Indico Documentation

Release 2.0.dev0

Indico Team

Sep 21, 2017

Contents

The effortless open source tool for event organization, archival and collaboration.

Welcome to Indico's documentation. This documentation is split into several parts, from installing Indico to developing Indico plugins. To dive into the internals of Indico, check out the API documentation. Read more about Indico in our official website.

CHAPTER 1

Installation / Configuration

To simply install and use Indico, follow the *production installation instructions*. For those who are interested in developing new features and plugins for Indico, check out the *development installation instructions*.

Installation guides

To simply install and use Indico, follow the *production installation instructions*. For those who are interested in developing new features and plugins for Indico, check out the *development installation instructions*.

Production

We provide guides to install Indico on CentOS and Debian systems. While other distributions are not officially supported, they should work fine, but the installation steps (especially package names) may need some slight adjustments.

Our guides cover a single-machine installation where Indico, Celery, Redis and PostgreSQL run on the same machine. This should be fine for almost all Indico instances, but adapting the steps to multiple machines is not particularly hard either.

CentOS7 / CC7

Except for minor differences, these guides apply to both vanilla CentOS7 and the CERN flavor of CentOS, CC7 (CentOS CERN 7).

nginx

Note: Please note that you **must** use Apache if you intend to use SSO using Shibboleth/SAML/ADFS. If that's not the case because you do not use SSO at all or use e.g. OAuth, we recommend using nginx.

1. Enable EPEL

```
yum install -y epel-release
```

Note: If you use CC7, EPEL is already enabled and this step is not necessary

2. Install Packages

Edit `/etc/yum.repos.d/CentOS-Base.repo` and add `exclude=postgresql*` to the `[base]` and `[updates]` sections, as described in the PostgreSQL wiki.

```
yum install -y https://download.postgresql.org/pub/repos/yum/9.6/redhat/rhel-7-x86_64/
↪pgdg-centos96-9.6-3.noarch.rpm
yum install -y postgresql96 postgresql96-server postgresql96-libs postgresql96-devel␣
↪postgresql96-contrib
yum install -y gcc redis nginx uwsgi uwsgi-plugin-python
yum install -y python-devel python-virtualenv libjpeg-turbo-devel libxslt-devel␣
↪libxml2-devel libffi-devel pcre-devel libyaml-devel
/usr/pgsql-9.6/bin/postgresql96-setup initdb
systemctl start postgresql-9.6.service redis.service
```

3. Create a Database

We create a user and database for indico and enable the necessary Postgres extensions (which can only be done by the Postgres superuser)

```
su - postgres -c 'createuser indico'
su - postgres -c 'createdb -O indico indico'
su - postgres -c 'psql indico -c "CREATE EXTENSION unaccent; CREATE EXTENSION pg_trgm;
↪"'
```

Warning: Do not forget to setup a cronjob that creates regular database backups once you start using Indico in production!

4. Configure uWSGI & nginx

The default uWSGI and nginx configuration files should work fine in most cases.

```
cat > /etc/uwsgi.ini <<'EOF'
[uwsgi]
uid = indico
gid = nginx
umask = 027
pidfile = /run/uwsgi/uwsgi.pid

processes = 4
enable-threads = true
chmod-socket = 770
```

```
socket = /opt/indico/web/uwsgi.sock
stats = /opt/indico/web/uwsgi-stats.sock
protocol = uwsgi

master = true
auto-procname = true
procname-prefix-spaced = indico
disable-logging = true

plugin = python
single-interpreter = true

touch-reload = /opt/indico/web/indico.wsgi
wsgi-file = /opt/indico/web/indico.wsgi
virtualenv = /opt/indico/.venv

vacuum = true
buffer-size = 20480
memory-report = true
max-requests = 2500
harakiri = 900
harakiri-verbose = true
reload-on-rss = 2048
evil-reload-on-rss = 8192
EOF
```

Note: Replace `YOURHOSTNAME` in the next file with the hostname on which your Indico instance should be available, e.g. `indico.yourdomain.com`

```
cat > /etc/nginx/conf.d/indico.conf <<'EOF'
server {
  listen 80;
  listen [::]:80;
  server_name YOURHOSTNAME;
  return 301 https://$server_name$request_uri;
}

server {
  listen       *:443 ssl http2;
  listen       [::]:443 ssl http2 default ipv6only=on;
  server_name  YOURHOSTNAME;

  ssl on;

  ssl_certificate           /etc/ssl/indico/indico.crt;
  ssl_certificate_key       /etc/ssl/indico/indico.key;
  ssl_session_cache         shared:SSL:10m;
  ssl_session_timeout       5m;
  ssl_protocols             TLSv1 TLSv1.1 TLSv1.2;
  ssl_ciphers               ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-
↪POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-
↪AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-
↪AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-
↪AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-
↪SHA384:ECDHE-ECDSA-AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-
↪AES128-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-ECDSA-DES-CBC3-SHA:ECDHE-
↪RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-
↪SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:DES-CBC3-SHA:!DSS;
```

```
  ssl_prefer_server_ciphers on;

  access_log            /opt/indico/log/nginx/access.log combined;
  error_log             /opt/indico/log/nginx/error.log;

  location /.xsf/indico/ {
    internal;
    alias /opt/indico/;
  }

  location ~ ^/static/assets/(core|(?:plugin|theme)-[^/]+)/(.*)$ {
    alias /opt/indico/assets/$1/$2;
    access_log off;
  }

  location ~ ^/(css|images|js|static(?!/plugins|/assets|/themes|/custom))(/.*)$ {
    alias /opt/indico/web/htdocs/$1$2;
    access_log off;
  }

  location /robots.txt {
    alias /opt/indico/web/htdocs/robots.txt;
    access_log off;
  }

  location / {
    root /var/empty/nginx;
    include /etc/nginx/uwsgi_params;
    uwsgi_pass unix:/opt/indico/web/uwsgi.sock;
    uwsgi_param UWSGI_SCHEME $scheme;
    uwsgi_read_timeout 15m;
    uwsgi_buffers 32 32k;
    uwsgi_busy_buffers_size 128k;
    uwsgi_hide_header X-Sendfile;
    client_max_body_size 1G;
  }
}
EOF
```

5. Create an SSL Certificate

First, create the folders for the certificate/key and set restrictive permissions on them:

```
mkdir /etc/ssl/indico
chown root:root /etc/ssl/indico/
chmod 700 /etc/ssl/indico
```

If you are just trying out Indico you can simply use a self-signed certificate (your browser will show a warning which you will have to confirm when accessing your Indico instance for the first time).

Note: Do not forget to replace `YOURHOSTNAME` with the same value you used above

```
openssl req -x509 -nodes -newkey rsa:4096 -subj /CN=YOURHOSTNAME -keyout /etc/ssl/
↪indico/indico.key -out /etc/ssl/indico/indico.crt
```

While a self-signed certificate works for testing, it is not suitable for a production system. You can either buy a certificate from any commercial certification authority or get a free one from Let's Encrypt.

Note: There's an optional step later in this guide to get a certificate from Let's Encrypt. We can't do it right now since the nginx config references a directory yet to be created, which prevents nginx from starting.

6. Configure SELinux

Indico works fine with SELinux enabled, but you need to load a custom SELinux module to tell SELinux about Indico's files and how they should be handled.

```
cat > /tmp/indico.cil <<'EOF'
; define custom type that logrotate can access
(type indico_log_t)
(typeattributeset file_type (indico_log_t))
(typeattributeset logfile (indico_log_t))
(roletype object_r indico_log_t)

; allow logrotate to reload systemd services
(allow logrotate_t init_t (service (start)))
(allow logrotate_t policykit_t (dbus (send_msg)))
(allow policykit_t logrotate_t (dbus (send_msg)))

; make sure the uwsgi socket is writable by the webserver
(typetransition unconfined_service_t usr_t sock_file "uwsgi.sock" httpd_sys_rw_
↪content_t)
(filecon "/opt/indico/web/uwsgi\.sock" socket (system_u object_r httpd_sys_rw_content_
↪t ((s0)(s0))))

; set proper types for our log dirs
(filecon "/opt/indico/log(/.*)?" any (system_u object_r indico_log_t ((s0)(s0))))
(filecon "/opt/indico/log/nginx(/.*)?" any (system_u object_r httpd_log_t ((s0)(s0))))
EOF
semodule -i /tmp/indico.cil
```

7. Install Indico

Celery runs as a background daemon. Add a systemd unit file for it:

```
cat > /etc/systemd/system/indico-celery.service <<'EOF'
[Unit]
Description=Indico Celery
After=network.target

[Service]
ExecStart=/opt/indico/.venv/bin/indico celery worker -B
Restart=always
SyslogIdentifier=indico-celery
User=indico
Group=nginx
UMask=0027
Type=simple
```

```
[Install]
WantedBy=multi-user.target
EOF
systemctl daemon-reload
```

Now create a user that will be used to run Indico and switch to it:

```
useradd -rm -g nginx -d /opt/indico -s /bin/bash indico
su - indico
```

You are now ready to install Indico:

```
virtualenv ~/.venv
source ~/.venv/bin/activate
pip install -U pip setuptools
pip install --pre indico
```

8. Configure Indico

Once Indico is installed, you can run the configuration wizard. You can keep the defaults for most options, but make sure to use `https://YOURHOSTNAME` when prompted for the Indico URL. Also specify valid email addresses when asked and enter a valid SMTP server Indico can use to send emails. When asked for the default timezone make sure this is the main time zone used in your Indico instance.

```
indico setup wizard
```

Now finish setting up the directory structure and permissions:

```
mkdir ~/log/nginx
chmod go-rwx ~/* ~/.[^.]*
chmod 710 ~/ ~/archive ~/assets ~/cache ~/log ~/tmp
chmod 750 ~/web ~/.venv
chmod g+w ~/log/nginx
restorecon -R ~/
echo -e "\nSTATIC_FILE_METHOD = ('xaccelredirect', {'/opt/indico': '/.xsf/indico'})" >
↪> ~/etc/indico.conf
```

9. Create database schema

Finally you can create the database schema and switch back to *root*:

```
indico db prepare
exit
```

10. Launch Indico

You can now start Indico and set it up to start automatically when the server is rebooted:

```
systemctl restart uwsgi.service nginx.service indico-celery.service
systemctl enable uwsgi.service nginx.service postgresql-9.6.service redis.service␣
↪indico-celery.service
```

11. Open the Firewall

```
firewall-cmd --permanent --add-port 443/tcp --add-port 80/tcp
firewall-cmd --reload
```

Note: This is only needed if you use CC7 as CentOS7 has no firewall enabled by default

12. Optional: Get a Certificate from Let's Encrypt

To avoid ugly SSL warnings in your browsers, the easiest option is to get a free certificate from Let's Encrypt. We also enable the cronjob to renew it automatically:

```
yum install -y python-certbot-nginx
certbot --nginx --rsa-key-size 4096 --no-redirect --staple-ocsp -d YOURHOSTNAME
rm -rf /etc/ssl/indico
systemctl start certbot-renew.timer
systemctl enable certbot-renew.timer
```

13. Create an Indico user

Access `https://YOURHOSTNAME` in your browser and follow the steps displayed there to create your initial user.

Apache

1. Enable EPEL

```
yum install -y epel-release
```

Note: If you use CC7, EPEL is already enabled and this step is not necessary

2. Install Packages

Edit `/etc/yum.repos.d/CentOS-Base.repo` and add `exclude=postgresql*` to the `[base]` and `[updates]` sections, as described in the PostgreSQL wiki.

```
yum install -y https://download.postgresql.org/pub/repos/yum/9.6/redhat/rhel-7-x86_64/
↪pgdg-centos96-9.6-3.noarch.rpm
yum install -y postgresql96 postgresql96-server postgresql96-libs postgresql96-devel␣
↪postgresql96-contrib
yum install -y httpd mod_proxy_uwsgi mod_ssl mod_xsendfile
yum install -y gcc redis uwsgi uwsgi-plugin-python
yum install -y python-devel python-virtualenv libjpeg-turbo-devel libxslt-devel␣
↪libxml2-devel libffi-devel pcre-devel libyaml-devel
/usr/pgsql-9.6/bin/postgresql96-setup initdb
systemctl start postgresql-9.6.service redis.service
```

3. Create a Database

We create a user and database for indico and enable the necessary Postgres extensions (which can only be done by the Postgres superuser)

```
su - postgres -c 'createuser indico'
su - postgres -c 'createdb -O indico indico'
su - postgres -c 'psql indico -c "CREATE EXTENSION unaccent; CREATE EXTENSION pg_trgm;
↪"'
```

Warning: Do not forget to setup a cronjob that creates regular database backups once you start using Indico in production!

4. Configure uWSGI & Apache

The default uWSGI and Apache configuration files should work fine in most cases.

```
cat > /etc/uwsgi.ini <<'EOF'
[uwsgi]
uid = indico
gid = apache
umask = 027
pidfile = /run/uwsgi/uwsgi.pid

processes = 4
enable-threads = true
socket = 127.0.0.1:8008
stats = /opt/indico/web/uwsgi-stats.sock
protocol = uwsgi

master = true
auto-procname = true
procname-prefix-spaced = indico
disable-logging = true

plugin = python
single-interpreter = true

touch-reload = /opt/indico/web/indico.wsgi
wsgi-file = /opt/indico/web/indico.wsgi
virtualenv = /opt/indico/.venv

vacuum = true
buffer-size = 20480
memory-report = true
max-requests = 2500
harakiri = 900
harakiri-verbose = true
reload-on-rss = 2048
evil-reload-on-rss = 8192
EOF
```

Note: Replace `YOURHOSTNAME` in the next files with the hostname on which your Indico instance should be avail-

able, e.g. `indico.yourdomain.com`

```
cat > /etc/httpd/conf.d/indico-sslredir.conf <<'EOF'
<VirtualHost *:80>
    ServerName YOURHOSTNAME
    RewriteEngine On
    RewriteRule ^(.*)$ https://%{HTTP_HOST}$1 [R=301,L]
</VirtualHost>
EOF

cat > /etc/httpd/conf.d/indico.conf <<'EOF'
<VirtualHost *:443>
    ServerName YOURHOSTNAME
    DocumentRoot "/var/empty/apache"

    SSLEngine               on
    SSLCertificateFile      /etc/ssl/indico/indico.crt
    SSLCertificateChainFile /etc/ssl/indico/indico.crt
    SSLCertificateKeyFile   /etc/ssl/indico/indico.key
    SSLProtocol             all -SSLv2 -SSLv3
    SSLCipherSuite          ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-
↪POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-
↪AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-
↪AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-
↪AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-
↪SHA384:ECDHE-ECDSA-AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-
↪AES128-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-ECDSA-DES-CBC3-SHA:ECDHE-
↪RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-
↪SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:DES-CBC3-SHA:!DSS
    SSLHonorCipherOrder     on

    XSendFile on
    XSendFilePath /opt/indico
    CustomLog /opt/indico/log/apache/access.log combined
    ErrorLog /opt/indico/log/apache/error.log
    LogLevel error
    ServerSignature Off

    AliasMatch "^/static/assets/(core|(?:plugin|theme)-[^/]+)/(.*)$" "/opt/indico/
↪assets/$1/$2"
    AliasMatch "^/(css|images|js|static(?!/plugins|/assets|/themes|/custom))(/.*)$" "/
↪opt/indico/web/htdocs/$1$2"
    Alias /robots.txt /opt/indico/web/htdocs/robots.txt

    SetEnv UWSGI_SCHEME https
    ProxyPass / uwsgi://127.0.0.1:8008/

    <Directory /opt/indico>
        AllowOverride None
        Require all granted
    </Directory>
</VirtualHost>
EOF
```

Now enable the uwsgi proxy module in apache:

```
echo 'LoadModule proxy_uwsgi_module modules/mod_proxy_uwsgi.so' > /etc/httpd/conf.
↪modules.d/proxy_uwsgi.conf
```

5. Create an SSL Certificate

First, create the folders for the certificate/key and set restrictive permissions on them:

```
mkdir /etc/ssl/indico
chown root:root /etc/ssl/indico/
chmod 700 /etc/ssl/indico
```

If you are just trying out Indico you can simply use a self-signed certificate (your browser will show a warning which you will have to confirm when accessing your Indico instance for the first time).

Note: Do not forget to replace `YOURHOSTNAME` with the same value you used above

```
openssl req -x509 -nodes -newkey rsa:4096 -subj /CN=YOURHOSTNAME -keyout /etc/ssl/
↪indico/indico.key -out /etc/ssl/indico/indico.crt
```

While a self-signed certificate works for testing, it is not suitable for a production system. You can either buy a certificate from any commercial certification authority or get a free one from Let's Encrypt.

Note: There's an optional step later in this guide to get a certificate from Let's Encrypt. We can't do it right now since the Apache config references a directory yet to be created, which prevents Apache from starting.

6. Configure SELinux

Indico works fine with SELinux enabled, but you need to load a custom SELinux module to tell SELinux about Indico's files and how they should be handled.

```
cat > /tmp/indico.cil <<'EOF'
; define custom type that logrotate can access
(type indico_log_t)
(typeattributeset file_type (indico_log_t))
(typeattributeset logfile (indico_log_t))
(roletype object_r indico_log_t)

; allow logrotate to reload systemd services
(allow logrotate_t init_t (service (start)))
(allow logrotate_t policykit_t (dbus (send_msg)))
(allow policykit_t logrotate_t (dbus (send_msg)))

; make sure the uwsgi socket is writable by the webserver
(typetransition unconfined_service_t usr_t sock_file "uwsgi.sock" httpd_sys_rw_
↪content_t)
(filecon "/opt/indico/web/uwsgi\.sock" socket (system_u object_r httpd_sys_rw_content_
↪t ((s0)(s0))))

; set proper types for our log dirs
(filecon "/opt/indico/log(/.*)?" any (system_u object_r indico_log_t ((s0)(s0))))
(filecon "/opt/indico/log/apache(/.*)?" any (system_u object_r httpd_log_t␣
↪((s0)(s0))))
```

```
EOF
semodule -i /tmp/indico.cil
```

7. Install Indico

Celery runs as a background daemon. Add a systemd unit file for it:

```
cat > /etc/systemd/system/indico-celery.service <<'EOF'
[Unit]
Description=Indico Celery
After=network.target

[Service]
ExecStart=/opt/indico/.venv/bin/indico celery worker -B
Restart=always
SyslogIdentifier=indico-celery
User=indico
Group=apache
UMask=0027
Type=simple

[Install]
WantedBy=multi-user.target
EOF
systemctl daemon-reload
```

Now create a user that will be used to run Indico and switch to it:

```
useradd -rm -g apache -d /opt/indico -s /bin/bash indico
su - indico
```

You are now ready to install Indico:

```
virtualenv ~/.venv
source ~/.venv/bin/activate
pip install -U pip setuptools
pip install --pre indico
```

8. Configure Indico

Once Indico is installed, you can run the configuration wizard. You can keep the defaults for most options, but make sure to use `https://YOURHOSTNAME` when prompted for the Indico URL. Also specify valid email addresses when asked and enter a valid SMTP server Indico can use to send emails. When asked for the default timezone make sure this is the main time zone used in your Indico instance.

```
indico setup wizard
```

Now finish setting up the directory structure and permissions:

```
mkdir ~/log/apache
chmod go-rwx ~/* ~/.[^.]*
chmod 710 ~/ ~/archive ~/assets ~/cache ~/log ~/tmp
chmod 750 ~/web ~/.venv
chmod g+w ~/log/apache
```

```
restorecon -R ~/
echo -e "\nSTATIC_FILE_METHOD = 'xsendfile'" >> ~/etc/indico.conf
```

9. Create database schema

Finally you can create the database schema and switch back to *root*:

```
indico db prepare
exit
```

10. Launch Indico

You can now start Indico and set it up to start automatically when the server is rebooted:

```
systemctl restart uwsgi.service httpd.service indico-celery.service
systemctl enable uwsgi.service httpd.service postgresql-9.6.service redis.service␣
↪indico-celery.service
```

11. Open the Firewall

```
firewall-cmd --permanent --add-port 443/tcp --add-port 80/tcp
firewall-cmd --reload
```

Note: This is only needed if you use CC7 as CentOS7 has no firewall enabled by default

12. Optional: Get a Certificate from Let's Encrypt

To avoid ugly SSL warnings in your browsers, the easiest option is to get a free certificate from Let's Encrypt. We also enable the cronjob to renew it automatically:

```
yum install -y python-certbot-apache
certbot --apache --rsa-key-size 4096 --no-redirect --staple-ocsp -d YOURHOSTNAME
rm -rf /etc/ssl/indico
systemctl start certbot-renew.timer
systemctl enable certbot-renew.timer
```

13. Create an Indico user

Access `https://YOURHOSTNAME` in your browser and follow the steps displayed there to create your initial user.

Optional: Shibboleth

If your organization uses Shibboleth/SAML-based SSO, follow these steps to use it in Indico:

1. Install Shibboleth

Add the Shibboleth yum repository:

Note: If you use CC7, Shibboleth is already available and there is no need to add the repo manually.

```
curl -fsSL -o /etc/yum.repos.d/shibboleth.repo 'https://shibboleth.net/cgi-bin/sp_
↪repo.cgi?platform=CentOS_7'
```

Now install Shibboleth itself. When prompted to accept the GPG key of the Shibboleth yum repo, confirm the prompt.

```
setsebool httpd_can_network_connect 1
yum install -y shibboleth xmltooling-schemas opensaml-schemas
```

2. Configure Shibboleth

This is outside the scope of this documentation and depends on your environment (Shibboleth, SAML, ADFS, etc). Please contact whoever runs your SSO infrastructure if you need assistance.

