the web server on localhost with

  ```
  ./manage.py runserver
  ```

- Open your browser at http://localhost:8000/ and log in as the administrator: `su@test.test`, password `superuser`.
- Activate the registration *Standard Message* (tagged *reg*) in the django admin interface at http://localhost:8000/admin/messages/standardmessage/ (i.e., select the *active* check box and save the object.
- Go to http://localhost:8000/maintenance/. There you will see the pending registration of Leibniz. Select the *accept* box and click *submit*.
- Go to the *Messages to be sent* tab. You will see one item there (the pending registration message to Leibniz). Send registration message to Leibniz. You can view and edit the message here (click on the envelope on the right hand side of the row of icons), and send it, if you like (start a *fake email server* before). You can also discard it, or postpone it (either keeping or discarding your edits).
- Look at hotel reservations (click the *hotel* link). In the table all participants who have requested a hotel reservation are shown (all participants in the test data set). You can assign hotels to the participants here, and also edit whether funding for the hotel is *granted*, or not, and whether the hotel reservation is *final*, or not. **After making changes to one of these fields, you must "leave" the field (by clicking another field or using the tab key) in order to save the changes. A message appears in the upper right corner when changes are saved.**
- Go to the list of participants. As long as you are logged in as an admin, each name has a small icon left to it. Click this icon to edit the data of the corresponding participant. (See *Edit/view participants' data* below for details.) Go to the data for Leibniz and choose the *Messages* tab. Here you see a list of messages that have been sent to Leibniz - in this case the registration message. Click the message title to show its full text. In this tab,

you can also send an individual message to the participant (which will be sent by email). Individual messages sent via the web site will also be listed in the list of messages, so that the participant and admins can later inspect them again.

- To see how the pages look for participants, log in as one of them. For these test data, each email has the form `first_name@test.test`, and the password is `first_name`, e.g. `carl@test.test`, password `carl`:
- Log in as Gauß, edit your talk title and abstract. Change registrations for dinner and excursion. Change other data on the *edit my data* page. (Note that notifications about such changes are sent by email to the address specified in `settings_local.py` - if you have set up a fake email server, you will be able to see this.)

4.2 Assumptions

The application makes the following assumptions. Of course, these could be changed, but changing them would require changing the core application (`template/*.html` and/or `*/*.py` files).

- Participants can ask for funding (hotel/travel). (No mechanism is implemented for participants to upload documents supporting their applications, so - if necessary - such documents (letters of recommendation) must be requested in another way, e.g. by email.)
- Participants can ask for a hotel reservation.
- Hotel and travel costs of the speakers are fully covered.

4.3 Adding participants/speakers

As an admin, you can add participants and speakers via the *Add participant* and *Add speaker* links in the left column.

Unlike participants registering themselves via the registration form, participants/speakers added by an admin do not receive a welcome email message, and their registration is automatically accepted by the system.

Speakers using the web site to register themselves, cannot indicate their *speaker* status, but first are treated as participants. To achieve the equivalent of the *Add speaker* form, you must do the following:

- Accept the registration on the *Maintenance page* (and send/discard the welcome message generated upon acceptance).
- Change the status from *participant* to *speaker* in the *Admin* tab of the edit menu.
- The *final* status of the speaker (i.e., whether he/she will be listed as *to be confirmed*, or not) can currently only be changed via the django admin interface at `http://your.url/admin/speaker/speaker/`. The default is *true*.
- Register the speaker for the dinner in the *Social* tab of the edit menu.

4.4 Maintenance

The *maintenance* page has several tabs:

4.4.1 Pending registrations

Here you can accept/discard incoming registrations.

4.4.2 Messages received

Messages sent from participants via the contact page, or as a message upon registration. You can categorize the messages as new, seen or settled.

4.4.3 Messages to be sent

Here you see a list of *Standard Messages* that are to be sent out to participants. (See also Chapter *Sending messages to participants*.) You can edit those messages, and then send/discard/postpone them.

4.4.4 Cancelling/reactivating registrations

To cancel a registration, go to the participant's data page (via the list of participant), and set *Active to false*, *cancelled to true*.

To reactivate a cancelled registration, go to the *Cancelled reg's* tab on the maintenance page, select the revive box for those you want to reactivate, and submit the data.

4.5 Edit/view participants' data

In the list of participants, to the left of each name there is an icon. Click this icon to get to the data edit page for that participant.

4.6 Download CSV

Clicking the *Download CSV* button, you can download a CSV file with the data displayed in this table.