3. Enable Shibboleth in Apache

Add the following code to your `/etc/httpd/conf.d/indico.conf` right before the `AliasMatch` lines:

```
<LocationMatch "^(/Shibboleth\.sso|/login/shib-sso/shibboleth)">
    AuthType shibboleth
    ShibRequestSetting requireSession 1
    ShibExportAssertion Off
    Require valid-user
</LocationMatch>
```

4. Enable Shibboleth in Indico

Add the following code to your `/opt/indico/etc/indico.conf`:

```
# SSO
AUTH_PROVIDERS = {
    'shib-sso': {
        'type': 'shibboleth',
        'title': 'SSO',
        'attrs_prefix': 'ADFS_',
        'callback_uri': '/login/shib-sso/shibboleth',
        # 'logout_uri': 'https://login.yourcompany.tld/logout'
    }
}
IDENTITY_PROVIDERS = {
    'shib-sso': {
        'type': 'shibboleth',
        'title': 'SSO',
        'identifier_field': 'ADFS_LOGIN',
        'mapping': {
            'affiliation': 'ADFS_HOMEINSTITUTE',
```

```
            'first_name': 'ADFS_FIRSTNAME',
            'last_name': 'ADFS_LASTNAME',
            'email': 'ADFS_EMAIL',
            'phone': 'ADFS_PHONENUMBER'
        },
        'trusted_email': True
    }
}
```

The values for `attrs_prefix`, `mapping` and `identifier_field` may be different in your environment. Uncomment and set `logout_uri` if your SSO infrastructure provides a logout URL (usually used to log you out from all applications).

If you only want to use SSO, without allowing people to login locally using username/password, disable it by setting `LOCAL_IDENTITIES = False` in `indico.conf`.

Warning: We assume that emails received from SSO are already validated. If this is not the case, make sure to disable `trusted_email` which will require email validation in Indico when logging in for the first time. Otherwise people could take over the account of someone else by using their email address!

Note: The example config is rather simple and only accesses data from SSO during login. This is not sufficient for advanced features such as automatic synchronization of names, affiliations and phone numbers or using centrally managed groups. To use these features, you need to use e.g. the LDAP identity provider and use the information received via SSO to retrieve the user details from LDAP. If you need assistance with this, feel free to ask us on IRC (#indico @ Freenode) or via e-mail (indico-team@cern.ch).

Note: Please note that you **must** use Apache if you intend to use SSO using Shibboleth/SAML/ADFS. If that's not the case because you do not use SSO at all or use e.g. OAuth, we recommend using nginx.

Debian / Ubuntu

Except for minor differences, this guide applies to both Debian and Ubuntu. It has been tested with Debian 8 (Jessie), Debian 9 (Stretch) and Ubuntu 16.04 (Xenial).

nginx

Note: Please note that you **must** use Apache if you intend to use SSO using Shibboleth/SAML/ADFS. If that's not the case because you do not use SSO at all or use e.g. OAuth, we recommend using nginx.

1. Install Packages

PostgreSQL and nginx are installed from their upstream repos to get much more recent versions.

```
echo "deb http://apt.postgresql.org/pub/repos/apt/ $(lsb_release -cs)-pgdg main" > /
↪etc/apt/sources.list.d/pgdg.list
echo "deb http://nginx.org/packages/$(lsb_release -is | tr '[:upper:]' '[:lower:]')/
↪$(lsb_release -cs) nginx" > /etc/apt/sources.list.d/nginx.list
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | apt-key add -
wget --quiet -O - https://nginx.org/keys/nginx_signing.key | apt-key add -
apt update
apt install -y --install-recommends postgresql-9.6 libpq-dev nginx python-dev python-
↪virtualenv libxslt1-dev libxml2-dev libffi-dev libpcre3-dev libyaml-dev build-
↪essential redis-server uwsgi uwsgi-plugin-python
```

If you use Debian, run this command:

```
apt install -y libjpeg62-turbo-dev
```

If you use Ubuntu, run this instead:

```
apt install -y libjpeg-turbo8-dev zlib1g-dev
```

2. Create a Database

We create a user and database for indico and enable the necessary Postgres extensions (which can only be done by the Postgres superuser)

```
su - postgres -c 'createuser indico'
su - postgres -c 'createdb -O indico indico'
su - postgres -c 'psql indico -c "CREATE EXTENSION unaccent; CREATE EXTENSION pg_trgm;
↪"'
```

Warning: Do not forget to setup a cronjob that creates regular database backups once you start using Indico in production!

3. Configure uWSGI & nginx

The default uWSGI and nginx configuration files should work fine in most cases.

```
ln -s /etc/uwsgi/apps-available/indico.ini /etc/uwsgi/apps-enabled/indico.ini
cat > /etc/uwsgi/apps-available/indico.ini <<'EOF'
[uwsgi]
uid = indico
gid = nginx
umask = 027

processes = 4
enable-threads = true
chmod-socket = 770
socket = /opt/indico/web/uwsgi.sock
stats = /opt/indico/web/uwsgi-stats.sock
protocol = uwsgi

master = true
auto-procname = true
```

```
procname-prefix-spaced = indico
disable-logging = true

plugin = python
single-interpreter = true

touch-reload = /opt/indico/web/indico.wsgi
wsgi-file = /opt/indico/web/indico.wsgi
virtualenv = /opt/indico/.venv

vacuum = true
buffer-size = 20480
memory-report = true
max-requests = 2500
harakiri = 900
harakiri-verbose = true
reload-on-rss = 2048
evil-reload-on-rss = 8192
EOF
```

Note: Replace `YOURHOSTNAME` in the next file with the hostname on which your Indico instance should be available, e.g. `indico.yourdomain.com`

```
cat > /etc/nginx/conf.d/indico.conf <<'EOF'
server {
  listen 80;
  listen [::]:80;
  server_name YOURHOSTNAME;
  return 301 https://$server_name$request_uri;
}

server {
  listen       *:443 ssl http2;
  listen       [::]:443 ssl http2 default ipv6only=on;
  server_name  YOURHOSTNAME;

  ssl on;

  ssl_certificate           /etc/ssl/indico/indico.crt;
  ssl_certificate_key       /etc/ssl/indico/indico.key;
  ssl_session_cache         shared:SSL:10m;
  ssl_session_timeout       5m;
  ssl_protocols             TLSv1 TLSv1.1 TLSv1.2;
  ssl_ciphers               ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-
↪POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-
↪AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-
↪AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-
↪AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-
↪SHA384:ECDHE-ECDSA-AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-
↪AES128-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-ECDSA-DES-CBC3-SHA:ECDHE-
↪RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-
↪SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:DES-CBC3-SHA:!DSS;
  ssl_prefer_server_ciphers on;

  access_log            /opt/indico/log/nginx/access.log combined;
  error_log             /opt/indico/log/nginx/error.log;
```

```
  location /.xsf/indico/ {
    internal;
    alias /opt/indico/;
  }

  location ~ ^/static/assets/(core|(?:plugin|theme)-[^/]+)/(.*)$ {
    alias /opt/indico/assets/$1/$2;
    access_log off;
  }

  location ~ ^/(css|images|js|static(?!/plugins|/assets|/themes|/custom))(/.*)$ {
    alias /opt/indico/web/htdocs/$1$2;
    access_log off;
  }

  location /robots.txt {
    alias /opt/indico/web/htdocs/robots.txt;
    access_log off;
  }

  location / {
    root  /var/empty/nginx;
    include /etc/nginx/uwsgi_params;
    uwsgi_pass unix:/opt/indico/web/uwsgi.sock;
    uwsgi_param UWSGI_SCHEME $scheme;
    uwsgi_read_timeout 15m;
    uwsgi_buffers 32 32k;
    uwsgi_busy_buffers_size 128k;
    uwsgi_hide_header X-Sendfile;
    client_max_body_size 1G;
  }
}
EOF
```

4. Create an SSL Certificate

First, create the folders for the certificate/key and set restrictive permissions on them:

```
mkdir /etc/ssl/indico
chown root:root /etc/ssl/indico/
chmod 700 /etc/ssl/indico
```

If you are just trying out Indico you can simply use a self-signed certificate (your browser will show a warning which you will have to confirm when accessing your Indico instance for the first time).

Note: Do not forget to replace `YOURHOSTNAME` with the same value you used above

```
openssl req -x509 -nodes -newkey rsa:4096 -subj /CN=YOURHOSTNAME -keyout /etc/ssl/
↪indico/indico.key -out /etc/ssl/indico/indico.crt
```

While a self-signed certificate works for testing, it is not suitable for a production system. You can either buy a certificate from any commercial certification authority or get a free one from Let's Encrypt.

Note: There's an optional step later in this guide to get a certificate from Let's Encrypt. We can't do it right now since the nginx config references a directory yet to be created, which prevents nginx from starting.

5. Install Indico

Celery runs as a background daemon. Add a systemd unit file for it:

```
cat > /etc/systemd/system/indico-celery.service <<'EOF'
[Unit]
Description=Indico Celery
After=network.target

[Service]
ExecStart=/opt/indico/.venv/bin/indico celery worker -B
Restart=always
SyslogIdentifier=indico-celery
User=indico
Group=nginx
UMask=0027
Type=simple

[Install]
WantedBy=multi-user.target
EOF
systemctl daemon-reload
```

Now create a user that will be used to run Indico and switch to it:

```
useradd -rm -g nginx -d /opt/indico -s /bin/bash indico
su - indico
```

You are now ready to install Indico:

```
virtualenv ~/.venv
source ~/.venv/bin/activate
pip install -U pip setuptools
pip install --pre indico
```

6. Configure Indico

Once Indico is installed, you can run the configuration wizard. You can keep the defaults for most options, but make sure to use `https://YOURHOSTNAME` when prompted for the Indico URL. Also specify valid email addresses when asked and enter a valid SMTP server Indico can use to send emails. When asked for the default timezone make sure this is the main time zone used in your Indico instance.

```
indico setup wizard
```

Now finish setting up the directory structure and permissions:

```
mkdir ~/log/nginx
chmod go-rwx ~/* ~/.[^.]*
chmod 710 ~/ ~/archive ~/assets ~/cache ~/log ~/tmp
chmod 750 ~/web ~/.venv
```

```
chmod g+w ~/log/nginx
echo -e "\nSTATIC_FILE_METHOD = ('xaccelredirect', {'/opt/indico': '/.xsf/indico'})" >
↪> ~/etc/indico.conf
```

7. Create database schema

Finally, you can create the database schema and switch back to *root*:

```
indico db prepare
exit
```

8. Launch Indico

You can now start Indico and set it up to start automatically when the server is rebooted:

```
systemctl restart uwsgi.service nginx.service indico-celery.service
systemctl enable uwsgi.service nginx.service postgresql.service redis-server.service␣
↪indico-celery.service
```

9. Optional: Get a Certificate from Let's Encrypt

Note: You need to use at least Debian 9 (Stretch) to use certbot. If you are still using Debian 8 (Jessie), consider updating or install certbot from backports.

If you use Ubuntu, install the certbot PPA:

```
apt install -y software-properties-common
add-apt-repository -y ppa:certbot/certbot
apt update
```

To avoid ugly SSL warnings in your browsers, the easiest option is to get a free certificate from Let's Encrypt. We also enable the cronjob to renew it automatically:

```
apt install -y python-certbot-nginx
certbot --nginx --rsa-key-size 4096 --no-redirect --staple-ocsp -d YOURHOSTNAME
rm -rf /etc/ssl/indico
systemctl start certbot.timer
systemctl enable certbot.timer
```

10. Create an Indico user

Access `https://YOURHOSTNAME` in your browser and follow the steps displayed there to create your initial user.

Apache

1. Install Packages

PostgreSQL is installed from its upstream repos to get a much more recent version.

```
echo "deb http://apt.postgresql.org/pub/repos/apt/ $(lsb_release -cs)-pgdg main" > /
↪etc/apt/sources.list.d/pgdg.list
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | apt-key add -
apt update
apt install -y --install-recommends postgresql-9.6 libpq-dev apache2 libapache2-mod-
↪proxy-uwsgi libapache2-mod-xsendfile python-dev python-virtualenv libxslt1-dev␣
↪libxml2-dev libffi-dev libpcre3-dev libyaml-dev build-essential redis-server uwsgi␣
↪uwsgi-plugin-python
```

If you use Debian, run this command:

```
apt install -y libjpeg62-turbo-dev
```

If you use Ubuntu, run this instead:

```
apt install -y libjpeg-turbo8-dev zlib1g-dev
```

2. Create a Database

We create a user and database for indico and enable the necessary Postgres extensions (which can only be done by the Postgres superuser)

```
su - postgres -c 'createuser indico'
su - postgres -c 'createdb -O indico indico'
su - postgres -c 'psql indico -c "CREATE EXTENSION unaccent; CREATE EXTENSION pg_trgm;
↪"'
```

Warning: Do not forget to setup a cronjob that creates regular database backups once you start using Indico in production!

3. Configure uWSGI & Apache

The default uWSGI and Apache configuration files should work fine in most cases.

```
ln -s /etc/uwsgi/apps-available/indico.ini /etc/uwsgi/apps-enabled/indico.ini
cat > /etc/uwsgi/apps-available/indico.ini <<'EOF'
[uwsgi]
uid = indico
gid = www-data
umask = 027

processes = 4
enable-threads = true
socket = 127.0.0.1:8008
stats = /opt/indico/web/uwsgi-stats.sock
protocol = uwsgi

master = true
```

```
auto-procname = true
procname-prefix-spaced = indico
disable-logging = true

plugin = python
single-interpreter = true

touch-reload = /opt/indico/web/indico.wsgi
wsgi-file = /opt/indico/web/indico.wsgi
virtualenv = /opt/indico/.venv

vacuum = true
buffer-size = 20480
memory-report = true
max-requests = 2500
harakiri = 900
harakiri-verbose = true
reload-on-rss = 2048
evil-reload-on-rss = 8192
EOF
```

Note: Replace `YOURHOSTNAME` in the next files with the hostname on which your Indico instance should be available, e.g. `indico.yourdomain.com`

```
cat > /etc/apache2/sites-available/indico-sslredir.conf <<'EOF'
<VirtualHost *:80>
    ServerName YOURHOSTNAME
    RewriteEngine On
    RewriteRule ^(.*)$ https://%{HTTP_HOST}$1 [R=301,L]
</VirtualHost>
EOF

cat > /etc/apache2/sites-available/indico.conf <<'EOF'
<VirtualHost *:443>
    ServerName YOURHOSTNAME
    DocumentRoot "/var/empty/apache"

    SSLEngine             on
    SSLCertificateFile    /etc/ssl/indico/indico.crt
    SSLCertificateKeyFile /etc/ssl/indico/indico.key
    SSLProtocol           all -SSLv2 -SSLv3
    SSLCipherSuite        ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-
↪POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-
↪AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-
↪AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-
↪AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-
↪SHA384:ECDHE-ECDSA-AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-
↪AES128-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-ECDSA-DES-CBC3-SHA:ECDHE-
↪RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-
↪SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:DES-CBC3-SHA:!DSS
    SSLHonorCipherOrder   on

    XSendFile on
    XSendFilePath /opt/indico
    CustomLog /opt/indico/log/apache/access.log combined
    ErrorLog /opt/indico/log/apache/error.log
```

```
    LogLevel error
    ServerSignature Off

    AliasMatch "^/static/assets/(core|(?:plugin|theme)-[^/]+)/(.*)$" "/opt/indico/
↪assets/$1/$2"
    AliasMatch "^/(css|images|js|static(?!/plugins|/assets|/themes|/custom))(/.*)$" "/
↪opt/indico/web/htdocs/$1$2"
    Alias /robots.txt /opt/indico/web/htdocs/robots.txt

    SetEnv UWSGI_SCHEME https
    ProxyPass / uwsgi://127.0.0.1:8008/

    <Directory /opt/indico>
        AllowOverride None
        Require all granted
    </Directory>
</VirtualHost>
EOF
```

Now enable the necessary modules and the indico site in apache:

```
a2enmod proxy_uwsgi rewrite ssl xsendfile
a2dissite 000-default
a2ensite indico indico-sslredir
```

4. Create an SSL Certificate

First, create the folders for the certificate/key and set restrictive permissions on them:

```
mkdir /etc/ssl/indico
chown root:root /etc/ssl/indico/
chmod 700 /etc/ssl/indico
```

If you are just trying out Indico you can simply use a self-signed certificate (your browser will show a warning which you will have to confirm when accessing your Indico instance for the first time).

Note: Do not forget to replace `YOURHOSTNAME` with the same value you used above

```
openssl req -x509 -nodes -newkey rsa:4096 -subj /CN=YOURHOSTNAME -keyout /etc/ssl/
↪indico/indico.key -out /etc/ssl/indico/indico.crt
```

While a self-signed certificate works for testing, it is not suitable for a production system. You can either buy a certificate from any commercial certification authority or get a free one from Let's Encrypt.

Note: There's an optional step later in this guide to get a certificate from Let's Encrypt. We can't do it right now since the Apache config references a directory yet to be created, which prevents Apache from starting.

5. Install Indico

Celery runs as a background daemon. Add a systemd unit file for it:

```
cat > /etc/systemd/system/indico-celery.service <<'EOF'
[Unit]
Description=Indico Celery
After=network.target

[Service]
ExecStart=/opt/indico/.venv/bin/indico celery worker -B
Restart=always
SyslogIdentifier=indico-celery
User=indico
Group=www-data
UMask=0027
Type=simple

[Install]
WantedBy=multi-user.target
EOF
systemctl daemon-reload
```

Now create a user that will be used to run Indico and switch to it:

```
useradd -rm -g www-data -d /opt/indico -s /bin/bash indico
su - indico
```

You are now ready to install Indico:

```
virtualenv ~/.venv
source ~/.venv/bin/activate
pip install -U pip setuptools
pip install --pre indico
```

6. Configure Indico

Once Indico is installed, you can run the configuration wizard. You can keep the defaults for most options, but make sure to use `https://YOURHOSTNAME` when prompted for the Indico URL. Also specify valid email addresses when asked and enter a valid SMTP server Indico can use to send emails. When asked for the default timezone make sure this is the main time zone used in your Indico instance.

```
indico setup wizard
```

Now finish setting up the directory structure and permissions:

```
mkdir ~/log/apache
chmod go-rwx ~/* ~/.[^.]*
chmod 710 ~/ ~/archive ~/assets ~/cache ~/log ~/tmp
chmod 750 ~/web ~/.venv
chmod g+w ~/log/apache
echo -e "\nSTATIC_FILE_METHOD = 'xsendfile'" >> ~/etc/indico.conf
```

7. Create database schema

Finally, you can create the database schema and switch back to *root*:

```
indico db prepare
exit
```

8. Launch Indico

You can now start Indico and set it up to start automatically when the server is rebooted:

```
systemctl restart uwsgi.service apache2.service indico-celery.service
systemctl enable uwsgi.service apache2.service postgresql.service redis-server.
↪service indico-celery.service
```

9. Optional: Get a Certificate from Let's Encrypt

Note: You need to use at least Debian 9 (Stretch) to use certbot. If you are still using Debian 8 (Jessie), consider updating or install certbot from backports.

If you use Ubuntu, install the certbot PPA:

```
apt install -y software-properties-common
add-apt-repository -y ppa:certbot/certbot
apt update
```

To avoid ugly SSL warnings in your browsers, the easiest option is to get a free certificate from Let's Encrypt. We also enable the cronjob to renew it automatically:

```
apt install -y python-certbot-apache
certbot --apache --rsa-key-size 4096 --no-redirect --staple-ocsp -d YOURHOSTNAME
rm -rf /etc/ssl/indico
systemctl start certbot.timer
systemctl enable certbot.timer
```

10. Create an Indico user

Access `https://YOURHOSTNAME` in your browser and follow the steps displayed there to create your initial user.

Optional: Shibboleth

If your organization uses Shibboleth/SAML-based SSO, follow these steps to use it in Indico:

1. Install Shibboleth

```
apt install -y libapache2-mod-shib2
a2enmod shib2
```

2. Configure Shibboleth

This is outside the scope of this documentation and depends on your environment (Shibboleth, SAML, ADFS, etc). Please contact whoever runs your SSO infrastructure if you need assistance.

3. Enable Shibboleth in Apache

Add the following code to your `/etc/apache2/sites-available/indico.conf` right before the `AliasMatch` lines:

```
<LocationMatch "^(/Shibboleth\.sso|/login/shib-sso/shibboleth)">
    AuthType shibboleth
    ShibRequestSetting requireSession 1
    ShibExportAssertion Off
    Require valid-user
</LocationMatch>
```

4. Enable Shibboleth in Indico

Add the following code to your `/opt/indico/etc/indico.conf`:

```
# SSO
AUTH_PROVIDERS = {
    'shib-sso': {
        'type': 'shibboleth',
        'title': 'SSO',
        'attrs_prefix': 'ADFS_',
        'callback_uri': '/login/shib-sso/shibboleth',
        # 'logout_uri': 'https://login.yourcompany.tld/logout'
    }
}
IDENTITY_PROVIDERS = {
    'shib-sso': {
        'type': 'shibboleth',
        'title': 'SSO',
        'identifier_field': 'ADFS_LOGIN',
        'mapping': {
            'affiliation': 'ADFS_HOMEINSTITUTE',
            'first_name': 'ADFS_FIRSTNAME',
            'last_name': 'ADFS_LASTNAME',
            'email': 'ADFS_EMAIL',
            'phone': 'ADFS_PHONENUMBER'
        },
        'trusted_email': True
    }
}
```

The values for `attrs_prefix`, `mapping` and `identifier_field` may be different in your environment. Uncomment and set `logout_uri` if your SSO infrastructure provides a logout URL (usually used to log you out from all applications).

If you only want to use SSO, without allowing people to login locally using username/password, disable it by setting `LOCAL_IDENTITIES = False` in `indico.conf`.

Warning: We assume that emails received from SSO are already validated. If this is not the case, make sure to disable `trusted_email` which will require email validation in Indico when logging in for the first time. Otherwise people could take over the account of someone else by using their email address!

Note: The example config is rather simple and only accesses data from SSO during login. This is not sufficient for advanced features such as automatic synchronization of names, affiliations and phone numbers or using centrally managed groups. To use these features, you need to use e.g. the LDAP identity provider and use the information received via SSO to retrieve the user details from LDAP. If you need assistance with this, feel free to ask us on IRC (#indico @ Freenode) or via e-mail (indico-team@cern.ch).

Note: Please note that you **must** use Apache if you intend to use SSO using Shibboleth/SAML/ADFS. If that's not the case because you do not use SSO at all or use e.g. OAuth, we recommend using nginx.

Upgrade

It is important to keep your Indico instance up to date to have the latest bug fixes and features. Upgrading can be done with almost no user-facing downtime.

Warning: When upgrading a production system it is highly recommended to create a database backup before starting.

First of all, stop the Celery worker. To do so, run this as *root*:

```
systemctl stop indico-celery.service
```

Now switch to the *indico* user and activate the virtualenv:

```
su - indico
source ~/.venv/bin/activate
```

You are now ready to install the latest version of Indico:

```
pip install -U --pre indico
```

Some versions may include database schema upgrades. Make sure to perform them immediately after upgrading. If there are no schema changes, the command will simply do nothing.

```
indico db upgrade
```

Note: Some database structure changes require an *exclusive lock* on some tables in the database. Unless you have very high activity on your instance, this lock can be acquired quickly, but if the upgrade command seems to hang for more than a few seconds, you can restart uWSGI by running `systemctl restart uwsgi.service` as *root* (in a separate shell, i.e. don't abort the upgrade command!) which will ensure nothing is accessing Indico for a moment.

Unless you just restarted uWSGI, it is now time to reload it so the new version is actually used:

```
touch ~/web/indico.wsgi
```

Also start the Celery worker again (once again, as *root*):

```
systemctl start indico-celery.service
```

Upgrading from 1.9.11 to 2.0

Make sure that you have the latest 1.9.11 version installed and that you used `indico db upgrade` to have the most recent database structure.

To upgrade to 2.0, follow the upgrade instructions above. After successfully running the upgrade, use `indico db reset_alembic` to clear pre-2.0 database migration information, since all the old migration steps from the 1.9.x version line have been removed in 2.0.

Upgrade Indico from 1.2

Warning: **ATTENTION: This process is not yet fully tested, use it at your own risk!** We are currently working with some of our users running Indico 1.2 in order to ensure migration works reliably and without any data loss. If you'd like to help, please let us know!

If you're running a version that is lower than 2.0, you will have to run a special migration command provided by the `indico-migrate` package. This document will guide you over the steps needed to perform the upgrade.

Prerequisites

In order to migrate to version 2.0 of Indico you will first of all need to make sure you have **at least version 1.2** of Indico installed. Migration of databases using earlier versions will either **fail** or very likely result in **data loss**. So, please make sure that you are **on 1.2.x** before migrating.

Warning: If you are running a version of the experimental (thus unsupported) **1.9.x branch**, you will have to perform a **step-by-step migration**. We hope that, as advised, no-one upgraded to intermediate 1.9.x releases. If you did and need help with it, please **ping us on IRC**.

Backing up ZODB

The migration script doesn't write to the ZODB, but we still recommend that you **make a backup** just in case:

```
repozo -B -F -r <some-place-safe> -f <indico-db-dir>/Data.fs --verbose
```

You should replace `<some-place-safe>` with the directory in your filesystem where you want to keep the backup. As for `<indico-db-dir>`, that's the directory where the database file is kept. That should be `/opt/indico/db` in most Indico installations.

Make sure that backup files have been created (you should have an `*.index` and an `*.fs` file).

Now, let's shut down the ZEO daemon:

```
zdaemon -C /opt/indico/etc/zdctl.conf stop
```

Double check that the daemon is not running:

```
zdaemon -C /opt/indico/etc/zdctl.conf status
```

Moving legacy data

Indico 2.0 will use a directory structure that is similar to Indico 1.x, so first of all you will need to rename the old tree:

```
mv /opt/indico /opt/indico-legacy
```

Warning: After the migration is done, **do not** delete the `/opt/indico-legacy` directory without first moving the `archive` dir elsewhere. Please read the full guide until the end.

Installing Indico 2.0

The first step should be to have a working Indico 2.0 setup. In order to do that, you should follow the regular Indico 2.x installation instructions up to the "Configure Indico" step. We provide below direct links to the relevant sections of the installation guides.

On a **Debian/Ubuntu** system:

nginx	Apache
1. Install Packages	*1. Install Packages*
2. Create a Database	*2. Create a Database*
3. Configure uWSGI & nginx	*3. Configure uWSGI & Apache*
4. Create an SSL Certificate	*4. Create an SSL Certificate*
5. Install Indico	*5. Install Indico*
6. Configure Indico	*6. Configure Indico*

On a **CentOS7-based system**:

nginx	Apache
1. Enable EPEL	*1. Enable EPEL*
2. Install Packages	*2. Install Packages*
3. Create a Database	*3. Create a Database*
4. Configure uWSGI & nginx	*4. Configure uWSGI & Apache*
5. Create an SSL Certificate	*5. Create an SSL Certificate*
6. Configure SELinux	*6. Configure SELinux*
7. Install Indico	*7. Install Indico*
8. Configure Indico	*8. Configure Indico*

Configuration Wizard

You will then need to run the Configuration Wizard, following the normal installation guide (Debian/Ubuntu or CentOS). When the wizard asks you about the **"Old archvive dir"**, make sure to set it to the archive dir in the `indico-legacy` directory.

```
...
If you are upgrading from Indico 1.2, please specify the path to the
ArchiveDir of the old indico version.  Leave this empty if you are not
upgrading.
Old archive dir: /opt/indico-legacy/archive
...
```

Running `indico-migrate`

First of all, make sure that you are using the **user** and **virtualenv** created using the step **"Install Indico"** and that the legacy dir is owned by this **user**:

```
chown -R indico /opt/indico-legacy
su - indico
source ~/.venv/bin/activate
```

You should then install the package using:

```
pip install indico-migrate
```

`indico-migrate` requires a series of parameters that have to be tuned according to your current setup. We now provide a list of values that should work in most standard Indico installations. However, please **carefully read** the documentation of the indico-migrate command, to make sure there are no option conflicts with your setup.

Most frequenty, `indico-migrate postgresql:///indico file:///opt/indico-legacy/db/Data.fs` will work, followed by the following parameters:

- `--archive-dir /opt/indico-legacy/archive`
- `--storage-backend legacy`
- `--default-email default@<organization-hostname>`
- `--default-currency EUR`
- `--symlink-target ~/archive/legacy_symlinks/`
- `--symlink-backend legacy-symlinks`
- `--migrate-broken-events` (optional - use it if you want to migrate events that don't belong to any category in v1.2. If any such events exist, the will be added to a new category named *Lost & Found*.

(don't forget to replace `<organization-hostname>` with the e-mail hostname of your organization)

An example:

```
indico-migrate postgresql:///indico file:///opt/indico-legacy/db/Data.fs --archive-
↪dir /opt/indico-legacy/archive --storage-backend legacy --default-email␣
↪default@acme.example.com --default-currency EUR --symlink-target ~/archive/legacy_
↪symlinks/ --symlink-backend legacy-symlinks --migrate-broken-events
```

Note: If for some reason the migration fails, `indico-migrate` will ask you whether you would like to post an error report on a public pastebin (Gist). The link will not be advertised and only the log information that was shown on screen (plus the exception traceback that was printed) will be included. If you are not comfortable with letting `indico-migrate` post this on a public pastebin, you can always send us your `migration.log` file (which gets generated automatically).

Post-migration work

After the migration is done you may need to apply some adjustments in your `indico.conf`. You may want to read our guide on how to configure an Identity/Authentication provider.

We really recommend as well that you move your old Indico archive (`/opt/indico-legacy/archive`) inside your new Indico directory:

```
mv /opt/indico-legacy/archive /opt/indico/legacy-archive
```

The legacy archive will remain **read-only**. You should update your `indico.conf` (`STORAGE_BACKENDS` option) to reflect the new path:

```
STORAGE_BACKENDS = {
    # ...
    'legacy': 'fs-readonly:/opt/indico/legacy-archive'
    # ...
}
```

Finishing up

You can now proceed with the remaining installation steps:

On a **Debian/Ubuntu** system:

nginx	Apache
8. Launch Indico	*8. Launch Indico*
9. Optional: Get a Certificate from Let's Encrypt	*9. Optional: Get a Certificate from Let's Encrypt*
10. Create an Indico user	*10. Create an Indico user*

On a **CentOS7-based system**:

nginx	Apache
10. Launch Indico	*10. Launch Indico*
11. Open the Firewall	*11. Open the Firewall*
12. Optional: Get a Certificate from Let's Encrypt	*12. Optional: Get a Certificate from Let's Encrypt*
13. Create an Indico user	*13. Create an Indico user*

Sanitizing HTML

Indico 2.0 uses Markdown for the descriptions of contributions and categories. Contribution descriptions that previously contained HTML will still work, but new ones will only support Markdown syntax (including basic HTML). As for the descriptions of categories, they are interpreted as Markdown as of version 2.0, which means that some existing data may be broken. In order to make the lives of users who are migrating easier, we have included with `indico-migrate` a command that automatically performs the migration of Category descriptions to Markdown.

First of all, let's see what would be the impact of running the command:

```
indico-html-sanitize --dry-run -v -l log.html category_descriptions
```

By opening `log.html` you will be able to check if there are any special cases that will need manual intervention. If you're happy with the changes, you can just choose to save them:

```
indico-html-sanitize category_descriptions
```

Removing old data

Even if you're sure the migration succeeded and all data was kept, please keep around the backup of your ZODB you made at the beginning of this guide. **After** and **only after** having **moved the legacy archive** to the new Indico dir and stored a **backup of your ZODB** in a safe place, you can proceed to delete the old `/opt/indico` directory:

```
rm -rf /opt/indico-legacy
```

Installation guide (development)

Todo

Write guide

CHAPTER 2

Server administration

Indico uses Celery for task management and scheduling and the Livesync Flask plugin for search integration. This part of the documentation explains how to set them up and run automatically.

Server administration

Indico uses Celery for task management and scheduling and the Livesync Flask plugin for search integration. This part of the documentation explains how to set them up and run automatically.

Celery

Todo

Complete page

Livesync

Todo

Complete page

CHAPTER 3

Plugins

Indico can be extended through plugins, standalone packages of code that do not require any modifications to the Indico core itself. A plugin can perform something very simple such as adding a new command to the Indico CLI to more complex functionalities like introducing new payment methods, chat integration, etc. We suggest that you first have a look at Getting started and then head over to the more advance topics in the table of contents.

Extending Indico with plugins

Indico can be extended through plugins, standalone packages of code that do not require any modifications to the Indico core itself. A plugin can perform something very simple such as adding a new command to the Indico CLI to more complex functionalities like introducing new payment methods, chat integration, etc. We suggest that you first have a look at Getting started and then head over to the more advance topics in the table of contents.

Getting started with Indico plugins

Todo

Write a **REAL**, simple example of a plugin. Include link to Github repo.

Example plugin

The following is a minimal plugin that makes use of all capababilites of the plugin API. The **display name** of the plugin is defined by the first line of the docstring and the **description** by the rest of it. The plugin may override signal handlers to hook into Indico and aditionally run any initialization needed. For example, it will add some command to Indico CLI, extend the shell context and register some assets. Also, *init* is used to inject CSS and JS bundles outside of the plugin scope.

```
class ExamplePlugin(IndicoPlugin):
    """Example Plugin

    An example plugin that demonstrates the capabilities of the new Indico plugin␣
↪system.
    """

    settings_form = SettingsForm

    def init(self):
        super(ExamplePlugin, self).init()
        self.inject_css('global_css')
        self.inject_js('global_js')

    def get_blueprints(self):
        return blueprint

    def add_cli_command(self, manager):
        @manager.command
        @with_plugin_context(self)
        def example():
            """Example command from example plugin"""
            print 'example plugin says hi', current_plugin
            if self.settings.get('show_message'):
                print self.settings.get('dummy_message')

    def extend_shell_context(self, add_to_context):
        add_to_context('bar', name='foo', doc='foobar from example plugin', color=
↪'magenta!')

    def register_assets(self):
        self.register_js_bundle('example_js', 'js/example.js')
        self.register_js_bundle('global_js', 'js/global.js')
        self.register_css_bundle('example_css', 'css/example.scss')
        self.register_css_bundle('global_css', 'css/global.scss')
```

The plugin can specify its settings via a `IndicoForm`:

```
class SettingsForm(IndicoForm):
    dummy_message = StringField('Dummy Message')
    show_message = BooleanField('Show Message')
```

The plugin can also specify request handlers and templates. Templates will be loaded from a *templates* folder within your plugin folder. Your plugin can even load templates from other modules by prefixing the name of the template *'other_plugin:example'* with *render_template()*.

```
class WPExample(WPMainBase):
    def _getBody(self, params):
        return render_plugin_template('example.html', **params)


class RHExample(RH):
    def _process(self):
        return WPExample(self, foo=u'bar').display()


class RHTest(RH):
    def _process(self):
```

```
        return render_plugin_template('test.html')


blueprint = IndicoPluginBlueprint('example', __name__)
blueprint.add_url_rule('/example', 'example', view_func=RHExample)
blueprint.add_url_rule('/example/x', 'example', view_func=RHExample)
blueprint.add_url_rule('/test', 'test', view_func=RHTest)
```

Plugin API reference

Indico's plugin system allows you to extend indico with additional modules which can be installed separately and do not require any modifications to the indico core itself.

class indico.core.plugins.**IndicoPlugin**(*plugin_engine*, *app*)

Bases: flask_pluginengine.plugin.Plugin

Base class for an Indico plugin

All your plugins need to inherit from this class. It extends the *Plugin* class from Flask-PluginEngine with useful indico-specific functionality that makes it easier to write custom plugins.

When creating your plugin, the class-level docstring is used to generate the friendly name and description of a plugin. Its first line becomes the name while everything else goes into the description.

This class provides methods for some of the more common hooks Indico provides. Additional signals are defined in signals and can be connected to custom functions using connect().

acl_event_settings = frozenset([])
A set containing the names of event-specific settings which store ACLs

acl_settings = frozenset([])
A set containing the names of settings which store ACLs

category = None
The group category that the plugin belongs to

configurable = False
If the plugin should link to a details/config page in the admin interface

default_event_settings = {}
A dictionary containing default values for event-specific settings

default_settings = {}
A dictionary containing default values for settings

default_user_settings = {}
A dictionary containing default values for user-specific settings

event_settings
classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: def f(cls, arg1, arg2, ...): ... f = classmethod(f)

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

event_settings_converters = {}
A dict containing custom converters for event-specific settings

get_blueprints()
Return blueprints to be registered on the application

A single blueprint can be returned directly, for multiple blueprint you need to yield them or return an iterable.

get_vars_js()
Return a dictionary with variables to be added to vars.js file

init()
Called when the plugin is being loaded/initialized.

If you want to run custom initialization code, this is the method to override. Make sure to call the base method or the other overridable methods in this class will not be called anymore.

inject_css(*name*, *view_class=None*, *subclasses=True*, *condition=None*)
Injects a CSS bundle into Indico's pages

Parameters

- **name** – Name of the bundle
- **view_class** – If a WP class is specified, only inject it into pages using that class
- **subclasses** – also inject into subclasses of *view_class*
- **condition** – a callable to determine whether to inject or not. only called, when the view_class criterion matches

inject_js(*name*, *view_class=None*, *subclasses=True*, *condition=None*)
Injects a JS bundle into Indico's pages

Parameters

- **name** – Name of the bundle
- **view_class** – If a WP class is specified, only inject it into pages using that class
- **subclasses** – also inject into subclasses of *view_class*
- **condition** – a callable to determine whether to inject or not. only called, when the view_class criterion matches

inject_vars_js()
Returns a string that will define variables for the plugin in the vars.js file

register_assets()
Add assets to the plugin's webassets environment.

In most cases the whole method can consist of calls to *register_js_bundle()* and *register_css_bundle()*.

register_css_bundle(*name*, **files*)
Registers an SCSS bundle in the plugin's webassets environment

register_js_bundle(*name*, **files*)
Registers a JS bundle in the plugin's webassets environment

settings
classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: def f(cls, arg1, arg2, ...): ... f = classmethod(f)

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

settings_converters = {}
A dict containing custom converters for settings

settings_form = None
WTForm for the plugin's settings (requires *configurable=True*). All fields must return JSON-serializable types.

settings_form_field_opts = {}
A dictionary which can contain the kwargs for a specific field in the *settings_form*.

strict_settings = True
If *settings*, *event_settings* and *user_settings* should use strict mode, i.e. only allow keys in *default_settings*, *default_event_settings* or *default_user_settings* (or the related *acl_settings* sets). This should not be disabled in most cases; if you need to store arbitrary keys, consider storing a dict inside a single top-level setting.

template_hook(*name*, *receiver*, *priority=50*, *markup=True*)
Registers a function to be called when a template hook is invoked.

For details see :func:~'indico.web.flask.templating.register_template_hook'

translation_domain
Return the domain for this plugin's translation_path

translation_path
Return translation files to be used by the plugin. By default, get <root_path>/translations, unless it does not exist

user_settings
classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: def f(cls, arg1, arg2, ...): ... f = classmethod(f)

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

user_settings_converters = {}
A dict containing custom converters for user-specific settings

class indico.core.plugins.**IndicoPluginBlueprint**(*name*, **args*, ***kwargs*)
Bases: flask_pluginengine.mixins.PluginBlueprintMixin, indico.web.flask.wrappers.IndicoBlueprint

The Blueprint class all plugins need to use.

It contains the necessary logic to run the blueprint's view functions inside the correct plugin context and to make the static folder work.

indico.core.plugins.**get_plugin_template_module**(*template_name*, ***context*)
Like `get_template_module()`, but using plugin templates

indico.core.plugins.**include_plugin_css_assets**(*bundle_name*)
Jinja template function to generate HTML tags for a plugin CSS asset bundle.

indico.core.plugins.**include_plugin_js_assets**(*bundle_name*)
Jinja template function to generate HTML tags for a plugin JS asset bundle.

indico.core.plugins.**plugin_url_rule_to_js**(*endpoint*)
Like `url_rule_to_js()` but prepending plugin name prefix to the endpoint

indico.core.plugins.**url_for_plugin**(*endpoint*, **targets*, ***values*)
Like `url_for()` but prepending `'plugin_'` to the blueprint name.

Hooking into Indico using Signals

Contents

Signals allow you to hook into certain parts of Indico without adding any code to the core (which is something a plugin can and should not do). Each signal has a *sender* which can be any object (depending on the signal) and possibly some keyword arguments. Some signals also make use of their return value or even require one. Check the documentation of each signal on how it's used.

To avoid breakage with newer versions of Indico, it is highly advised to always accept extra `**kwargs` in your signal receiver. For example, a receiver function could look like this:

```
def receiver(sender, something, **kwargs):
    do_stuff_with(something)
```

indico.core.signals

indico.core.signals.**add_form_fields**
Lets you add extra fields to a form. The *sender* is the form class and should always be specified when subscribing to this signal.

The signal handler should return one or more `'name', Field` tuples. Each field will be added to the form as `ext__<name>` and is automatically excluded from the form's *data* property and its *populate_obj* method.

To actually process the data, you can use e.g. the *form_validated* signal and then store it in *flask.g* until another signal informs you that the operation the user was performing has been successful.

indico.core.signals.**after_process**
Called after an Indico request has been processed.

indico.core.signals.**app_created**
Called when the app has been created. The *sender* is the flask app.

indico.core.signals.**db_schema_created**
Executed when a new database schema is created. The *sender* is the name of the schema.

indico.core.signals.**form_validated**
Triggered when an IndicoForm was validated successfully. The *sender* is the form object.

This signal may return `False` to mark the form as invalid even though WTForms validation was successful. In this case it is highly recommended to mark a field as erroneous or indicate the error in some other way.

indico.core.signals.**get_conditions**
Expected to return one or more classes inheriting from *Condition*. The *sender* is a string (or some other object) identifying the context. The additional kwargs passed to this signal depend on the context.

indico.core.signals.**get_fields**
Expected to return *BaseField* subclasses. The *sender* is an object (or just a string) identifying for what to get fields. This signal should never be registered without restricting the sender to ensure only the correct field types are returned.

indico.core.signals.**get_placeholders**
Expected to return one or more *Placeholder* objects. The *sender* is a string (or some other object) identifying the context. The additional kwargs passed to this signal depend on the context.

indico.core.signals.**get_storage_backends**
Expected to return one or more Storage subclasses.

indico.core.signals.**import_tasks**
Called when Celery needs to import all tasks. Use this signal if you have modules containing task registered using one of the Celery decorators but don't import them anywhere. The signal handler should only `import` these modules and do nothing else.

indico.core.signals.**model_committed**
Triggered when an IndicoModel class was committed. The *sender* is the model class, the model instance is passed as *obj* and the change type as a string (delete/insert/update) in the *change* kwarg.

indico.core.signals.acl

indico.core.signals.acl.**can_access**
Called when *ProtectionMixin.can_access* is used to determine if a user can access something or not.