There are several choices for the selection of participants to be included in the table. Furthermore, you can choose to include/exclude a number of data fields for each participant. Submit your choices, ad you will be shown a HTML table with the data. At the top of that page, there is a *Download CSV* button which allow you to download the data.

CSV stands for *comma separated values* and is a very simple spreadsheet format. You can import these files into LibreOffice or Excel. The separator use is a comma. The character encoding is UTF8.

Depending on your system, it might also be possible to copy-and-paste the HTML table into your spreadsheet program.

4.7 to be confirmed status of speakers

The *final* status of the speaker (i.e., whether he/she will be listed as *to be confirmed*, or not) can be changed via the django admin interface at `http://your.url/admin/speaker/speaker/`.

4.8 Deleting participants

To delete a participant, delete the corresponding User object in the django admin interface.

4.9 News

You can edit and create the news items shown on the home page in the django admin interface.

CHAPTER

FIVE

USER FUNCTIONALITY

5.1 Participants

5.1.1 Registering

For users who are not logged in, a *Register* link is displayed on the left, which leads to the registration form.

5.1.2 Edit data

Participants who are logged in can edit their data in the tabs on the *Edit my data* page.

5.1.3 Information

Personalized information is displayed on the *Local information* and *Travel* pages (for participants with a finialized hotel reservation: their hotel is specially marked), on the *Social program* page, and of course on the *Edit my data page*.

5.2 Speakers

5.2.1 Edit talk

In addition to the things participants can do, to speakers a link *Edit my talk* is displayed, where they can enter the title and abstract of their talk.

CHAPTER

SIX

SENDING MESSAGES TO PARTICIPANTS

6.1 Bulk messages

Some example `StdMessage` objects are created by `./manage.py syncdb` (from initial_data.xml), and need to be edited.

The fields defining a `StdMessage` object are

Sender The email address that will be used as the from address.

Slug A short unique identifier of the message.

Subject The subject of the email.

Template The text of the email. As the name of the field indicates, you can use the django template language here to "individualize" the emails. `p` is the participant object of the participant who will receive this message. The URL of your web site (as specified in `settings_local.py`) is available as `WEBSITE_URL`.

Use the personal salutation of the participants, make some parts conditional on the status (speaker/participant), etc.

See the standard messages in the initial and test data for examples.

Active Messages will be sent only, if this is true.

Preview If true (which is the default), messages will be shown in the *Messages to be sent* tab on the `/maintenance/` page, where you can review (and edit) them. If false, then messages will be sent out automatically.

Filter Here you can define the group of recipients of the message. See the examples for how this works.

Participants Here you can see which participants received the message so far. You should not edit this field yourself, unless you know what you are doing.

When are Standard Messages sent? There is no built-in mechanism in CWS which checks whether the date in an acitve Standard Message's *Date* field has been reached. Rather, Standard Messages currently are sent when either

- The StandardMessage object is saved (in the django admin), or
- The Participant object is saved.

This means that when a message is marked as active, then it will be sent out to all participants (to whom the filter criterion applies). It will also be sent to new participants when the Participant object is saved upon registration.

6.2 Individual messages

You can send an individual message to a participant in the *Messages* tab of the *edit page* for that participant (which you can reach clicking the icon to the left of the participant's name in the list of participants).

6.3 Messages from participants

Via the *contact* page, participants can send messages to you. Those messages are sent via email, but they also appear in the *messages received* tab on the maintenance page, and are listed on the participant's edit page.

CHAPTER

SEVEN

LICENCE

The CWS package is licensed under the MIT License:

```
Copyright (C) 2008-2011 by Ulrich Goertz

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
```

In the /static subdirectory, several software packages are included which come under licences of their own:

- Jquery, Jquery license
- JqueryUI, Jquery license
- Slickgrid, see `static/slickgrid/MIT-LICENSE.txt`
- Pines notify, licenced under GPL/LGPL/MPL
- Blueprint CSS, see `static/blueprint/LICENSE`
- Silk icon package by M. James, CC Attribution, see `static/icon/readme.txt`

CHAPTER

EIGHT

THE SOURCE CODE

8.1 The conference module

This is the main part of the system.

class conference.models.**ChangeRequestHotel**(**args*, ***kwargs*)
ChangeRequestHotel(id, participant_id, author_id, date, arrival, departure, seen, settled, final_comment)

exception ChangeRequestHotel.**DoesNotExist**

exception ChangeRequestHotel.**MultipleObjectsReturned**

ChangeRequestHotel.**author**

ChangeRequestHotel.**get_next_by_arrival**(**moreargs*, ***morekwargs*)