The *sender* is the type of the object that's using the mixin. The actual instance is passed as *obj*. The *user* and *allow_admin* arguments of *can_access* are passed as kwargs with the same name.

The *authorized* argument is `None` when this signal is called at the beginning of the access check and `True` or `False` at the end when regular access rights have already been checked. For expensive checks (such as anything involving database queries) it is recommended to skip the check while *authorized* is `None` since the regular access check is likely to be cheaper (due to ACLs being preloaded etc).

If the signal returns `True` or `False`, the access check succeeds or fails immediately. If multiple subscribers to the signal return contradictory results, `False` wins and access is denied.

`indico.core.signals.acl.`**`can_manage`**
Called when *ProtectionMixin.can_manage* is used to determine if a user can manage something or not.

The *sender* is the type of the object that's using the mixin. The actual instance is passed as *obj*. The *user*, *role*, *allow_admin*, *check_parent* and *explicit_role* arguments of *can_manage* are passed as kwargs with the same name.

If the signal returns `True` or `False`, the access check succeeds or fails without any further checks. If multiple subscribers to the signal return contradictory results, `False` wins and access is denied.

`indico.core.signals.acl.`**`entry_changed`**
Called when an ACL entry is changed.

The *sender* is the type of the object that's using the mixin. The actual instance is passed as *obj*. The *User*, *GroupProxy* or *EmailPrincipal* is passed as *principal* and *entry* contains the actual ACL entry (a *PrincipalMixin* instance) or `None` in case the entry was deleted. *is_new* is a boolean indicating whether the given principal was in the ACL before. If *quiet* is `True`, signal handlers should not perform noisy actions such as logging or sending emails related to the change.

If the ACL uses roles, *old_data* will contain a dictionary of the previous roles/permissions (see *PrincipalRolesMixin.current_data*).

`indico.core.signals.acl.`**`get_management_roles`**
Expected to return *ManagementRole* subclasses. The *sender* is the type of the object the roles may be used for. Functions subscribing to this signal **MUST** check the sender by specifying it using the first argument of *connect_via()* or by comparing it inside the function.

`indico.core.signals.acl.`**`protection_changed`**
Called when the protection mode of an object is changed.

The *sender* is the type of the object that's using the mixin. The actual instance is passed as *obj*. The old protection mode is passed as *old_mode*, the new mode as *mode*.

indico.core.signals.agreements

`indico.core.signals.agreements.`**`get_definitions`**
Expected to return a list of AgreementDefinition classes.

indico.core.signals.attachments

`indico.core.signals.attachments.`**`attachment_accessed`**
Called when an attachment is accessed. The *sender* is the *Attachment* that was accessed. The user who accessed the attachment is passed in the *user* kwarg. The *from_preview* kwarg will be set to `True` if the download link on the preview page was used to access the attachment or if the attachment was loaded to be displayed on the preview page (opening the preview itself already sends this signal with *from_preview=False*).

`indico.core.signals.attachments.`**`attachment_created`**
Called when a new attachment is created. The *sender* object is the new *Attachment*. The user who created the attachment is passed in the *user* kwarg.

indico.core.signals.attachments.**attachment_deleted**
Called when an attachment is deleted. The *sender* object is the *Attachment* that was deleted. The user who deleted the attachment is passed in the *user* kwarg.

indico.core.signals.attachments.**attachment_updated**
Called when an attachment is updated. The *sender* is the *Attachment* that was updated. The user who updated the attachment is passed in the *user* kwarg.

indico.core.signals.attachments.**folder_created**
Called when a new attachment folder is created. The *sender* is the new *AttachmentFolder* object. The user who created the folder is passed in the *user* kwarg. This signal is never triggered for the internal default folder.

indico.core.signals.attachments.**folder_deleted**
Called when a folder is deleted. The *sender* is the *AttachmentFolder* that was deleted. The user who deleted the folder is passed in the *user* kwarg.

indico.core.signals.attachments.**folder_updated**
Called when a folder is updated. The *sender* is the *AttachmentFolder* that was updated. The user who updated the folder is passed in the *user* kwarg.

indico.core.signals.attachments.**get_file_previewers**
Expected to return one or more *Previewer* subclasses.

indico.core.signals.category

indico.core.signals.category.**created**
Called when a new category is created. The *sender* is the new category.

indico.core.signals.category.**deleted**
Called when a category is deleted. The *sender* is the category.

indico.core.signals.category.**moved**
Called when a category is moved into another category. The *sender* is the category and the old parent category is passed in the *old_parent* kwarg.

indico.core.signals.category.**updated**
Called when a new category is created. The *sender* is the new category.

indico.core.signals.event

indico.core.signals.event.**abstract_created**
Called when a new abstract is created. The *sender* is the new abstract.

indico.core.signals.event.**abstract_deleted**
Called when an abstract is deleted. The *sender* is the abstract.

indico.core.signals.event.**abstract_state_changed**
Called when an abstract is withdrawn. The *sender* is the abstract.

indico.core.signals.event.**abstract_updated**
Called when an abstract is modified. The *sender* is the abstract.

indico.core.signals.event.**cloned**
Called when an event is cloned. The *sender* is the *Event* object of the old event, the new event is passed in the *new_event* kwarg.

indico.core.signals.event.**contribution_created**
Called when a new contribution is created. The *sender* is the new contribution.

indico.core.signals.event.**contribution_deleted**
Called when a contribution is deleted. The *sender* is the contribution.

indico.core.signals.event.**contribution_updated**
Called when a contribution is modified. The *sender* is the contribution. A dict containing `old, new` tuples for all changed values is passed in the `changes` kwarg.

indico.core.signals.event.**created**
Called when a new event is created. The *sender* is the new Event.

indico.core.signals.event.**deleted**
Called when an event is deleted. The *sender* is the event object. The *user* kwarg contains the user performing the deletion if available.

indico.core.signals.event.**generate_ticket_qr_code**
Called when generating the QR code for a ticket. The data included in the QR code is passed in the *ticket_data* kwarg and may be modified.

indico.core.signals.event.**get_feature_definitions**
Expected to return *EventFeature* subclasses.

indico.core.signals.event.**get_log_renderers**
Expected to return *EventLogRenderer* classes.

indico.core.signals.event.**is_ticketing_handled**
Called when resolving whether Indico should send tickets with e-mails or it will be handled by other module. The *sender* is the *RegistrationForm* object.

Should return `True` or `False`.

indico.core.signals.event.**moved**
Called when an event is moved to a different category. The *sender* is the event, the old category is in the *old_parent* kwarg.

indico.core.signals.event.**note_added**
Called when a note is added. The *sender* is the note.

indico.core.signals.event.**note_deleted**
Called when a note is deleted. The *sender* is the note.

indico.core.signals.event.**note_modified**
Called when a note is modified. The *sender* is the note.

indico.core.signals.event.**person_updated**
Called when an EventPerson is modified. The *sender* is the EventPerson.

indico.core.signals.event.**print_badge_template**
Called when printing a badge template. The registration form is passed in the *regform* kwarg.

indico.core.signals.event.**registration_deleted**
Called when a registration is removed. The *sender* is the *Registration* object.

indico.core.signals.event.**registration_form_created**
Called when a new registration form is created. The *sender* is the *RegistrationForm* object.

indico.core.signals.event.**registration_form_deleted**
Called when a registration form is removed. The *sender* is the *RegistrationForm* object.

indico.core.signals.event.**registration_personal_data_modified**
Called when the registration personal data is modified. The *sender* is the *Registration* object; the change is passed in the *change* kwarg.

indico.core.signals.event.**registration_state_updated**
Called when the state of registration changes. The *sender* is the *Registration* object; the previous state is passed in the *previous_state* kwarg.

indico.core.signals.event.**session_block_deleted**
Called when a session block is deleted. The *sender* is the session block. This signal is called before the `db.session.delete()` on the block is executed.

indico.core.signals.event.**session_deleted**
Called when a session is deleted. The *sender* is the session.

indico.core.signals.event.**session_updated**
Called when a session is updated. The *sender* is the session.

indico.core.signals.event.**sidemenu**
Expected to return `MenuEntryData` objects to be added to the event side menu. A single entry can be returned directly, multiple entries must be yielded.

indico.core.signals.event.**subcontribution_created**
Called when a new subcontribution is created. The *sender* is the new subcontribution.

indico.core.signals.event.**subcontribution_deleted**
Called when a subcontribution is deleted. The *sender* is the subcontribution.

indico.core.signals.event.**subcontribution_updated**
Called when a subcontribution is modified. The *sender* is the subcontribution.

indico.core.signals.event.**times_changed**
Called when the times of a scheduled object (contribution, break or session block) change, either by a change in duration or start time. The *sender* is the type of the object; the timetable entry is passed as *entry* and the object is passed as *obj*. Information about the changes are passed as *changes* which is a dict containing old/new tuples for *start_dt*, *duration* and *end_dt*. If an attribute did not change, it is not included in the dict. If the time of the event itself changes, *entry* is `None` and *obj* contains the *Event*.

indico.core.signals.event.**timetable_buttons**
Expected to return a list of tuples ('button_name', 'js-call-class'). Called when building the timetable view.

indico.core.signals.event.**timetable_entry_created**
Called when a new timetable entry is created. The *sender* is the new entry.

indico.core.signals.event.**timetable_entry_deleted**
Called when a timetable entry is deleted. The *sender* is the entry. This signal is triggered right before the entry deletion is performed.

indico.core.signals.event.**timetable_entry_updated**
Called when a timetable entry is updated. The *sender* is the entry. A dict containing `old, new` tuples for all changed values is passed in the `changes` kwarg.

indico.core.signals.event.**type_changed**
Called when the type of an event is changed. The *sender* is the event, the old type is passed in the *old_type* kwarg.

indico.core.signals.event.**updated**
Called when basic data of an event is updated. The *sender* is the event. A dict of changes is passed in the *changes* kwarg, with `(old, new)` tuples for each change. Note than the *person_links* change may happen with *old* and *new* being the same lists for technical reasons. If the key is present, it should be assumed that something changed (usually the order or some data on the person link).

indico.core.signals.event_management

indico.core.signals.event_management.**get_cloners**
: Expected to return one or more `EventCloner` subclasses implementing a cloning operation for something within an event.

indico.core.signals.event_management.**image_created**
: Called when a new image is created. The *sender* object is the new `ImageFile`. The user who uploaded the image is passed in the `user` kwarg.

indico.core.signals.event_management.**image_deleted**
: Called when an image is deleted. The *sender* object is the `ImageFile` that is about to be deleted. The user who uploaded the image is passed in the `user` kwarg.

indico.core.signals.event_management.**management_url**
: Expected to return a URL for the event management page of the plugin. This is used when someone who does not have event management access wants to go to the event management area. He is then redirected to one of the URLs returned by plugins, i.e. it is not guaranteed that the user ends up on a specific plugin's management page. The signal should return None if the current user (available via `session.user`) cannot access the management area. The *sender* is the event object.

indico.core.signals.menu

indico.core.signals.menu.**items**
: Expected to return one or more *SideMenuItem* to be added to the side menu. The *sender* is an id string identifying the target menu.

indico.core.signals.menu.**sections**
: Expected to return one or more *SideMenuSection* objects to be added to the side menu. The *sender* is an id string identifying the target menu.

indico.core.signals.plugin

indico.core.signals.plugin.**cli**
: Expected to return one or more click commands/groups. If they use *indico.cli.core.cli_command* / *indico.cli.core.cli_group* they will be automatically executed within a plugin context and run within a Flask app context by default.

indico.core.signals.plugin.**get_blueprints**
: Expected to return one or more IndicoPluginBlueprint-based blueprints which will be registered on the application. The Blueprint must be named either *PLUGINNAME* or *compat_PLUGINNAME*.

indico.core.signals.plugin.**get_conference_themes**
: Expected to return `(name, css, title)` tuples for conference stylesheets. `name` is the internal name used for the stylesheet which will be stored when the theme is selected in an event. `css` is the location of the CSS file, relative to the plugin's `static` folder. `title` is the title displayed to the user when selecting the theme.

indico.core.signals.plugin.**get_event_request_definitions**
: Expected to return one or more RequestDefinition subclasses.

indico.core.signals.plugin.**get_event_themes_files**
: Expected to return the path of a themes yaml containing event theme definitions.

indico.core.signals.plugin.**inject_css**
: Expected to return a list of CSS URLs which are loaded after all other CSS. The *sender* is the WP class of the page.

indico.core.signals.plugin.**inject_js**
: Expected to return a list of JS URLs which are loaded after all other JS. The *sender* is the WP class of the page.

indico.core.signals.plugin.**shell_context**
: Called after adding stuff to the *indico shell* context. Receives the *add_to_context* and *add_to_context_multi* keyword args with functions which allow you to add custom items to the context.

indico.core.signals.plugin.**template_hook**
: Expected to return a `(is_markup, priority, value)` tuple. The returned value will be inserted at the location where this signal is triggered; if multiple receivers are connected to the signal, they will be ordered by priority. If *is_markup* is True, the value will be wrapped in a *Markup* object which will cause it to be rendered as HTML. The *sender* is the name of the actual hook. The keyword arguments depend on the hook.

indico.core.signals.users

indico.core.signals.users.**email_added**
: Called when a new email address is added to a user. The *sender* is the user object and the email address is passed in the *email* kwarg.

indico.core.signals.users.**merged**
: Called when two users are merged. The *sender* is the main user while the merged user (i.e. the one being deleted in the merge) is passed via the *source* kwarg.

indico.core.signals.users.**preferences**
: Expected to return a *ExtraUserPreferences* subclass which implements extra preferences for the user preference page. The *sender* is the user for whom the preferences page is being shown which might not be the currently logged-in user!

indico.core.signals.users.**registered**
: Called once a user registers (either locally or joins through a provider). The *sender* is the new user object. The kwarg *from_moderation* indicates whether the user went through a moderation process (this also includes users created by an administrator manually) or was created immediately on registration; the identity associated with the registration is passed in the *identity* kwarg.

indico.core.signals.users.**registration_requested**
: Called when a user requests to register a new indico account, i.e. if moderation is enabled. The *sender* is the registration request.

Adding models to your plugin

Plugins must describe its database model the in the *models* folder if needed:

```
class Foo(db.Model):
    __tablename__ = 'foo'
    __table_args__ = {'schema': 'plugin_example'}

    id = db.Column(
        db.Integer,
        primary_key=True
    )
    bar = db.Column(
        db.String,
        nullable=False,
        default=''
    )
    location_id = db.Column(
```

```
        db.Integer,
        db.ForeignKey('roombooking.locations.id'),
        nullable=False
    )
    location = db.relationship(
        'Location',
        backref=db.backref('example_foo', cascade='all, delete-orphan', lazy='dynamic
↪'),
    )

    @return_ascii
    def __repr__(self):
        return u'<Foo({}, {}, {})>'.format(self.id, self.bar, self.location)
```

Thanks to **Alembic**, the migration needed to create the tables in the database can also be included in the plugin. The steps to do so are:

1. Create a revision for the changes your plugin will add with `indico db --plugin example migrate -m 'short description'`
2. Fine-tune the revision file generated under *migrations*.
3. Run `indico db --plugin example upgrade` to have Alembic upgrade your DB with the changes.

CHAPTER 4

HTTP API

Indico allows you to programmatically access the content of its database by exposing various information like category contents, events, rooms and room bookings through a web service, the HTTP Export API.

Indico - HTTP API

Indico allows you to programmatically access the content of its database by exposing various information like category contents, events, rooms and room bookings through a web service, the HTTP Export API.

Accessing the API

URL structure

Indico allows you to programmatically access the content of its database by exposing various information like category contents, events, rooms and room bookings through a web service, the HTTP Export API.

The basic URL looks like:

http://my.indico.server/export/WHAT/{[}LOC/{]}ID.TYPE?PARAMS&ak=KEY&timestamp=TS&signature=SIG

where:

- *WHAT* is the element you want to export (one of *categ*, *event*, *room*, *reservation*)
- *LOC* is the location of the element(s) specified by *ID* and only used for certain elements, for example, for the room booking (https://indico.server/export/room/CERN/120.json?ak=0...)
- *ID* is the ID of the element you want to export (can be a - separated list). As for example, the 120 in the above URL.
- *TYPE* is the output format (one of *json*, *jsonp*, *xml*, *html*, *ics*, *atom*, *bin*)
- *PARAMS* are various parameters affecting (filtering, sorting, ...) the result list
- *KEY*, *TS*, *SIG* are part of the *API Authentication*.

Some examples could be:

- Export data about events in a category: https://my.indico/export/categ/2.json?from=today&to=today&pretty=yes
- Export data about a event: https://indico.server/export/event/137346.json?occ=yes&pretty=yes
- Export data about rooms: https://indico.server/export/room/CERN/120.json?ak=00000000-0000-0000-0000-000000000000&pretty=yes
- Export your reservations: https://indico.server/export/reservation/CERN.json?ak=00000000-0000-0000-0000-000000000000&detail=reservations&from=today&to=today&bookedfor=USERNAME&pretty=yes

See more details about querying in Exporters.

API Authentication

General

The HTTP Export API uses an API key and - depending on the config - a cryptographic signature for each request.

To create an API key, go to *My Profile » HTTP API* and click the *Create API key* button. This will create an *API Key* and a *Secret Key* (if signatures are required).

It is recommended to always use the highest security level. That means if only an *API key* is available always include it and if a *secret key* is available, always sign your requests. Since you might want to retrieve only public information (instead of everything visible to your Indico user) you can add the param *onlypublic=yes* to the query string.

It is also possible to re-use the existing Indico session. This only makes sense if your browser accesses the API, e.g. because you are developing on Indico and want to access the API via an AJAX request. Additionally this method of authentication is restricted to GET requests. To use it, add *cookieauth=yes* to the query string and do not specify an API key, timestamp or signature. To prevent data leakage via CSRF the CSRF token of the current session needs to be provided as a GET argument *csrftoken* or a HTTP header *X-CSRF-Token*.

Request Signing

To sign a request, you need the following:

- The requested path, e.g. */export/categ/123.json*
- Any additional params, e.g. *limit=10*
- The current UNIX timestamp
- You *API key* and *secret key*

1. Add your API key to the params (*limit=10&ak=your-api-key*)
2. Add the current timestamp to the params (*limit=10&ak=your-api-key&timestamp=1234567890*)
3. Sort the query string params (*ak=your-api-key&limit=10&timestamp=1234567890*)
4. Merge path and the sorted query string to a single string (*/export/categ/123.json?ak=your-api-key&limit=10&timestamp=1234567890*)
5. Create a HMAC-SHA1 signature of this string using your *secret key* as the key.
6. Append the hex-encoded signature to your query string: *?ak=your-api-key&limit=10&timestamp=1234567890&signature=your-signature*

Note that a signed request might be valid only for a few seconds or minutes, so you **need** to sign it right before sending it and not store the generated URL as it is likely to expire soon.

You can find example code for Python and PHP in the following sections.

If persistent signatures are enabled, you can also omit the timestamp. In this case the URL is valid forever. When using this feature, please make sure to use these URLs only where necessary - use timestamped URLs whenever possible.

Request Signing for Python

A simple example in Python:

```
import hashlib
import hmac
import urllib
import time


def build_indico_request(path, params, api_key=None, secret_key=None, only_
↪public=False, persistent=False):
    items = params.items() if hasattr(params, 'items') else list(params)
    if api_key:
        items.append(('apikey', api_key))
    if only_public:
        items.append(('onlypublic', 'yes'))
    if secret_key:
        if not persistent:
            items.append(('timestamp', str(int(time.time()))))
        items = sorted(items, key=lambda x: x[0].lower())
        url = '%s?%s' % (path, urllib.urlencode(items))
        signature = hmac.new(secret_key, url, hashlib.sha1).hexdigest()
        items.append(('signature', signature))
    if not items:
        return path
    return '%s?%s' % (path, urllib.urlencode(items))


if __name__ == '__main__':
    API_KEY = '00000000-0000-0000-0000-000000000000'
    SECRET_KEY = '00000000-0000-0000-0000-000000000000'
    PATH = '/export/categ/1337.json'
    PARAMS = {
        'limit': 123
    }
    print build_indico_request(PATH, PARAMS, API_KEY, SECRET_KEY)
```

Request Signing for PHP

A simple example in PHP:

```
<?php

function build_indico_request($path, $params, $api_key = null, $secret_key = null,
↪$only_public = false, $persistent = false) {
    if($api_key) {
        $params['apikey'] = $api_key;
```

```
    }

    if($only_public) {
        $params['onlypublic'] = 'yes';
    }

    if($secret_key) {
        if(!$persistent) {
            $params['timestamp'] = time();
        }
        uksort($params, 'strcasecmp');
        $url = $path . '?' . http_build_query($params);
        $params['signature'] = hash_hmac('sha1', $url, $secret_key);
    }

    if(!$params) {
        return $path;
    }

    return $path . '?' . http_build_query($params);
}

if(true) { // change to false if you want to include this file
    $API_KEY = '00000000-0000-0000-0000-000000000000';
    $SECRET_KEY = '00000000-0000-0000-0000-000000000000';
    $PATH = '/export/categ/1337.json';
    $PARAMS = array(
        'limit' => 123
    );
    echo build_indico_request($PATH, $PARAMS, $API_KEY, $SECRET_KEY) . "\n";
}
```

Common Parameters

The following parameters are valid for all requests no matter which element is requested. If a parameter has a shorter form, it's given in parentheses.

Param	Short	Description
from/to	f/t	**Accepted formats:** • ISO 8601 subset - YYYY-MM-DD[THH:MM] • 'today', 'yesterday', 'tomorrow' and 'now' • days in the future/past: '[+/-]DdHHhMMm'
pretty	p	Pretty-print the output. When exporting as JSON it will include whitespace to make the json more human-readable.
onlypublic	op	Only return results visible to unauthenticated users when set to *yes*.
onlyauthed	oa	Fail if the request is unauthenticated for any reason when this is set to *yes*.
cookieauth	ca	Use the Indico session cookie to authenticate instead of an API key.
limit	n	Return no more than the X results.
offset	O	Skip the first X results.
detail	d	Specify the detail level (values depend on the exported element)
order	o	Sort the results. Must be one of *id*, *start*, *end*, *title*.
descending	c	Sort the results in descending order when set to *yes*.
tz	-	Assume given timezone (default UTC) for specified dates. Example: `Europe/Lisbon`.

API Resources

Categories

URL Format

/export/categ/ID.TYPE

The ID can be either a single category ID or a - separated list. In an authenticated request the special ID *favorites* will be resolved to the user's list of favorites.

Parameters

Param	Short	Description
location	l	Only include events taking place at the specified location. The * and *?* wildcards may be used.
room	r	Only include events taking place in the specified room. The * and *?* wildcards may be used.
type	T	Only include events of the specified type. Must be one of: simple_event (or lecture), meeting, conference

Detail Levels

events

Returns basic data about the events in the category.

This is the result of the following the query https://my.indico/export/categ/2.json?from=today&to=today&pretty=yes:


```
{
    "count": 2,
    "_type": "HTTPAPIResult",
    "complete": true,
    "url": "https://my.indico/export/categ/2.json?from=today&to=today&pretty=yes",
    "ts": 1308841641,
    "results": [
        {
            "category": "TEST Category",
            "startDate": {
                "date": "2011-06-17",
                "tz": "Europe/Zurich",
                "time": "08:00:00"
            },
            "_type": "Conference",
            "endDate": {
                "date": "2011-06-30",
                "tz": "Europe/Zurich",
                "time": "18:00:00"
            },
            "description": "",
            "title": "Test EPayment",
            "url": "http://pcituds07.cern.ch/indico/conferenceDisplay.py?confId=137344
↪",
            "location": "CERN",
            "_fossil": "conferenceMetadata",
            "timezone": "Europe/Zurich",
            "type": "conference",
            "id": "137344",
            "room": "1-1-025"
        },
        {
            "category": "TEST Category",
            "startDate": {
                "date": "2011-06-23",
                "tz": "Europe/Zurich",
                "time": "08:00:00"
            },
            "_type": "Conference",
```

```
            "endDate": {
                "date": "2011-06-24",
                "tz": "Europe/Zurich",
                "time": "18:00:00"
            },
            "description": "",
            "title": "Export Test",
            "url": "http://pcituds07.cern.ch/indico/conferenceDisplay.py?confId=137346
↪",
            "location": "CERN",
            "_fossil": "conferenceMetadata",
            "timezone": "Europe/Zurich",
            "type": "meeting",
            "id": "137346",
            "room": null
        }
    ]
}
```


Events

URL Format

/export/event/ID.TYPE

The ID can be either a single event ID or a - separated list.

Parameters

Param	Short	Description
occurrences	occ	Include the daily event times in the exported data.

Detail Levels

events

Returns basic data about the event. In this example occurrences are included, too.

Result for https://indico.server/export/event/137346.json?occ=yes&pretty=yes:


```
{
    "count": 1,
    "_type": "HTTPAPIResult",
    "complete": true,
    "url": "https://indico.server/export/event/137346.json?occ=yes&pretty=yes",
    "ts": 1308899256,
    "results": [
        {
            "category": "TEST Category",
            "startDate": {
                "date": "2011-06-23",
                "tz": "Europe/Zurich",
                "time": "08:00:00"
```

```
            },
            "_type": "Conference",
            "endDate": {
                "date": "2011-06-24",
                "tz": "Europe/Zurich",
                "time": "18:00:00"
            },
            "description": "",
            "title": "Export Test",
            "url": "http://indico.server/conferenceDisplay.py?confId=137346",
            "room": null,
            "occurrences": [
                {
                    "_fossil": "period",
                    "endDT": {
                        "date": "2011-06-23",
                        "tz": "Europe/Zurich",
                        "time": "08:40:00"
                    },
                    "startDT": {
                        "date": "2011-06-23",
                        "tz": "Europe/Zurich",
                        "time": "08:00:00"
                    },
                    "_type": "Period"
                },
                {
                    "_fossil": "period",
                    "endDT": {
                        "date": "2011-06-24",
                        "tz": "Europe/Zurich",
                        "time": "15:00:00"
                    },
                    "startDT": {
                        "date": "2011-06-24",
                        "tz": "Europe/Zurich",
                        "time": "12:00:00"
                    },
                    "_type": "Period"
                }
            ],
            "_fossil": "conferenceMetadata",
            "timezone": "Europe/Zurich",
            "type": "meeting",
            "id": "137346",
            "location": "CERN"
        }
    ]
}
```


contributions

Includes the contributions of the event.

Output for https://indico.server/export/event/137346.json?detail=contributions&pretty=yes:

```
{
    "count": 1,
    "_type": "HTTPAPIResult",
    "complete": true,
    "url": "https://indico.server/export/event/137346.json?detail=contributions&
↪pretty=yes",
    "ts": 1308899252,
    "results": [
        {
            "category": "TEST Category",
            "startDate": {
                "date": "2011-06-23",
                "tz": "Europe/Zurich",
                "time": "08:00:00"
            },
            "_type": "Conference",
            "endDate": {
                "date": "2011-06-24",
                "tz": "Europe/Zurich",
                "time": "18:00:00"
            },
            "description": "",
            "title": "Export Test",
            "url": "http://indico.server/conferenceDisplay.py?confId=137346",
            "type": "meeting",
            "location": "CERN",
            "_fossil": "conferenceMetadataWithContribs",
            "timezone": "Europe/Zurich",
            "contributions": [
                {
                    "startDate": {
                        "date": "2011-06-23",
                        "tz": "Europe/Zurich",
                        "time": "08:20:00"
                    },
                    "_type": "Contribution",
                    "endDate": {
                        "date": "2011-06-23",
                        "tz": "Europe/Zurich",
                        "time": "08:40:00"
                    },
                    "description": "",
                    "title": "d1c2",
                    "track": null,
                    "duration": 20,
                    "session": null,
                    "location": "CERN",
                    "_fossil": "contributionMetadata",
                    "type": null,
                    "id": "1",
                    "room": null
                },
                {
                    "startDate": {
                        "date": "2011-06-23",
                        "tz": "Europe/Zurich",
                        "time": "08:00:00"
                    },
```

```
                    "_type": "Contribution",
                    "endDate": {
                        "date": "2011-06-23",
                        "tz": "Europe/Zurich",
                        "time": "08:20:00"
                    },
                    "description": "",
                    "title": "d1c1",
                    "track": null,
                    "duration": 20,
                    "session": null,
                    "location": "CERN",
                    "_fossil": "contributionMetadata",
                    "type": null,
                    "id": "0",
                    "room": null
                },
                {
                    "startDate": {
                        "date": "2011-06-24",
                        "tz": "Europe/Zurich",
                        "time": "14:00:00"
                    },
                    "_type": "Contribution",
                    "endDate": {
                        "date": "2011-06-24",
                        "tz": "Europe/Zurich",
                        "time": "14:20:00"
                    },
                    "description": "",
                    "title": "d2s1c1",
                    "track": null,
                    "duration": 20,
                    "session": "d2s1",
                    "location": "CERN",
                    "_fossil": "contributionMetadata",
                    "type": null,
                    "id": "3",
                    "room": null
                },
                {
                    "startDate": {
                        "date": "2011-06-24",
                        "tz": "Europe/Zurich",
                        "time": "12:00:00"
                    },
                    "_type": "Contribution",
                    "endDate": {
                        "date": "2011-06-24",
                        "tz": "Europe/Zurich",
                        "time": "14:00:00"
                    },
                    "description": "",
                    "title": "d2c1",
                    "track": null,
                    "duration": 120,
                    "session": null,
                    "location": "CERN",
```

```
                    "_fossil": "contributionMetadata",
                    "type": null,
                    "id": "2",
                    "room": null
                }
            ],
            "id": "137346",
            "room": null
        }
    ]
}
```

subcontributions

Like *contributions*, but inside the contributions the subcontributions are included in a field named *subContributions*.

sessions

Includes details about the different sessions and groups contributions by sessions. The top-level *contributions* list only contains contributions which are not assigned to any session. Subcontributions are included in this details level, too.

For example, https://indico.server/export/event/137346.json?detail=sessions&pretty=yes:

```
{
    "count": 1,
    "_type": "HTTPAPIResult",
    "complete": true,
    "url": "https://indico.server/export/event/137346.json?detail=sessions&pretty=yes
↪",
    "ts": 1308899771,
    "results": [
        {
            "category": "TEST Category",
            "startDate": {
                "date": "2011-06-23",
                "tz": "Europe/Zurich",
                "time": "08:00:00"
            },
            "_type": "Conference",
            "endDate": {
                "date": "2011-06-24",
                "tz": "Europe/Zurich",
                "time": "18:00:00"
            },
            "description": "",
            "title": "Export Test",
            "url": "http://indico.server/conferenceDisplay.py?confId=137346",
            "contributions": [
                {
                    "startDate": {
                        "date": "2011-06-23",
                        "tz": "Europe/Zurich",
                        "time": "08:20:00"
                    },
                    "_type": "Contribution",
```

```
                    "endDate": {
                        "date": "2011-06-23",
                        "tz": "Europe/Zurich",
                        "time": "08:40:00"
                    },
                    "description": "",
                    "subContributions": [],
                    "title": "d1c2",
                    "track": null,
                    "duration": 20,
                    "session": null,
                    "location": "CERN",
                    "_fossil": "contributionMetadataWithSubContribs",
                    "type": null,
                    "id": "1",
                    "room": null
                },
                {
                    "startDate": {
                        "date": "2011-06-23",
                        "tz": "Europe/Zurich",
                        "time": "08:00:00"
                    },
                    "_type": "Contribution",
                    "endDate": {
                        "date": "2011-06-23",
                        "tz": "Europe/Zurich",
                        "time": "08:20:00"
                    },
                    "description": "",
                    "subContributions": [],
                    "title": "d1c1",
                    "track": null,
                    "duration": 20,
                    "session": null,
                    "location": "CERN",
                    "_fossil": "contributionMetadataWithSubContribs",
                    "type": null,
                    "id": "0",
                    "room": null
                },
                {
                    "startDate": {
                        "date": "2011-06-24",
                        "tz": "Europe/Zurich",
                        "time": "12:00:00"
                    },
                    "_type": "Contribution",
                    "endDate": {
                        "date": "2011-06-24",
                        "tz": "Europe/Zurich",
                        "time": "14:00:00"
                    },
                    "description": "",
                    "subContributions": [],
                    "title": "d2c1",
                    "track": null,
                    "duration": 120,
```

```
                "session": null,
                "location": "CERN",
                "_fossil": "contributionMetadataWithSubContribs",
                "type": null,
                "id": "2",
                "room": null
            }
        ],
        "sessions": [
            {
                "startDate": {
                    "date": "2011-06-24",
                    "tz": "Europe/Zurich",
                    "time": "14:00:00"
                },
                "_type": "Session",
                "room": "",
                "numSlots": 1,
                "color": "#EEE0EF",
                "material": [],
                "isPoster": false,
                "sessionConveners": [],
                "location": "CERN",
                "address": "",
                "_fossil": "sessionMetadata",
                "title": "d2s1",
                "textColor": "#1D041F",
                "contributions": [
                    {
                        "startDate": {
                            "date": "2011-06-24",
                            "tz": "Europe/Zurich",
                            "time": "14:00:00"
                        },
                        "_type": "Contribution",
                        "endDate": {
                            "date": "2011-06-24",
                            "tz": "Europe/Zurich",
                            "time": "14:20:00"
                        },
                        "description": "",
                        "subContributions": [],
                        "title": "d2s1c1",
                        "track": null,
                        "duration": 20,
                        "session": "d2s1",
                        "location": "CERN",
                        "_fossil": "contributionMetadataWithSubContribs",
                        "type": null,
                        "id": "3",
                        "room": null
                    }
                ],
                "id": "0"
            }
        ],
        "location": "CERN",
        "_fossil": "conferenceMetadataWithSessions",
```

```
            "timezone": "Europe/Zurich",
            "type": "meeting",
            "id": "137346",
            "room": null
        }
    ]
}
```


Timetable

URL Format

/export/timetable/ID.TYPE

The ID should be the event ID, e.g. *123*.

Results

Returns the timetable of the event.

Result for https://indico.server/export/timetable/137346.json?ak=00000000-0000-0000-0000-000000000000&pretty=yes:


```
{
    "count": 1,
    "additionalInfo": {},
    "_type": "HTTPAPIResult",
    "complete": true,
    "url": "https:\/\/indico.server\/export\/timetable\/137346.json?ak=00000000-0000-
↪0000-0000-000000000000&pretty=yes",
    "ts": 1367242732,
    "results": {
        "137346": {
            "20130429": {
                "c0": {
                    "startDate": {
                        "date": "2013-04-29",
                        "tz": "Europe\/Zurich",
                        "time": "16:00:00"
                    },
                    "_type": "ContribSchEntry",
                    "material": [],
                    "endDate": {
                        "date": "2013-04-29",
                        "tz": "Europe\/Zurich",
                        "time": "16:30:00"
                    },
                    "description": "",
                    "title": "Contrib 1",
                    "id": "c0",
                    "contributionId": "0",
                    "sessionSlotId": null,
                    "conferenceId": "137346",
                    "presenters": [],
                    "sessionId": null,
```

```
                    "location": "CERN",
                    "uniqueId": "a137346t0",
                    "_fossil": "contribSchEntryDisplay",
                    "sessionCode": null,
                    "entryType": "Contribution",
                    "room": "160-1-009"
                }
            }
        }
```


Event Search

URL Format

/export/event/search/TERM.TYPE

The TERM should be a string, e.g. “ichep”

Results

Returns the events found.

Result for https://indico.server/export/event/search/ichep.json?ak=00000000-0000-0000-0000-000000000000&pretty=yes:


```
{
    "count": 5,
    "additionalInfo": {},
    "_type": "HTTPAPIResult",
    "complete": true,
    "url": "https:\/\/indico.server\/export\/event\/search\/ichep.json?ak=00000000-
↪0000-0000-0000-000000000000&pretty=yes",
    "ts": 1367245058,
    "results": [
        {
            "startDate": {
                "date": "2010-07-16",
                "tz": "UTC",
                "time": "11:00:00"
            },
            "hasAnyProtection": false,
            "id": "101465",
            "title": "Rehearsals for ICHEP Friday 16th July Afternoon Session"
        },
        {
            "startDate": {
                "date": "2010-08-06",
                "tz": "UTC",
                "time": "12:00:00"
            },
            "hasAnyProtection": false,
            "id": "102669",
            "title": "Overview of LHC physics results at ICHEP"
        },
        {
```

```
            "startDate": {
                "date": "2010-08-18",
                "tz": "UTC",
                "time": "17:00:00"
            },
            "hasAnyProtection": false,
            "id": "104128",
            "title": "Seminer Oturumu: \"ATLAS status and highlights as of ICHEP\" Dr.
↪ Tayfun Ince (Universitaet Bonn)"
        },
        {
            "startDate": {
                "date": "2011-07-23",
                "tz": "UTC",
                "time": "11:00:00"
            },
            "hasAnyProtection": false,
            "id": "145521",
            "title": "89th Plenary ECFA and Joint EPS\/ICHEP-ECFA Session - Grenoble,␣
↪France"
        },
        {
            "startDate": {
                "date": "2012-01-12",
                "tz": "UTC",
                "time": "08:00:00"
            },
            "hasAnyProtection": false,
            "id": "168897",
            "title": "ICHEP 2012 Outreach Planning Meeting"
        }
    ]
}
```


Files

General Information

The file export is only available for authenticated users, i.e. when using an API key and a signature (if enabled).

URL Format

/export/event/EVENT_ID/session/SESSION_ID/contrib/CONTRIBUTION_ID/subcontrib/SUBCONTRIBUTION_ID/material/MATERIA

All ID's should be single ID, not separated list.

The *EVENT_ID* should be the event ID, e.g. *123*.
The *SESSION_ID (optional)* should be the session ID, e.g. *4*.
The *CONTRIBUTION_ID (optional)* should be the contribution ID, e.g. *3*.
The *SUBCONTRIBUTION_ID (optional)* should be the sub-contribution ID, e.g. *1*.
The *MATERIAL_ID* should by the material name if it came default group e.g. *Slides* or material ID if not, e.g. *2*.
The *RESOURCE_ID* should by the resource ID.

Only supported *TYPE* for files is *bin* (binary data).

Parameters

None

Detail Levels

file

Returns file (or an error in *JSON* format).

For example: https://indico.server/export/event/23/session/0/contrib/3/material/slides/3.bin?ak=00000000-0000-0000-0000-000000000000

User

General Information

The user export is only available for authenticated users, i.e. when using an API key and a signature (if enabled).

URL Format

/export/user/USER_ID.TYPE

The *USER_ID* should be the user ID, e.g. *44*.

Parameters

None

Results

Returns the user information (or an error in *JSON* format).

Result for https://indico.server/export/user/36024.json?ak=00000000-0000-0000-0000-000000000000&pretty=yes:


```
{
    "count": 1,
    "additionalInfo": {},
    "_type": "HTTPAPIResult",
    "complete": true,
    "url": "https:\/\/indico.server\/export\/user\/36024.json?ak=00000000-0000-0000-
↪0000-000000000000&pretty=yes",
    "ts": 1367243741,
    "results": [
        {
        "_type": "Avatar",
        "name": "Alberto RESCO PEREZ",
        "firstName": "Alberto",
```

```
        "affiliation": "CERN",
        "familyName": "Resco Perez",
        "email": "test@cern.ch",
        "phone": "+41XXXXXXXXX",
        "_fossil": "avatar",
        "title": "",
        "id": "36024"
        }
    ]
}
```


Room Booking

Bookings

Creating bookings

General Information

The Room Booking API is only available for authenticated users, i.e. when using an API key and a signature (if enabled). If the room booking system is restricted to certain users/groups this restriction applies for this API, too. The request will fail if there is a collision with another booking, blocking or unavailable period.

Note that it is not possible to pre-book a room through this api.

URL Format

/api/roomBooking/bookRoom.TYPE

TYPE should be *json* or *xml*.

Parameters

The following parameters are required:

Param	Values	Description
location	text	Room location, e.g. *CERN*
roomid	text	Room id
from/to	f/t	**Start/End time for a booking. Accepted formats:** • ISO 8601 subset - YYYY-MM-DD[THH:MM] • 'today', 'yesterday', 'tomorrow' and 'now' • days in the future/past: '[+/-]DdHHhMMm'
reason	text	Reason for booking a room
username	text	User login name for whom the booking will be created

Booking a room

POST request

Returns *reservation id* if the booking was successful or error information it there were any problems.

For example:

```
curl --data "username=jdoe&from=2012-12-30T21:30&to=2012-12-30T22:15&reason=meeting&
↪location=CERN&roomid=189" 'http://indico.server/indico/api/roomBooking/bookRoom.json
↪'
```

Result:

```
{
    {
        "url": "\/api\/roomBooking\/bookRoom.json",
        "_type": "HTTPAPIResult",
        "results": {
            "reservationID": 45937
        },
        "ts": 1354695663
    }
}
```

Retrieving bookings

General Information

The reservation export is only availabled for authenticated users, i.e. when using an API key and a signature (if enabled). If the room booking system is restricted to certain users/groups this restriction applies for the reservation export API, too.

Please note that the room export with the *reservations* detail level is much more appropriate if you need reservations for specific rooms.

URL Format

/export/reservation/LOCATION.TYPE

The *LOCATION* should be the room location, e.g. *CERN*. A - separated list of multiple locations is allowed, too.

Parameters

Param	Short	Values	Description
occurrences	occ	yes, no	Include all occurrences of room reservations.
cancelled	cxl	yes, no	If specified only include cancelled (*yes*) or non-cancelled (*no*) reservations.
rejected	rej	yes, no	If specified only include rejected/non-rejected resvs.
confirmed	-	yes, no, pending	If specified only include bookings/pre-bookings with the given state.
archival	arch	yes, no	If specified only include bookings (not) from the past.
recurring	rec	yes, no	If specified only include bookings which are (not) recurring.
repeating	rep	yes, no	Alias for *recurring*
avc	-	yes, no	If specified only include bookings which (do not) use AVC.
avcsupport	avcs	yes, no	If specified only include bookings which (do not) need AVC Support.
startupsupport	sts	yes, no	If specified only include bookings which (do not) need Startup Support.
bookedfor	bf	text (wildcards)	Only include bookings where the *booked for* field matches the given wildcard string.
occurs	-	yyyy-mm-dd	Only include bookings which have a valid occurrence on the given date. Multiple dates can be separated by commas.