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

ChangeRequestHotel.**get_next_by_departure**(**moreargs*, ***morekwargs*)

ChangeRequestHotel.**get_previous_by_arrival**(**moreargs*, ***morekwargs*)

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

ChangeRequestHotel.**get_previous_by_departure**(**moreargs*, ***morekwargs*)

ChangeRequestHotel.**hotelfeaturecr_set**

ChangeRequestHotel.**objects = <django.db.models.manager.Manager object at 0x23bce10>**

ChangeRequestHotel.**participant**

class conference.models.**Hotel**(**args*, ***kwargs*)
Hotel(id, name, address, contact, url, info, num_reservations, map_address_id, price_per_night, price_per_night_dz, price_remarks, order_by)

exception Hotel.**DoesNotExist**

exception Hotel.**MultipleObjectsReturned**

Hotel.**map_address**

Hotel.**objects = <django.db.models.manager.Manager object at 0x23b77d0>**

Hotel.**overbooked**()

Hotel.**participant_set**

Hotel.**save**()

class conference.models.**HotelFeature**(**args*, ***kwargs*)
HotelFeature(id, tag, description, help_text)

exception HotelFeature.**DoesNotExist**

exception HotelFeature.**MultipleObjectsReturned**

HotelFeature.**hotelfeaturecr_set**

HotelFeature.**hotelfeaturepart_set**

HotelFeature.**objects = <django.db.models.manager.Manager object at 0x23c4510>**

HotelFeature.**participants**

class conference.models.**HotelFeatureCR**(**args*, ***kwargs*)
HotelFeatureCR(id, crequ_id, feature_id, notes)

exception HotelFeatureCR.**DoesNotExist**

exception HotelFeatureCR.**MultipleObjectsReturned**

HotelFeatureCR.**crequ**

HotelFeatureCR.**feature**

HotelFeatureCR.**objects = <django.db.models.manager.Manager object at 0x23c6090>**

class conference.models.**HotelFeaturePart**(**args*, ***kwargs*)
HotelFeaturePart(id, participant_id, feature_id, notes)

exception HotelFeaturePart.**DoesNotExist**

exception HotelFeaturePart.**MultipleObjectsReturned**

HotelFeaturePart.**feature**

HotelFeaturePart.**objects = <django.db.models.manager.Manager object at 0x23c4ad0>**

HotelFeaturePart.**participant**

class conference.models.**Participant**(**args*, ***kwargs*)
Participant(id, user_id, institution, address, country, homepage, password, status, career, arrival, departure, hotel_reservation, hotel_assigned_id, hotel_final, funding_requested_hotel, funding_requested_travel, funding_hotel, funding_hotel_complete, funding_travel, funding_final, sponsor, in_participant_list, accepted, cancelled, time_created, time_lastchanged, added_by_admin, salutation)

Participant.**CAREER_CHOICES = ((0, ‘’), (1, ‘Student’), (2, ‘Post-Doc’), (3, ‘Professor’))**

exception Participant.**DoesNotExist**

exception Participant.**MultipleObjectsReturned**

Participant.**NONE = 0**

Participant.**POSTDOC = 2**

Participant.**PROFESSOR = 3**

Participant.**SPONSOR_CHOICES = ((‘-‘, ‘None’), (‘E’, ‘ERC’), (‘S’, ‘SFB’), (‘H’, ‘HUM’), (‘O’, ‘Oth’))**

Participant.**STATUS_CHOICES = ((‘S’, ‘Speaker’), (‘P’, ‘Participant’))**

Participant.**STUDENT = 1**

Participant.**arrival_le_14**()

Participant.**changerequesthotel_set**

Participant.**delete**()

Participant.**double**()

Participant.**funding_requested**()

Participant.**get_career_display**(**moreargs*, ***morekwargs*)

Participant.**get_duration_of_stay**()

Participant.**get_full_name**()

Participant.**get_next_by_arrival**(**moreargs*, ***morekwargs*)

Participant.**get_next_by_departure**(**moreargs*, ***morekwargs*)

Participant.**get_next_by_time_created**(**moreargs*, ***morekwargs*)

Participant.**get_next_by_time_lastchanged**(**moreargs*, ***morekwargs*)

Participant.**get_participantorgcomments**()

Participant.**get_participantorgcomments_admin**()

Participant.**get_participantorgcomments_available**()