Detail Levels

reservations

Returns detailed data about the reservations and the most important information about the booked room.

For example, https://indico.server/export/reservation/CERN.json?ak=00000000-0000-0000-0000-000000000000&detail=reservation&from=today&to=today&pretty=yes:


```
{
    "count": 1,
    "additionalInfo": {},
    "_type": "HTTPAPIResult",
    "url": "/export/reservation/CERN.json?ak=00000000-0000-0000-0000-000000000000&
↪detail=reservation&from=today&to=today&pretty=yes",
    "results": [
        {
            "_type": "Reservation",
            "repeat_unit": 1,
            "endDT": {
                "date": "2014-08-14",
                "tz": "Europe/Zurich",
                "time": "12:30:00"
            },
            "room": {
                "_type": "Room",
                "fullName": "500-1-001 - Main Auditorium",
                "id": 57
            },
            "needs_general_assistance": false,
            "isConfirmed": true,
            "isValid": true,
```

```
            "usesAVC": false,
            "repeatability": "daily",
            "repeat_step": 1,
            "vcList": [],
            "reason": "Summer Student Lecture programme",
            "bookedForName": "DOE, John",
            "is_rejected": false,
            "is_cancelled": false,
            "needsAVCSupport": false,
            "startDT": {
                "date": "2014-07-02",
                "tz": "Europe/Zurich",
                "time": "08:30:00"
            },
            "id": 63779,
            "bookingUrl": "http://indico.server/rooms/booking/CERN/63779/",
            "location": "CERN"
        }
    ],
    "ts": 1406727843
}
```


Rooms

General Information

The room export is only availabled for authenticated users, i.e. when using an API key and a signature (if enabled). If the room booking system is restricted to certain users/groups this restriction applies for the room export API, too.

URL Format

/export/room/LOCATION/ID.TYPE

The *LOCATION* should be the room location, e.g. *CERN*. The *ID* can be either a single room ID or a - separated list.

Parameters

Param	Short	Values	Description
occurrences	occ	yes, no	Include all occurrences of room reservations.
cancelled	cxl	yes, no	If specified only include cancelled (*yes*) or non-cancelled (*no*) reservations.
rejected	rej	yes, no	If specified only include rejected/non-rejected resvs.
confirmed	-	yes, no, pending	If specified only include bookings/pre-bookings with the given state.
archival	arch	yes, no	If specified only include bookings (not) from the past.
recurring	rec	yes, no	If specified only include bookings which are (not) recurring.
repeating	rep	yes, no	Alias for *recurring*
avc	-	yes, no	If specified only include bookings which (do not) use AVC.
avcsupport	avcs	yes, no	If specified only include bookings which (do not) need AVC Support.
startupsupport	sts	yes, no	If specified only include bookings which (do not) need Startup Support.
bookedfor	bf	text (wildcards)	Only include bookings where the *booked for* field matches the given wildcard string.
occurs	-	yyyy-mm-dd	Only include bookings which have a valid occurrence on the given date. Multiple dates can be separated by commas.

Detail Levels

rooms

Returns basic data about the rooms.

For example, https://indico.server/export/room/CERN/57.json?ak=00000000-0000-0000-0000-000000000000&pretty=yes:


```
{
    "count": 1,
    "additionalInfo": {},
    "_type": "HTTPAPIResult",
    "url": "/export/room/CERN/57.json?ak=00000000-0000-0000-0000-000000000000&
↪pretty=yes",
    "results": [
        {
            "building": "500",
            "_type": "Room",
            "name": "Main Auditorium",
            "floor": "1",
            "longitude": "6.0542704900999995",
            "vcList": [
                "Audio Conference",
                "Built-in (MCU) Bridge",
                "CERN MCU",
                "ESnet MCU",
                "EVO",
                "H323 point2point",
                "Vidyo"
            ],
            "equipment": [
```

```
                "Blackboard",
                "Computer Projector",
                "Ethernet",
                "Microphone",
                "PC",
                "Telephone conference",
                "Video conference",
                "Webcast/Recording",
                "Wireless"
            ],
            "roomNr": "001",
            "location": "CERN",
            "latitude": "46.23141394580001",
            "fullName": "500-1-001 - Main Auditorium",
            "id": 57,
            "bookingUrl": "/indico/rooms/room/CERN/57/book",
            "avc": true
        }
    ],
    "ts": 1406729635
}
```


reservations

Returns basic data about the rooms and their reservations in the given timeframe.

Output for https://indico.server/export/room/CERN/57.json?ak=00000000-0000-0000-0000-000000000000&detail=reservations&from=today&to=today&pretty=yes:


```
{
    "count": 1,
    "additionalInfo": {},
    "_type": "HTTPAPIResult",
    "url": "/export/room/CERN/57.json?ak=00000000-0000-0000-0000-000000000000&
↪detail=reservations&from=today&to=today&pretty=yes",
    "results": [
        {
            "building": "500",
            "_type": "Room",
            "name": "Main Auditorium",
            "floor": "1",
            "reservations": [
                {
                    "_type": "Reservation",
                    "repeat_unit": 1,
                    "endDT": {
                        "date": "2014-08-14",
                        "tz": "Europe/Zurich",
                        "time": "12:30:00"
                    },
                    "needs_general_assistance": false,
                    "isConfirmed": true,
                    "isValid": true,
                    "usesAVC": false,
                    "repeatability": "daily",
                    "repeat_step": 1,
```

```
                    "vcList": [],
                    "reason": "Summer Student Lecture programme",
                    "bookedForName": "DOE, John",
                    "is_rejected": false,
                    "is_cancelled": false,
                    "needsAVCSupport": false,
                    "startDT": {
                        "date": "2014-07-02",
                        "tz": "Europe/Zurich",
                        "time": "08:30:00"
                    },
                    "id": 63779,
                    "bookingUrl": "http://pcavc005.cern.ch:8000/indico/rooms/booking/
↪CERN/63779/",
                    "location": "CERN"
                }
            ],
            "longitude": "6.0542704900999995",
            "vcList": [
                "Audio Conference",
                "Built-in (MCU) Bridge",
                "CERN MCU",
                "ESnet MCU",
                "EVO",
                "H323 point2point",
                "Vidyo"
            ],
            "equipment": [
                "Blackboard",
                "Computer Projector",
                "Ethernet",
                "Microphone",
                "PC",
                "Telephone conference",
                "Video conference",
                "Webcast/Recording",
                "Wireless"
            ],
            "roomNr": "001",
            "location": "CERN",
            "latitude": "46.23141394580001",
            "fullName": "500-1-001 - Main Auditorium",
            "id": 57,
            "bookingUrl": "/indico/rooms/room/CERN/57/book",
            "avc": true
        }
    ],
    "ts": 1406731966
}
```


Get room by room name

General Information

The search room export is guest allowed because the room data is public (no the reservations).

URL Format

/export/roomName/LOCATION/ROOMNAME.TYPE

The *LOCATION* should be the room location, e.g. *CERN*. The *ROOMNAME* is a single ROOMNAME.

Parameters

No parameters needed.

Results

Returns basic data about the rooms.

For example, https://indico.server/export/roomName/CERN/Main Auditorium.json?ak=00000000-0000-0000-0000-000000000000&pretty=yes:


```
{
    "count": 1,
    "additionalInfo": {},
    "_type": "HTTPAPIResult",
    "url": "/export/roomName/CERN/Main Auditorium.json?ak=00000000-0000-0000-0000-
↪000000000000&pretty=yes",
    "results": [
        {
            "building": "500",
            "_type": "Room",
            "name": "Main Auditorium",
            "floor": "1",
            "longitude": "6.0542704900999995",
            "vcList": [
                "Audio Conference",
                "Built-in (MCU) Bridge",
                "CERN MCU",
                "ESnet MCU",
                "EVO",
                "H323 point2point",
                "Vidyo"
            ],
            "equipment": [
                "Blackboard",
                "Computer Projector",
                "Ethernet",
                "Microphone",
                "PC",
                "Telephone conference",
                "Video conference",
                "Webcast/Recording",
                "Wireless"
            ],
            "roomNr": "001",
            "location": "CERN",
            "latitude": "46.23141394580001",
            "fullName": "500-1-001 - Main Auditorium",
            "id": 57,
            "bookingUrl": "/indico/rooms/room/CERN/57/book",
```

```
            "avc": true
        }
    ],
    "ts": 1406732578
}
```


Video Services & Collaboration

URL Format

/export/video/SERVICE_ID.TYPE

The SERVICE_ID may be a single collaboration type or many separated by -. At present, the only TYPE compatible with the Video Services export is *ics* / iCalendar.

As the query is signed with a signature generated using secret API key, the query need not be timestamped. Instead, each booking is given its own unique identifier and, therefore, the generated query URL may be fed as a persistent calendar for importing in your application of choice. The link will only expire once your account has been closed, if TTL is required by your server administrator or your API key is deleted.

If TTL is required by your server administrator, requests should be both timestamped and signed.

Parameters

Param	Short	Values	Description
alarms	-	int	If defined with a value of x int, all bookings to be exported will be accompanied by a matching alarm set to occur x minutes prior to the start of the booking itself. The alarm is set to provide a popup reminder. The default value is 0 minutes.

Please be aware that specifying the alarm parameter in your query will assign alarms to *every* booking which is to be exported.

Service Identifiers Used in CERN

The following parameters are both for example to other installations, and for use within CERN installations of Indico, they represent the options available for configuration through the SERVICE_ID parameter.

SERVICE_ID	Linked Service
all	Traverse all plugin indices.
vidyo	Return Vidyo bookings only.
evo	Return EVO bookings only.
mcu	Return CERNMCU bookings only.
webcast	Return Webcast Requests only.
recording	Return Recording Requests only.

SERVICE_ID may be one of more of these identifiers, if more than one is required simply join the service names with -, please refer to common examples for usage scenarios.

Common Examples

All Bookings

To obtain all bookings in the past 7 days for all collaboration plugins registered:

https://indico.server/export/video/all.ics?ak=API_KEY&from=-70000&to=now&signature=SIGNATURE

To obtain the same output, but with alarms set to display 20 minutes prior to each event:

https://indico.server/export/video/all.ics?ak=API_KEY&alarms=20&from=-70000&to=now&signature=SIGNATURE

Individual Plugin Bookings

To obtain bookings from a specific plugin, in this example Vidyo, from a set date range and with alarms 30 minutes prior to the booking:

https://indico.server/export/video/vidyo.ics?ak=API_KEY&alarms=30&from=2011-08-01&to=2011-12-01&signature=SIGNATURE

Multiple Plugin Bookings

We may also reference more than one plugin's bookings, to request all EVO and CERNMCU bookings over a specified date range with no alarms:

https://indico.server/export/video/evo-mcu.ics?ak=API_KEY&from=2011-09-01&to=2011-09-08&signature=SIGNATURE

HTTP API Tools

CHAPTER 5

API reference

This part of the documentation focuses on the core modules of Indico and includes information about the **models** and **utility functions and classes** that are useful for understanding the internals of the application.

API reference

This part of the documentation focuses on the core modules of Indico and includes information about the **models** and **utility functions and classes** that are useful for understanding the internals of the application.

Event

Todo

Docstrings (module, models, operations, utilities, settings)

Models

class indico.modules.events.models.events.**Event**(***kwargs*)

Bases: indico.core.db.sqlalchemy.searchable_titles.SearchableTitleMixin, indico.core.db.sqlalchemy.descriptions.DescriptionMixin, indico.core.db.sqlalchemy.locations.LocationMixin, indico.core.db.sqlalchemy.protection.ProtectionManagersMixin, indico.core.db.sqlalchemy.attachments.AttachedItemsMixin, indico.core.db.sqlalchemy.notes.AttachedNotesMixin, *indico.modules.events.models.persons.PersonLinkDataMixin*, flask_sqlalchemy.Model

An Indico event

This model contains the most basic information related to an event.

Note that the ACL is currently only used for managers but not for view access!

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

ATTACHMENT_FOLDER_ID_COLUMN = u'event_id'

access_key

acl_entries
The ACL entries for the event

additional_info

allow_access_key = True

allow_location_inheritance = False

allow_no_access_contact = True

as_legacy
Return a legacy *Conference* object

can_lock(*user*)
Check whether the user can lock/unlock the event

category
The category containing the event

classmethod category_chain_overlaps(*category_ids*)
Create a filter that checks whether the event has any of the provided category ids in its parent chain.

Parameters **category_ids** – A list of category ids or a single category id

category_id
The ID of immediate parent category of the event

cfa

cfp

cloned_from
The event this one was cloned from

cloned_from_id
If this event was cloned, the id of the parent event

contact_emails

contact_phones

contact_title

created_dt
The creation date of the event

creator
The user who created the event

creator_id
The ID of the user who created the event

default_page
The event's default page (conferences only)

default_page_id
The ID of the event's default page (conferences only)

default_render_mode = 1

delete(*reason*, *user=None*)

disallowed_protection_modes = frozenset([])

display_tzinfo
The tzinfo of the event as preferred by the current user

duration

end_dt
The end date of the event

end_dt_display
The 'displayed end dt', which is usually the actual end dt, but may be overridden for a conference.

end_dt_local

end_dt_override

ends_after(*dt*)
Check whether the event ends on/after the specified date

event
Convenience property so all event entities have it

external_url

get_allowed_sender_emails(*include_current_user=True*, *include_creator=True*, *include_managers=True*, *include_contact=True*, *include_chairs=True*, *extra=None*)
Return the emails of people who can be used as senders (or rather Reply-to contacts) in emails sent from within an event.

Parameters

- **include_current_user** – Whether to include the email of the currently logged-in user
- **include_creator** – Whether to include the email of the event creator
- **include_managers** – Whether to include the email of all event managers
- **include_contact** – Whether to include the "event contact" emails
- **include_chairs** – Whether to include the emails of event chairpersons (or lecture speakers)
- **extra** – An email address that is always included, even if it is not in any of the included lists.

Returns An OrderedDict mapping emails to pretty names

get_contribution(*id_*)
Get a contribution of the event

get_contribution_field(*field_id*)

get_non_inheriting_objects()
Get a set of child objects that do not inherit protection

get_relative_event_ids()
Get the first, last, previous and next event IDs.

Any of those values may be `None` if there is no matching event or if it would be the current event.

Returns A dict containing `first`, `last`, `prev` and `next`.

get_session(*id_=None*, *friendly_id=None*)
Get a session of the event

get_session_block(*id_*, *scheduled_only=False*)
Get a session block of the event

get_verbose_title(*show_speakers=False*, *show_series_pos=False*)
Get the event title with some additional information

Parameters

- **show_speakers** – Whether to prefix the title with the speakers of the event.
- **show_series_pos** – Whether to suffix the title with the position and total count in the event's series.

global_abstract_reviewers
Users who can review on all tracks

global_conveners
Users who are conveners on all tracks

happens_between(*from_dt=None*, *to_dt=None*)
Check whether the event takes place within two dates

has_feature(**args*, ***kwargs*)
Checks if a feature is enabled for the event

has_logo

has_stylesheet

id
The ID of the event

inherit_location = False

inheriting_have_acl = True

is_deleted
If the event has been deleted

is_locked
If the event is locked (read-only mode)

classmethod is_visible_in(*category*)
Create a filter that checks whether the event is visible in the specified category.

iter_days(*tzinfo=None*)

keywords
A list of tags/keywords for the event

location_backref_name = u'events'

locator

log(*realm*, *kind*, *module*, *summary*, *user=None*, *type_=u'simple'*, *data=None*)
Creates a new log entry for the event

Parameters

- **realm** – A value from *EventLogRealm* indicating the realm of the action.
- **kind** – A value from *EventLogKind* indicating the kind of the action that was performed.
- **module** – A human-friendly string describing the module related to the action.
- **summary** – A one-line summary describing the logged action.
- **user** – The user who performed the action.
- **type** – The type of the log entry. This is used for custom rendering of the log message/data
- **data** – JSON-serializable data specific to the log type.

In most cases the `simple` log type is fine. For this type, any items from data will be shown in the detailed view of the log entry. You may either use a dict (which will be sorted) alphabetically or a list of `key, value` pairs which will be displayed in the given order.

logging_disabled
Temporarily disables event logging

This is useful when performing actions e.g. during event creation or at other times where adding entries to the event log doesn't make sense.

logo
The logo's raw image data

logo_metadata
The metadata of the logo (hash, size, filename, content_type)

logo_url

move(*category*)

move_start_dt(*start_dt*)
Set event start_dt and adjust its timetable entries

organizer_info

own_address

own_no_access_contact

own_room

own_room_id

own_room_name

own_venue

own_venue_id

own_venue_name

participation_regform

person_links
Persons associated with this event

possible_render_modes = set([<RenderMode.html: 1>])

preload_all_acl_entries()

protection_mode

protection_parent

published_registrations

references
External references associated with this event

render_mode = 1

scheduled_notes

series
The series this event is part of

series_id
The ID of the series this events belongs to

short_external_url

short_url

start_dt
The start date of the event

start_dt_display
The 'displayed start dt', which is usually the actual start dt, but may be overridden for a conference.

start_dt_local

start_dt_override

starts_between (*from_dt=None*, *to_dt=None*)
Check whether the event starts within two dates

stylesheet
The stylesheet's raw image data

stylesheet_metadata
The metadata of the stylesheet (hash, size, filename)

theme

timezone
The timezone of the event

title

type

type_

tzinfo

url

url_shortcut
The URL shortcut for the event

visibility
The visibility depth in category overviews

class indico.modules.events.models.events.**EventType**
Bases: indico.util.struct.enum.RichIntEnum

conference = 3

lecture = 1

legacy_name

meeting = 2

class indico.modules.events.models.persons.**AuthorsSpeakersMixin**
Bases: object

primary_authors

secondary_authors

speakers

class indico.modules.events.models.persons.**EventPerson**(***kwargs*)
Bases: *indico.modules.users.models.users.PersonMixin*, flask_sqlalchemy.Model

A person inside an event, e.g. a speaker/author etc.

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

address

affiliation

classmethod create_from_user(*user*, *event=None*, *is_untrusted=False*)

email

event

event_id

first_name

classmethod for_user(*user*, *event=None*, *is_untrusted=False*)
Return EventPerson for a matching User in Event creating if needed

has_role(*role*, *obj*)
Whether the person has a role in the ACL list of a given object

id

invited_dt

is_untrusted

last_name

classmethod link_user_by_email(*user*)
Links all email-based persons matching the user's email addresses with the user.

Parameters **user** – A User object.

locator
Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named locator as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

merge_person_info(**args*, ***kwargs*)

classmethod merge_users(*target*, *source*)
Merge the EventPersons of two users.

Parameters

- **target** – The target user of the merge
- **source** – The user that is being merged into *target*

phone

principal

user

user_id

class indico.modules.events.models.persons.**EventPersonLink**(**args*, ***kwargs*)
Bases: *indico.modules.events.models.persons.PersonLinkBase*

Association between EventPerson and Event.

Chairperson or speaker (lecture)

display_order

event_id

id

is_submitter

object_relationship_name = u'event'

person

person_id

person_link_backref_name = u'event_links'

person_link_unique_columns = (u'event_id',)

class indico.modules.events.models.persons.**PersonLinkBase**(**args*, ***kwargs*)
Bases: *indico.modules.users.models.users.PersonMixin*, flask_sqlalchemy.Model

Base class for EventPerson associations.

address

affiliation

display_order = Column(None, Integer(), table=None, nullable=False, default=ColumnDefault(0))

display_order_key

email

first_name

id = Column(None, Integer(), table=None, primary_key=True, nullable=False)

last_name

object

object_relationship_name = None
The name of the relationship pointing to the object the person is linked to

person = <RelationshipProperty at 0x7faacb5e5af0; no key>

person_id = Column(None, Integer(), ForeignKey(u'events.persons.id'), table=None, nullable=False)

person_link_backref_name = None
The name of the backref on the *EventPerson*

person_link_unique_columns = None
The columns which should be included in the unique constraint.

phone

title

class indico.modules.events.models.persons.**PersonLinkDataMixin**
Bases: object

person_link_data

class indico.modules.events.models.principals.**EventPrincipal**(***kwargs*)
Bases: indico.core.db.sqlalchemy.principals.PrincipalRolesMixin, flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

allow_emails = True

allow_networks = True

email

event_id
The ID of the associated event

full_access

id
The ID of the acl entry

ip_network_group

ip_network_group_id

local_group

local_group_id

multipass_group_name

multipass_group_provider

principal_backref_name = u'in_event_acls'

principal_for = u'Event'

read_access

roles

type

unique_columns = (u'event_id',)

user

user_id

class indico.modules.events.models.references.**EventReference**(***kwargs*)
Bases: *indico.modules.events.models.references.ReferenceModelBase*

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

event_id

id

reference_backref_name = u'event_references'

reference_type

reference_type_id

value

class indico.modules.events.models.references.**ReferenceModelBase**(***kwargs*)
Bases: `flask_sqlalchemy.Model`

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

id = Column(None, Integer(), table=None, primary_key=True, nullable=False)

reference_backref_name = None
The name of the backref on the *ReferenceType*

reference_type = <RelationshipProperty at 0x7faacb8729e0; no key>

reference_type_id = Column(None, Integer(), ForeignKey(u'indico.reference_types.id'), table=None, nullable=False

url
The URL of the referenced entity.