Participant.**get_previous_by_arrival**(**moreargs*, ***morekwargs*)

Participant.**get_previous_by_departure**(**moreargs*, ***morekwargs*)

Participant.**get_previous_by_time_created**(**moreargs*, ***morekwargs*)

Participant.**get_previous_by_time_lastchanged**(**moreargs*, ***morekwargs*)

Participant.**get_sponsor_display**(**moreargs*, ***morekwargs*)

Participant.**get_status_display**(**moreargs*, ***morekwargs*)

Participant.**hotel_assigned**

Participant.**hotelfeature_set**

Participant.**hotelfeaturepart_set**

Participant.**isspeaker**()

Participant.**late**()

Participant.**objects = <conference.models.ParticipantManager object at 0x23bc190>**

Participant.**save**()

Participant.**select**()

Participant.**set_password**(*password=''*)

Set a new password. If the empy string is passed as an argument, a random password is chosen. The new password is saved **as clear text** as self.password, and the password of the corresponding User object is set accordingly.

Participant.**socialevent_set**

Participant.**socialeventpart_set**

Participant.**special**()

Participant.**support_granted_as_requested**()

Participant.**talk_title_given**()

Participant.**tex_address**()

Participant.**user**

class conference.models.**ParticipantManager**

ParticipantManager.**active**()

class conference.models.**SocialEvent**(**args*, ***kwargs*)
SocialEvent(id, tag, title, description, date)

exception SocialEvent.**DoesNotExist**

exception SocialEvent.**MultipleObjectsReturned**

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

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

SocialEvent.**objects = <django.db.models.manager.Manager object at 0x23c6750>**

SocialEvent.**participants**

SocialEvent.**socialeventpart_set**

class conference.models.**SocialEventPart**(**args*, ***kwargs*)
SocialEventPart(id, event_id, participant_id, num_persons)

exception SocialEventPart.**DoesNotExist**

exception SocialEventPart.**MultipleObjectsReturned**

SocialEventPart.**event**

SocialEventPart.**objects = <django.db.models.manager.Manager object at 0x23c6cd0>**

SocialEventPart.**participant**

conference.models.**changes**(*self*)

8.2 Speakers

class speaker.models.**Speaker**(**args*, ***kwargs*)
Speaker(id, participant_id, last_name, first_name, plenary, final)

exception Speaker.**DoesNotExist**

exception Speaker.**MultipleObjectsReturned**

Speaker.**full_name**()

Speaker.**objects = <django.db.models.manager.Manager object at 0x2ba0190>**

Speaker.**participant**

Speaker.**talk_set**

class speaker.models.**Talk**(**args*, ***kwargs*)
Talk(id, speaker_id, title, abstract, comments_to_organizers, possible_date_from, possible_date_to, date)

exception Talk.**DoesNotExist**

exception Talk.**MultipleObjectsReturned**

Talk.**get_next_by_possible_date_from**(**moreargs*, ***morekwargs*)

Talk.**get_next_by_possible_date_to**(**moreargs*, ***morekwargs*)

Talk.**get_previous_by_possible_date_from**(**moreargs*, ***morekwargs*)

Talk.**get_previous_by_possible_date_to**(**moreargs*, ***morekwargs*)

Talk.**objects = <django.db.models.manager.Manager object at 0x2ba0610>**

Talk.**save**()

Talk.**speaker**

8.3 Messages

class messages.models.**Message**(**args*, ***kwargs*)
messages to participants

sources: * individual messages from edit_admin page * bulk messages from StandardMessage class

as long as sent==False, they have a preview status (and can be edited & sent/discarded in maintenance mode)

exception Message.**DoesNotExist**

exception Message.**MultipleObjectsReturned**

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

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

Message.**objects = <messages.models.MessageManager object at 0x2c97b50>**

Message.**participant**

Message.**send_and_save**()

Message.**stdmsg**

class messages.models.**MessageManager**

MessageManager.**preview**()

MessageManager.**sent**()

MessageManager.**use_for_related_fields = True**

class messages.models.**ParticipantComment**(**args*, ***kwargs*)
A messages sent from the participant, using the contact page.

exception ParticipantComment.**DoesNotExist**

exception ParticipantComment.**MultipleObjectsReturned**

ParticipantComment.**TOPIC_CHOICES = (('-', 'Please choose a topic'), ('R', 'Registration'), ('H', 'Hotel reservation**

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

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

ParticipantComment.**get_topic_display**(**moreargs*, ***morekwargs*)