`None` if no URL template is defined.

urn
The URN of the referenced entity.

`None` if no scheme is defined.

value = Column(None, String(), table=None, nullable=False)

class indico.modules.events.models.references.**ReferenceType**(***kwargs*)
Bases: flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

id
The unique ID of the reference type

locator
Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named locator as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The other locator can then be accessed by passing obj.locator.other to the code expecting an object with a locator.

name
The name of the referenced system

scheme
The scheme used to build an URN for the reference

url_template
A URL template to build a link to a referenced entity

class indico.modules.events.models.reviews.**ProposalCommentMixin**
Bases: object

can_edit(*user*)

timeline_item_type = u'comment'

class indico.modules.events.models.reviews.**ProposalGroupProxy**(*group*)
Bases: object

Represents the object that the proposals can be grouped by.

It provides all necessary methods for building the URLs, displaying the grouping information, etc.

full_title

full_title_attr = u'full_title'

locator
Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

title

title_attr = u'title'

class indico.modules.events.models.reviews.**ProposalMixin**
Bases: object

Classes that represent a proposal object should extend this class (ex: Abstract, Paper).

call_for_proposals_attr = None
Attribute to retrieve the object with access to the reviewing settings

can_comment(*user*)

can_review(*user*, *check_state=False*)

cfp

create_comment_endpoint = None

create_judgment_endpoint = None

create_review_endpoint = None

delete_comment_endpoint = None

edit_comment_endpoint = None

edit_review_endpoint = None

get_delete_comment_url(*comment*)

get_last_revision()

get_revisions()

get_save_comment_url(*comment=None*)

get_save_judgment_url()

get_save_review_url(*group=None*, *review=None*)

is_in_final_state

proposal_type = None
A unique identifier to handle rendering differences between proposal types

revisions_enabled = True
Whether there is support for multiple revisions per proposal or just one

class indico.modules.events.models.reviews.**ProposalReviewMixin**
Bases: object

Mixin for proposal reviews

Classes that represent a review of a proposal should extend this class (ex: AbstractReview, PaperReview).

can_edit(*user*)

group

group_attr = None
Object used to group reviews together

group_proxy_cls
Proxy class to provide the necessary properties and methods to the review grouping object

alias of *ProposalGroupProxy*

revision

revision_attr = None
The revision object that the review refers to

score

timeline_item_type = u'review'
A unique identifier to handle rendering differences between timeline items

class indico.modules.events.models.reviews.**ProposalRevisionMixin**
Bases: object

Properties and methods of a proposal revision.

get_reviewed_for_groups(*user*, *include_reviewed=False*)

get_reviewer_render_data(**args*, ***kwargs*)

get_reviews(*group=None*, *user=None*)

get_timeline(*user=None*)

proposal

proposal_attr = None
The attribute of the revision used to fetch the proposal object.

revisions_enabled = True
Whether the reviewing process supports multiple revisions per proposal. If set to false it is assumed that the reviewing process supports only one revision per proposal.

class indico.modules.events.models.series.**EventSeries**(***kwargs*)
Bases: flask_sqlalchemy.Model

A series of events.

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

id
The ID of the series

show_links
Whether to show links to the other events in the same series on the main event page.

show_sequence_in_title
Whether to show the sequence number of an event in its title on category display pages and on the main event page.

class indico.modules.events.models.settings.**EventSetting**(***kwargs*)
Bases: indico.core.settings.models.base.JSONSettingsBase, *indico.modules.events.models.settings.EventSettingsMixin*, flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

event

event_id

id

module

name

settings_backref_name = u'settings'

value

class indico.modules.events.models.settings.**EventSettingPrincipal**(***kwargs*)
Bases: indico.core.settings.models.base.PrincipalSettingsBase, *indico.modules.events.models.settings.EventSettingsMixin*, flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

email = None

event

event_id

extra_key_cols = (u'event_id',)

id

ip_network_group = None

ip_network_group_id = None

local_group

local_group_id

module

multipass_group_name

multipass_group_provider

name

principal_backref_name = u'in_event_settings_acls'

settings_backref_name = u'settings_principals'

type

user

user_id

class indico.modules.events.models.settings.**EventSettingsMixin**
Bases: object

event = <RelationshipProperty at 0x7faacb465848; no key>

event_id = Column(None, Integer(), ForeignKey(u'events.events.id'), table=None, nullable=False)

settings_backref_name = None

class indico.modules.events.models.static_list_links.**StaticListLink**(***kwargs*)
Bases: flask_sqlalchemy.Model

Display configuration data used in static links to listing pages.

This allows users to share links to listing pages in events while preserving e.g. column/filter configurations.

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

classmethod create(*event*, *type_*, *data*)
Create a new static list link.

If one exists with the same data, that link is used instead of creating a new one.

Parameters

- **event** – the *Event* for which to create the link
- **type** – the type of the link
- **data** – the data to associate with the link

Returns the newly created *StaticListLink*

created_dt

data

event

event_id

id

last_used_dt

classmethod load(*event*, *type_*, *uuid*)
Load the data associated with a link

Parameters

- **event** – the *Event* the link belongs to
- **type** – the type of the link
- **uuid** – the UUID of the link

Returns the link data or `None` if the link does not exist

type

uuid

Operations

indico.modules.events.operations.**clone_event**(*event*, *start_dt*, *cloners*, *category=None*)
Clone an event on a given date/time.

Runs all required cloners.

Parameters

- **start_dt** – The start datetime of the new event;
- **cloners** – A set containing the names of all enabled cloners;
- **category** – The *Category* the new event will be created in.

indico.modules.events.operations.**create_event**(**args*, ***kwargs*)
Create a new event.

Parameters

- **category** – The category in which to create the event
- **event_type** – An *EventType* value
- **data** – A dict containing data used to populate the event
- **add_creator_as_manager** – Whether the creator (current user) should be added as a manager
- **features** – A list of features that will be enabled for the event. If set, only those features will be used and the default feature set for the event type will be ignored.

indico.modules.events.operations.**create_event_references**(*event*, *data*)

indico.modules.events.operations.**create_reference_type**(*data*)

indico.modules.events.operations.**delete_reference_type**(*reference_type*)

indico.modules.events.operations.**lock_event**(*event*)

indico.modules.events.operations.**unlock_event**(*event*)

indico.modules.events.operations.**update_event**(*event*, *update_timetable=False*, ***data*)

indico.modules.events.operations.**update_event_protection**(*event*, *data*)

indico.modules.events.operations.**update_event_type**(*event*, *type_*)

indico.modules.events.operations.**update_reference_type**(*reference_type*, *data*)

Utilities

class indico.modules.events.util.**ListGeneratorBase**(*event*, *entry_parent=None*)
Bases: object

Base class for classes performing actions on Indico object lists.

Parameters

- **event** – The associated *Event*
- **entry_parent** – The parent of the entries of the list. If it's None, the parent is assumed to be the event itself.

default_list_config = None
The default list configuration dictionary

endpoint = None
The endpoint of the list management page

entry_parent = None
The parent object of the list items

event = None
The event the list is associated with

flash_info_message(*obj*)

generate_static_url()
Return a URL with a uuid referring to the list's configuration.

get_list_url(*uuid=None*, *external=False*)
Return the URL of the list management page.

list_link_type = None
Unique list identifier

static_items = None
Columns that originate from the list item's properties, relationships etc, but not from user defined fields (e.g. registration/contribution fields)

store_configuration()
Load the filters from the request and store them in the session.

class indico.modules.events.util.**ZipGeneratorMixin**
Mixin for RHs that generate zip with files

indico.modules.events.util.**create_event_logo_tmp_file**(*event*)
Creates a temporary file with the event's logo

indico.modules.events.util.**get_base_ical_parameters**(*user*, *detail*, *path*, *params=None*)
Returns a dict of all parameters expected by iCal template

indico.modules.events.util.**get_events_created_by**(*user*, *dt=None*)
Gets the IDs of events created by the user

Parameters

- **user** – A *User*
- **dt** – Only include events taking place on/after that date

Returns A set of event ids

indico.modules.events.util.**get_events_managed_by**(*user*, *dt=None*)
Gets the IDs of events where the user has management privs.

Parameters

- **user** – A *User*
- **dt** – Only include events taking place on/after that date

Returns A set of event ids

indico.modules.events.util.**get_events_with_linked_event_persons**(*user*, *dt=None*)
Returns a dict containing the event ids and role for all events where the user is a chairperson or (in case of a lecture) speaker.

Parameters

- **user** – A *User*
- **dt** – Only include events taking place on/after that date

indico.modules.events.util.**get_field_values**(*form_data*)
Split the form fields between custom and static

indico.modules.events.util.**get_object_from_args**(*args=None*)
Retrieves an event object from request arguments.

This utility is meant to be used in cases where the same controller can deal with objects attached to various parts of an event which use different URLs to indicate which object to use.

Parameters **args** – The request arguments. If unspecified, `request.view_args` is used.

Returns An `(object_type, event, object)` tuple. The event is always the `Event` associated with the object. The object may be an *Event*, *Session*, *Contribution* or *SubContribution*. If the object does not exist, `(object_type, None, None)` is returned.

indico.modules.events.util.**get_random_color**(*event*)

indico.modules.events.util.**get_theme**(*event*, *override_theme_id=None*)
Get the theme ID and whether it's an override.

This is useful for places where a user may specify a different timetable theme. If the override theme is not valid for the event, a message is flashed and an exception redirecting the user to the main event page is raised.

Raises **BadRequest** – if the override theme id is not valid

Returns a `(theme_id, is_override)` tuple

indico.modules.events.util.**register_event_time_change**(*event*)
Register a time-related change for an event

This is an internal helper function used in the model to record changes of the start time or end time. The changes are exposed through the *track_time_changes* contextmanager function.

indico.modules.events.util.**register_time_change**(*entry*)
Register a time-related change for a timetable entry

This is an internal helper function used in the models to record changes of the start time or duration. The changes are exposed through the *track_time_changes* contextmanager function.

indico.modules.events.util.**serialize_event_for_ical**(*event*, *detail_level*)

indico.modules.events.util.**serialize_event_person**(*person*)
Serialize EventPerson to JSON-like object

indico.modules.events.util.**serialize_person_link**(*person_link*)
Serialize PersonLink to JSON-like object

indico.modules.events.util.**set_custom_fields**(*obj*, *custom_fields_data*)

indico.modules.events.util.**track_time_changes**(**args*, ***kwds*)
Track time changes of event objects.

This provides a list of changes while the context manager was active and also triggers *times_changed* signals.

If the code running inside the `with` block of this context manager raises an exception, no signals will be triggered.

Parameters

- **auto_extend** – Whether entry parents will get their boundaries automatically extended or not. Passing `'start'` will extend only start datetime, `'end'` to extend only end datetime.
- **user** – The *User* that will trigger time changes.

indico.modules.events.util.**update_object_principals**(*obj*, *new_principals*, *read_access=False*, *full_access=False*, *role=None*)
Updates an object's ACL with a new list of principals

Exactly one argument out of *read_access*, *full_access* and *role* must be specified.

Parameters

- **obj** – The object to update. Must have `acl_entries`
- **new_principals** – The set containing the new principals
- **read_access** – Whether the read access ACL should be updated
- **full_access** – Whether the full access ACL should be updated
- **role** – The role ACL that should be updated

Settings

class indico.modules.events.settings.**EventACLProxy**(*proxy*)
Bases: indico.core.settings.proxy.ACLProxyBase

Proxy class for event-specific ACL settings

add_principal(*event*, **args*, ***kwargs*)
Adds a principal to an ACL

Parameters

- **event** – Event (or its ID)
- **name** – Setting name
- **principal** – A *User* or a *GroupProxy*

contains_user(*event*, **args*, ***kwargs*)
Checks if a user is in an ACL.

To pass this check, the user can either be in the ACL itself or in a group in the ACL.

Parameters

- **event** – Event (or its ID)

- **name** – Setting name
- **user** – A *User*

get(*event*, **args*, ***kwargs*)
Retrieves an ACL setting

Parameters

- **event** – Event (or its ID)
- **name** – Setting name

merge_users(*target*, *source*)
Replaces all ACL user entries for *source* with *target*

remove_principal(*event*, **args*, ***kwargs*)
Removes a principal from an ACL

Parameters

- **event** – Event (or its ID)
- **name** – Setting name
- **principal** – A *User* or a *GroupProxy*

set(*event*, **args*, ***kwargs*)
Replaces an ACL with a new one

Parameters

- **event** – Event (or its ID)
- **name** – Setting name
- **acl** – A set containing principals (users/groups)

class indico.modules.events.settings.**EventSettingProperty**(*proxy*, *name*, *default=<object object>*, *attr=None*)
Bases: indico.core.settings.proxy.SettingProperty

attr = u'event'

class indico.modules.events.settings.**EventSettingsProxy**(*module*, *defaults=None*, *strict=True*, *acls=None*, *converters=None*)
Bases: indico.core.settings.proxy.SettingsProxyBase

Proxy class to access event-specific settings for a certain module

acl_proxy_class
alias of *EventACLProxy*

delete(*event*, **args*, ***kwargs*)
Deletes settings.

Parameters

- **event** – Event (or its ID)
- **names** – One or more names of settings to delete

delete_all(*event*, **args*, ***kwargs*)
Deletes all settings.

Parameters **event** – Event (or its ID)

get(*event*, **args*, ***kwargs*)
Retrieves the value of a single setting.

Parameters

- **event** – Event (or its ID)
- **name** – Setting name
- **default** – Default value in case the setting does not exist

Returns The settings's value or the default value

get_all(*event*, **args*, ***kwargs*)
Retrieves all settings

Parameters

- **event** – Event (or its ID)
- **no_defaults** – Only return existing settings and ignore defaults.

Returns Dict containing the settings

query
Returns a query object filtering by the proxy's module.

set(*event*, **args*, ***kwargs*)
Sets a single setting.

Parameters

- **event** – Event (or its ID)
- **name** – Setting name
- **value** – Setting value; must be JSON-serializable

set_multi(*event*, **args*, ***kwargs*)
Sets multiple settings at once.

Parameters

- **event** – Event (or its ID)
- **items** – Dict containing the new settings

class indico.modules.events.settings.**ThemeSettingsProxy**
Bases: object

defaults

get_themes_for(**args*, ***kwargs*)

settings

themes

indico.modules.events.settings.**event_or_id**(*f*)

Abstract

Todo

Docstrings (module, models, operations, utilities, settings)

Models

class indico.modules.events.abstracts.models.abstracts.**Abstract**(***kwargs*)

Bases: *indico.modules.events.models.reviews.ProposalMixin*, *indico.modules.events.models.reviews.ProposalRevisionMixin*, indico.core.db.sqlalchemy.descriptions.DescriptionMixin, *indico.modules.events.contributions.models.contributions.CustomFieldsMixin*, *indico.modules.events.models.persons.AuthorsSpeakersMixin*, flask_sqlalchemy.Model

Represents an abstract that can be associated to a Contribution.

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

accepted_contrib_type

accepted_contrib_type_id

accepted_track

accepted_track_id

call_for_proposals_attr = u'cfa'

can_access(*user*)

can_comment(*user*, *check_state=False*)

can_convene(*user*)

can_edit(*user*)

can_judge(*user*, *check_state=False*)

can_review(*user*, *check_state=False*)

can_see_reviews(*user*)

can_withdraw(*user*, *check_state=False*)

candidate_contrib_types

candidate_tracks

create_comment_endpoint = u'abstracts.comment_abstract'

create_judgment_endpoint = u'abstracts.judge_abstract'

create_review_endpoint = u'abstracts.review_abstract'

data_by_field

default_render_mode = 2

delete_comment_endpoint = u'abstracts.delete_abstract_comment'

duplicate_of

duplicate_of_id

edit_comment_endpoint = u'abstracts.edit_abstract_comment'

edit_review_endpoint = u'abstracts.edit_review'

edit_track_mode

event

event_id

field_values
Data stored in abstract/contribution fields

friendly_id

get_reviewed_for_groups(*user*, *include_reviewed=False*)

get_timeline(*user=None*)

get_track_question_scores()

get_track_reviewing_state(*track*)

get_track_score(*track*)

id

is_deleted

is_in_final_state

judge
User who judged the abstract

judge_id
ID of the user who judged the abstract

judgment_comment

judgment_dt

locator
Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

marshmallow_aliases = {u'_description': u'content'}

merged_into

merged_into_id

modified_by

modified_by_id

modified_dt

person_links
Persons associated with this abstract

possible_render_modes = set([<RenderMode.markdown: 2>])

proposal_type = u'abstract'

public_state

render_mode = 2

reset_state()

reviewed_for_tracks

reviewing_state

revisions_enabled = False

score

state

submission_comment

submitted_contrib_type

submitted_contrib_type_id

submitted_dt

submitted_for_tracks

submitter
User who submitted the abstract

submitter_id
ID of the user who submitted the abstract

title

user_owns(*user*)

verbose_title

class indico.modules.events.abstracts.models.abstracts.**AbstractPublicState**
Bases: indico.util.struct.enum.RichIntEnum

accepted = 3

awaiting = -1

duplicate = 6

merged = 5

rejected = 4

under_review = -2

withdrawn = 2

class indico.modules.events.abstracts.models.abstracts.**AbstractReviewingState**
Bases: indico.util.struct.enum.RichIntEnum

conflicting = 3

in_progress = 1

mixed = 5

negative = 4

not_started = 0

positive = 2

class indico.modules.events.abstracts.models.abstracts.**AbstractState**
Bases: indico.util.struct.enum.RichIntEnum

accepted = 3

duplicate = 6

merged = 5

rejected = 4

submitted = 1

withdrawn = 2

class indico.modules.events.abstracts.models.abstracts.**EditTrackMode**
Bases: int, indico.util.struct.enum.IndicoEnum

both = 1

none = 0

reviewed_for = 2

class indico.modules.events.abstracts.models.call_for_abstracts.**CallForAbstracts**(*event*)
Bases: object

Proxy class to facilitate access to the call for abstracts settings

allow_attachments

allow_comments

allow_contributors_in_comments

allow_convener_judgment

announcement

can_edit_abstracts(*user*)

can_submit_abstracts(*user*)

close()

end_dt

has_ended

has_started

is_open

judgment_instructions

modification_end_dt

modification_ended

open()

rating_range

reviewing_instructions

schedule(*start_dt*, *end_dt*, *modification_end_dt*)

start_dt

submission_instructions

class indico.modules.events.abstracts.models.comments.**AbstractComment**(***kwargs*)

Bases: *indico.modules.events.models.reviews.ProposalCommentMixin*, indico.core.db.sqlalchemy.review_comments.ReviewCommentMixin, flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

abstract

abstract_id

can_edit(*user*)

can_view(*user*)

created_dt

id

is_deleted

locator

Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

marshmallow_aliases = {u'_text': u'text'}

modified_by

modified_by_id

modified_dt

render_mode = 2

user

user_backref_name = u'abstract_comments'

user_id

user_modified_backref_name = u'modified_abstract_comments'

visibility

class indico.modules.events.abstracts.models.email_logs.**AbstractEmailLogEntry**(***kwargs*)
Bases: flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

abstract

abstract_id

body

classmethod create_from_email(*email_data*, *email_tpl*, *user=None*)
Create a new log entry from the data used to send an email

Parameters

- **email_data** – email data as returned from *make_email*
- **email_tpl** – the abstract email template that created the email
- **user** – the user who performed the action causing the notification

data

email_template

email_template_id

id

recipients

sent_dt

subject

user

user_id

class indico.modules.events.abstracts.models.email_templates.**AbstractEmailTemplate**(***kwargs*)
Bases: flask_sqlalchemy.Model

Represents an email template for abstracts notifications.

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

body
The body of the template

event

event_id

extra_cc_emails
List of extra email addresses to be added as CC in the email

id

include_authors
Whether to include authors' email addresses as To for emails

include_coauthors
Whether to include co-authors' email addresses as CC for emails

include_submitter
Whether to include the submitter's email address as To for emails

locator
Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

position
The relative position of the template in the list of templates

reply_to_address
The address to use as Reply-To in the email

rules
Conditions need to be met to send the email

stop_on_match
Whether to stop checking the rest of the conditions when a match is found

subject
The subject of the email

title

class indico.modules.events.abstracts.models.fields.**AbstractFieldValue**(***kwargs*)
Bases: *indico.modules.events.contributions.models.fields.ContributionFieldValueBase*

Store a field values related to abstracts.

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

abstract_id

contribution_field

contribution_field_backref_name = u'abstract_values'

contribution_field_id

data

class indico.modules.events.abstracts.models.files.**AbstractFile**(***kwargs*)
Bases: indico.core.storage.models.StoredFileMixin, flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

abstract

abstract_id

add_file_date_column = False

content_type
The MIME type of the file

created_dt = None

filename
The name of the file

id

locator

md5
An MD5 hash of the file.

Automatically assigned when *save()* is called.

size
The size of the file (in bytes).