ParticipantComment.**objects = <django.db.models.manager.Manager object at 0x2c94090>**

ParticipantComment.**participant**

class messages.models.**ParticipantInvComment**(**args*, ***kwargs*)
ParticipantInvComment(id, author_id, participant_id, text, date)

exception ParticipantInvComment.**DoesNotExist**

exception ParticipantInvComment.**MultipleObjectsReturned**

ParticipantInvComment.**author**

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

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

ParticipantInvComment.**objects = <django.db.models.manager.Manager object at 0x2c94610>**

ParticipantInvComment.**participant**

class messages.models.**ParticipantOrgComment**(**args*, ***kwargs*)
ParticipantOrgComment(id, title, template, filters, active)

exception ParticipantOrgComment.**DoesNotExist**

exception ParticipantOrgComment.**MultipleObjectsReturned**

ParticipantOrgComment.**objects = <messages.models.ParticipantOrgCommentManager object at 0x2c97510>**

ParticipantOrgComment.**relevant_for**(*p*)

class messages.models.**ParticipantOrgCommentManager**

ParticipantOrgCommentManager.**active**()

class messages.models.**StandardMessage**(**args*, ***kwargs*)
StandardMessage(id, sender, slug, subject, template, active, preview, filters)

exception StandardMessage.**DoesNotExist**

exception StandardMessage.**MultipleObjectsReturned**

StandardMessage.**message_set**

StandardMessage.**objects = <messages.models.StandardMessageManager object at 0x2c94a90>**

StandardMessage.**participants**

StandardMessage.**relevant_ps**()

StandardMessage.**send_all**()

StandardMessage.**send_p**(*p*)

StandardMessage.**show_ex**()

class messages.models.**StandardMessageManager**

StandardMessageManager.**active**()

messages.models.**build_query_filter_from_spec**(*spec*, *field_mapping=None*)
from http://www.djangosnippets.org/snippets/676/ # fixes by ug

Assemble a django "Q" query filter object from a specification that consists of a possibly-nested list of query filter descriptions. These descriptions themselves specify Django primitive query filters, along with boolean "and", "or", and "not" operators. This format can be serialized and deserialized, allowing django queries to be composed client-side and sent across the wire using JSON.

Each filter description is a list. The first element of the list is always the filter operator name. This name is one of either django's filter operators, "eq" (a synonym for "exact"), or the boolean operators "and", "or", and "not".

Primitive query filters have three elements:

[filteroperator, fieldname, queryarg]

"filteroperator" is a string name like "in", "range", "icontains", etc. "fieldname" is the django field being queried. Any name that django accepts is allowed, including references to fields in foreign keys using the "__" syntax described in the django API reference. "queryarg" is the argument you'd pass to the *filter()* method in the Django database API.

"and" and "or" query filters are lists that begin with the appropriate operator name, and include subfilters as additional list elements:

['or', [subfilter], ...] ['and', [subfilter], ...]

"not" query filters consist of exactly two elements:

['not', [subfilter]]

As a special case, the empty list "[]" or None return all elements.

If field_mapping is specified, the field name provided in the spec is looked up in the field_mapping dictionary. If there's a match, the result is subsitituted. Otherwise, the field name is used unchanged to form the query. This feature allows client-side programs to use "nice" names that can be mapped to more complex django names. If you decide to use this feature, you'll probably want to do a similar mapping on the field names being returned to the client.

This function returns a Q object that can be used anywhere you'd like in the django query machinery.

This function raises ValueError in case the query is malformed, or perhaps other errors from the underlying DB code.

Example queries:

['and', ['contains', 'name', 'Django'], ['range', 'apps', [1, 4]]] ['not', ['in', 'tags', ['colors', 'shapes', 'animals']]] ['or', ['eq', 'id', 2], ['icontains', 'city', 'Boston']]

messages.models.**new_msgs**(*self*)
Return queryset with all new messages.

messages.models.**on_save_participant**(*instance*, ***kwargs*)

messages.models.**on_save_stdmessage**(*instance*, ***kwargs*)

messages.models.**seen_msgs**(*self*)

messages.models.**sent_stdmsgs**(*self*)

8.4 News

class news.models.**News**(**args*, ***kwargs*)
News(id, title, date, text)

exception News.**DoesNotExist**

exception News.**MultipleObjectsReturned**

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

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

News.**objects = <django.db.models.manager.Manager object at 0x2980790>**

CHAPTER

NINE

INDICES AND TABLES

- *genindex*
- *search*

INDEX