Automatically assigned when *save()* is called.

storage_backend

storage_file_id

class indico.modules.events.abstracts.models.persons.**AbstractPersonLink**(**args*, ***kwargs*)
Bases: *indico.modules.events.models.persons.PersonLinkBase*

Association between EventPerson and Abstract.

abstract_id

author_type

display_order

id

is_speaker

locator

Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

object_relationship_name = u'abstract'

person

person_id

person_link_backref_name = u'abstract_links'

person_link_unique_columns = (u'abstract_id',)

class indico.modules.events.abstracts.models.review_questions.**AbstractReviewQuestion**(***kwargs*)

Bases: indico.core.db.sqlalchemy.review_questions.ReviewQuestionMixin, flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

event

event_backref_name = u'abstract_review_questions'

event_id

id

is_deleted

no_score

position

text

class indico.modules.events.abstracts.models.review_ratings.**AbstractReviewRating**(***kwargs*)

Bases: indico.core.db.sqlalchemy.review_ratings.ReviewRatingMixin, flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

id

question

question_class
alias of `AbstractReviewQuestion`

question_id

review

review_class
alias of `AbstractReview`

review_id

value

class indico.modules.events.abstracts.models.reviews.**AbstractAction**
Bases: indico.util.struct.enum.RichIntEnum

accept = 1

change_tracks = 3

mark_as_duplicate = 4

merge = 5

reject = 2

class indico.modules.events.abstracts.models.reviews.**AbstractCommentVisibility**
Bases: indico.util.struct.enum.RichIntEnum

Most to least restrictive visibility for abstract comments

contributors = 4

conveners = 2

judges = 1

reviewers = 3

users = 5

class indico.modules.events.abstracts.models.reviews.**AbstractReview**(***kwargs*)
Bases: *indico.modules.events.models.reviews.ProposalReviewMixin*, indico.core.db.sqlalchemy.descriptions.RenderModeMixin, flask_sqlalchemy.Model

Represents an abstract review, emitted by a reviewer

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

abstract

abstract_id

can_edit(*user*, *check_state=False*)

can_view(*user*)

comment

created_dt

default_render_mode = 2

group_attr = u'track'

id

locator

Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

marshmallow_aliases = {u'_comment': u'comment'}

modified_dt

possible_render_modes = set([<RenderMode.markdown: 2>])

proposed_action

proposed_contribution_type

proposed_contribution_type_id

proposed_related_abstract

proposed_related_abstract_id

proposed_tracks

render_mode = 2

revision_attr = u'abstract'

score

track

track_id

user

user_id

visibility

Operations

indico.modules.events.abstracts.operations.**add_abstract_files**(*abstract*, *files*, *log_action=True*)

indico.modules.events.abstracts.operations.**close_cfa**(*event*)

indico.modules.events.abstracts.operations.**create_abstract**(*event*, *abstract_data*, *custom_fields_data=None*, *send_notifications=False*)

indico.modules.events.abstracts.operations.**create_abstract_comment**(*abstract*, *comment_data*)

indico.modules.events.abstracts.operations.**create_abstract_review**(*abstract*, *track*, *user*, *review_data*, *questions_data*)

indico.modules.events.abstracts.operations.**delete_abstract**(*abstract*, *delete_contrib=False*)

indico.modules.events.abstracts.operations.**delete_abstract_comment**(*comment*)

indico.modules.events.abstracts.operations.**delete_abstract_files**(*abstract*, *files*)

indico.modules.events.abstracts.operations.**judge_abstract**(*abstract*, *abstract_data*, *judgment*, *judge*, *contrib_session=None*, *merge_persons=False*, *send_notifications=False*)

indico.modules.events.abstracts.operations.**open_cfa**(*event*)

indico.modules.events.abstracts.operations.**reset_abstract_state**(*abstract*)

indico.modules.events.abstracts.operations.**schedule_cfa**(*event*, *start_dt*, *end_dt*, *modification_end_dt*)

indico.modules.events.abstracts.operations.**update_abstract**(*abstract*, *abstract_data*, *custom_fields_data=None*)

indico.modules.events.abstracts.operations.**update_abstract_comment**(*comment*, *comment_data*)

indico.modules.events.abstracts.operations.**update_abstract_review**(*review*, *review_data*, *questions_data*)

indico.modules.events.abstracts.operations.**update_reviewed_for_tracks**(*abstract*, *tracks*)

indico.modules.events.abstracts.operations.**withdraw_abstract**(*abstract*)

Utilities

indico.modules.events.abstracts.util.**build_default_email_template**(*event*, *tpl_type*)

Build a default e-mail template based on a notification type provided by the user.

indico.modules.events.abstracts.util.**clear_boa_cache**(*event*)

Delete the cached book of abstract

indico.modules.events.abstracts.util.**create_boa**(*event*)

Create the book of abstracts if necessary

Returns The path to the PDF file

indico.modules.events.abstracts.util.**create_mock_abstract**(**args*, ***kwargs*)

Create a mock abstract that can be used in previews.

Brace for geek references.

indico.modules.events.abstracts.util.**generate_spreadsheet_from_abstracts**(*abstracts*, *static_item_ids*, *dynamic_items*)

Generates a spreadsheet data from a given abstract list.

Parameters

- **abstracts** – The list of abstracts to include in the file
- **static_item_ids** – The abstract properties to be used as columns
- **dynamic_items** – Contribution fields as extra columns

indico.modules.events.abstracts.util.**get_events_with_abstract_persons**(*user*, *dt=None*)

Return a dict of event ids and the abstract submission related roles the user has in that event.

Parameters

- **user** – A *User*
- **dt** – Only include events taking place on/after that date

indico.modules.events.abstracts.util.**get_events_with_abstract_reviewer_convener**(*user*, *dt=None*)

Return a dict of event ids and the abstract reviewing related roles the user has in that event.

Parameters

- **user** – A *User*
- **dt** – Only include events taking place on/after that date

indico.modules.events.abstracts.util.**get_roles_for_event**(*event*)

Return a dictionary of all abstract reviewing roles for this event.

Parameters **event** – the actual event object.

Returns A dictionary in the form `{track: {role: [users]}}`

indico.modules.events.abstracts.util.**get_track_reviewer_abstract_counts**(*event*, *user*)

Get the numbers of abstracts per track for a specific user.

Note that this does not take into account if the user is a reviewer for a track; it just checks whether the user has reviewed an abstract in a track or not.

Returns A dict mapping tracks to dicts containing the counts.

indico.modules.events.abstracts.util.**get_user_abstracts**(*event*, *user*)
Get the list of abstracts where the user is a reviewer/convener

indico.modules.events.abstracts.util.**get_user_tracks**(*event*, *user*)
Get the list of tracks where the user is a reviewer/convener

indico.modules.events.abstracts.util.**get_visible_reviewed_for_tracks**(*abstract*, *user*)

indico.modules.events.abstracts.util.**has_user_tracks**(*event*, *user*)

indico.modules.events.abstracts.util.**make_abstract_form**(*event*, *notification_option=False*, *management=False*)
Extends the abstract WTForm to add the extra fields.

Each extra field will use a field named `custom_ID`.

Parameters

- **event** – The *Event* for which to create the abstract form.
- **notification_option** – Whether to add a field to the form to disable triggering notifications for the abstract submission.

:param management Whether it's a manager using the abstract form :return: An *AbstractForm* subclass.

Placeholders

class indico.modules.events.abstracts.placeholders.**EventTitlePlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'The title of the event'

name = u'event_title'

classmethod render(*abstract*)

class indico.modules.events.abstracts.placeholders.**EventURLPlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'The URL of the event'

name = u'event_url'

classmethod render(*abstract*)

class indico.modules.events.abstracts.placeholders.**AbstractIDPlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'The ID of the abstract'

name = u'abstract_id'

classmethod render(*abstract*)

class indico.modules.events.abstracts.placeholders.**AbstractTitlePlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'The title of the abstract'

name = u'abstract_title'

classmethod render(*abstract*)

class indico.modules.events.abstracts.placeholders.**AbstractURLPlaceholder**
Bases: indico.util.placeholders.Placeholder

advanced = True

description = lu'The direct URL of the abstract'

name = u'abstract_url'

classmethod render(*abstract*)

class indico.modules.events.abstracts.placeholders.**AbstractTrackPlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'The name of the destination track'

name = u'abstract_track'

classmethod render(*abstract*)

class indico.modules.events.abstracts.placeholders.**AbstractSessionPlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'The name of the destination session'

name = u'abstract_session'

classmethod render(*abstract*)

class indico.modules.events.abstracts.placeholders.**PrimaryAuthorsPlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'The names of the primary authors (separated by commas)'

name = u'primary_authors'

classmethod render(*abstract*)

class indico.modules.events.abstracts.placeholders.**CoAuthorsPlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'The names of the co-authors (separated by commas)'

name = u'co_authors'

classmethod render(*abstract*)

class indico.modules.events.abstracts.placeholders.**SubmitterNamePlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'The full name of the submitter, no title'

name = u'submitter_name'

classmethod render(*abstract*)

class indico.modules.events.abstracts.placeholders.**SubmitterFirstNamePlaceholder**
Bases: indico.util.placeholders.Placeholder

advanced = True

description = lu'The first name of the submitter'

name = u'submitter_first_name'

classmethod render(*abstract*)

class indico.modules.events.abstracts.placeholders.**SubmitterLastNamePlaceholder**
Bases: indico.util.placeholders.Placeholder

advanced = True

description = lu'The last name of the submitter'

name = u'submitter_last_name'

classmethod **render**(*abstract*)

class indico.modules.events.abstracts.placeholders.**SubmitterTitlePlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'The title of the submitter (Dr., Prof., etc...)'

name = u'submitter_title'

classmethod **render**(*abstract*)

class indico.modules.events.abstracts.placeholders.**TargetAbstractIDPlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'The ID of the target abstract (merge)'

name = u'target_abstract_id'

classmethod **render**(*abstract*)

class indico.modules.events.abstracts.placeholders.**TargetAbstractTitlePlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'The title of the target abstract (merge)'

name = u'target_abstract_title'

classmethod **render**(*abstract*)

class indico.modules.events.abstracts.placeholders.**TargetSubmitterNamePlaceholder**
Bases: indico.util.placeholders.Placeholder

advanced = True

description = lu"The full name of the target abstract's submitter, no title (merge)"

name = u'target_submitter_name'

classmethod **render**(*abstract*)

class indico.modules.events.abstracts.placeholders.**TargetSubmitterFirstNamePlaceholder**
Bases: indico.util.placeholders.Placeholder

advanced = True

description = lu"The first name of the target abstract's submitter (merge)"

name = u'target_submitter_first_name'

classmethod **render**(*abstract*)

class indico.modules.events.abstracts.placeholders.**TargetSubmitterLastNamePlaceholder**
Bases: indico.util.placeholders.Placeholder

advanced = True

description = lu"The last name of the target abstract's submitter (merge)"

name = u'target_submitter_last_name'

classmethod **render**(*abstract*)

class indico.modules.events.abstracts.placeholders.**JudgmentCommentPlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'Comments written by event organizer (upon final decision)'

name = u'judgment_comment'

classmethod render (*abstract*)

class indico.modules.events.abstracts.placeholders.**ContributionTypePlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'The contribution type that is associated to the abstract'

name = u'contribution_type'

classmethod render (*abstract*)

class indico.modules.events.abstracts.placeholders.**ContributionURLPlaceholder**
Bases: indico.util.placeholders.Placeholder

advanced = True

description = lu'Contribution URL'

name = u'contribution_url'

classmethod render (*abstract*)

Settings

class indico.modules.events.abstracts.settings.**BOACorrespondingAuthorType**
Bases: indico.util.struct.enum.RichEnum

none = u'none'

speakers = u'speakers'

submitter = u'submitter'

class indico.modules.events.abstracts.settings.**BOASortField**
Bases: indico.util.struct.enum.RichEnum

abstract_title = u'title'

id = u'id'

schedule = u'schedule'

session_title = u'session_title'

speaker = u'speaker'

Agreement

Todo

Docstrings (module, models, utilities)

Models

class indico.modules.events.agreements.models.agreements.**Agreement**(***kwargs*)
Bases: flask_sqlalchemy.Model

Agreements between a person and Indico

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

accept(*from_ip*, *reason=None*, *on_behalf=False*)

accepted

attachment
Attachment

attachment_filename
Filename and extension of the attachment

belongs_to(*person*)

static create_from_data(*event*, *type_*, *person*)

data
Definition-specific data of the agreement

definition

event
The Event this agreement is associated with

event_id
ID of the event

id
Entry ID

identifier
Unique identifier within the event and type

is_orphan()

locator

pending

person_email
Email of the person agreeing

person_name
Full name of the person agreeing

reason
Explanation as to why the agreement was accepted/rejected

reject(*from_ip*, *reason=None*, *on_behalf=False*)

rejected

render(*form*, ***kwargs*)

reset()

signed_dt
The date and time the agreement was signed

signed_from_ip
The IP from which the agreement was signed

signed_on_behalf

state
A *AgreementState*

timestamp
The date and time the agreement was created

type
Type of agreement

user
The user this agreement is linked to

user_id
ID of a linked user

uuid
Entry universally unique ID

class indico.modules.events.agreements.models.agreements.**AgreementState**
Bases: indico.util.struct.enum.RichIntEnum

accepted = 1

accepted_on_behalf = 3
agreement accepted on behalf of the person

pending = 0

rejected = 2

rejected_on_behalf = 4
agreement rejected on behalf of the person

Utilities

indico.modules.events.agreements.util.**get_agreement_definitions**()

indico.modules.events.agreements.util.**send_new_agreements**(*event*, *name*, *people*, *email_body*, *cc_addresses*, *from_address*)
Creates and send agreements for a list of people on a given event.

Parameters

- **event** – The *Event* associated with the agreement
- **name** – The agreement type matcing a AgreementDefinition name
- **people** – The list of people for whom agreements will be created
- **email_body** – The body of the email
- **cc_addresses** – Email addresses to send CCs to
- **from_address** – Email address of the sender

Placeholders

class indico.modules.events.agreements.placeholders.**AgreementLinkPlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'Link to the agreement page'

name = u'agreement_link'

classmethod render (*definition*, *agreement*)

required = True

class indico.modules.events.agreements.placeholders.**PersonNamePlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'Name of the person'

name = u'person_name'

classmethod render (*definition*, *agreement*)

Contribution

Todo

Docstrings (module, models, operations, utilities)

Models

class indico.modules.events.contributions.models.contributions.**Contribution** (***kwargs*)
Bases: indico.core.db.sqlalchemy.descriptions.DescriptionMixin, indico.core.db.sqlalchemy.protection.ProtectionManagersMixin, indico.core.db.sqlalchemy.locations.LocationMixin, indico.core.db.sqlalchemy.attachments.AttachedItemsMixin, indico.core.db.sqlalchemy.notes.AttachedNotesMixin, *indico.modules.events.models.persons.PersonLinkDataMixin*, *indico.modules.events.models.persons.AuthorsSpeakersMixin*, *indico.modules.events.contributions.models.contributions.CustomFieldsMixin*, flask_sqlalchemy.Model

ATTACHMENT_FOLDER_ID_COLUMN = u'contribution_id'

PRELOAD_EVENT_ATTACHED_ITEMS = True

PRELOAD_EVENT_NOTES = True

abstract

abstract_id

access_key = None

acl_entries

allow_relationship_preloading = True

board_number

can_manage (*user*, *role=None*, *allow_admin=True*, *check_parent=True*, *explicit_role=False*)

default_render_mode = 2

disallowed_protection_modes = frozenset([])

duration

duration_display
The displayed duration of the contribution.

This is the duration of the poster session if applicable, otherwise the duration of the contribution itself.

duration_poster

end_dt

end_dt_display
The displayed end time of the contribution.

This is the end time of the poster session if applicable, otherwise the end time of the contribution itself.

end_dt_poster

event

event_id

field_values
Data stored in abstract/contribution fields

friendly_id
The human-friendly ID for the contribution

get_non_inheriting_objects()
Get a set of child objects that do not inherit protection.

id

inherit_location

inheriting_have_acl = True

is_deleted

is_paper_reviewer(*user*)

is_scheduled

is_user_associated(*user*, *check_abstract=False*)

keywords

location_backref_name = u'contributions'

location_parent

locator
Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

own_address

own_no_access_contact = None

own_room

own_room_id

own_room_name

own_venue

own_venue_id

own_venue_name

paper

paper_content_reviewers
Paper content reviewers

paper_judges
Paper reviewing judges

paper_layout_reviewers
Paper layout reviewers

pending_paper_files
Paper files not submitted for reviewing

person_links
Persons associated with this contribution

possible_render_modes = set([<RenderMode.html: 1>, <RenderMode.markdown: 2>])

classmethod preload_acl_entries(*event*)

protection_mode

protection_parent

references
External references associated with this contribution

render_mode

session

session_block

session_block_id

session_id

start_dt

start_dt_display
The displayed start time of the contribution.

This is the start time of the poster session if applicable, otherwise the start time of the contribution itself.

start_dt_poster

subcontribution_count

subcontributions

submitters

title

track

track_id

type

type_id

verbose_title

class indico.modules.events.contributions.models.contributions.**CustomFieldsMixin**
Bases: object

Methods to process custom field data.

get_field_value(*field_id*, *raw=False*)

set_custom_field(*field_id*, *field_value*)

class indico.modules.events.contributions.models.fields.**ContributionField**(***kwargs*)
Bases: flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

description

event

event_id

field

field_data

field_type

filter_choices

id

is_active

is_required

legacy_id

locator
Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

mgmt_field

position

title

class indico.modules.events.contributions.models.fields.**ContributionFieldValue**(***kwargs*)
Bases: *indico.modules.events.contributions.models.fields.ContributionFieldValueBase*

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

contribution_field

contribution_field_backref_name = u'contribution_values'

contribution_field_id

contribution_id

data

class indico.modules.events.contributions.models.fields.**ContributionFieldValueBase**(***kwargs*)
Bases: `flask_sqlalchemy.Model`

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

contribution_field = <RelationshipProperty at 0x7faacc300e20; no key>

contribution_field_backref_name = None
The name of the backref on the *ContributionField*

contribution_field_id

data = Column(None, JSON(astext_type=Text()), table=None, nullable=False)

friendly_data

class indico.modules.events.contributions.models.persons.**AuthorType**
Bases: int, indico.util.struct.enum.IndicoEnum

classmethod get_highest(**types*)

none = 0

primary = 1

secondary = 2

class indico.modules.events.contributions.models.persons.**ContributionPersonLink**(**args*, ***kwargs*)

Bases: *indico.modules.events.models.persons.PersonLinkBase*

Association between EventPerson and Contribution.

author_type

contribution_id

display_order

id

is_author

is_speaker

is_submitter

locator
Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

object_relationship_name = u'contribution'

person

person_id

person_link_backref_name = u'contribution_links'

person_link_unique_columns = (u'contribution_id',)

class indico.modules.events.contributions.models.persons.**SubContributionPersonLink**(**args*, ***kwargs*)

Bases: *indico.modules.events.models.persons.PersonLinkBase*

Association between EventPerson and SubContribution.

author_type = 0

display_order

id

is_speaker = True

object_relationship_name = u'subcontribution'

person

person_id

person_link_backref_name = u'subcontribution_links'

person_link_unique_columns = (u'subcontribution_id',)

subcontribution_id

class indico.modules.events.contributions.models.principals.**ContributionPrincipal**(***kwargs*)
Bases: indico.core.db.sqlalchemy.principals.PrincipalRolesMixin, flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

allow_emails = True

contribution_id
The ID of the associated contribution

disallowed_protection_modes = frozenset([])

email

full_access

id
The ID of the acl entry

ip_network_group = None

ip_network_group_id = None

local_group

local_group_id

multipass_group_name

multipass_group_provider

principal_backref_name = u'in_contribution_acls'

principal_for = u'Contribution'

read_access

roles

type

unique_columns = (u'contribution_id',)

user

user_id

class indico.modules.events.contributions.models.references.**ContributionReference**(***kwargs*)
Bases: *indico.modules.events.models.references.ReferenceModelBase*

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

contribution_id

id

reference_backref_name = u'contribution_references'

reference_type

reference_type_id

value

class indico.modules.events.contributions.models.references.**SubContributionReference**(***kwargs*)
Bases: *indico.modules.events.models.references.ReferenceModelBase*

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

id

reference_backref_name = u'subcontribution_references'

reference_type

reference_type_id

subcontribution_id

value

class indico.modules.events.contributions.models.subcontributions.**SubContribution**(***kwargs*)
Bases: indico.core.db.sqlalchemy.descriptions.DescriptionMixin, indico.core.db.sqlalchemy.attachments.AttachedItemsMixin, indico.core.db.sqlalchemy.notes.AttachedNotesMixin, flask_sqlalchemy.Model

ATTACHMENT_FOLDER_ID_COLUMN = u'subcontribution_id'

PRELOAD_EVENT_ATTACHED_ITEMS = True

PRELOAD_EVENT_NOTES = True

can_access(*user*, ***kwargs*)

can_manage(*user*, *role=None*, ***kwargs*)

contribution_id

default_render_mode = 2

duration

event

friendly_id
The human-friendly ID for the sub-contribution

get_access_list()

get_manager_list(*recursive=False*)

id

is_deleted

is_protected

location_parent

locator
Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

person_links
Persons associated with this contribution

position

possible_render_modes = set([<RenderMode.html: 1>, <RenderMode.markdown: 2>])

references
External references associated with this contribution

render_mode

session
Convenience property so all event entities have it

speakers

timetable_entry
Convenience property so all event entities have it

title

class indico.modules.events.contributions.models.types.**ContributionType**(***kwargs*)
Bases: flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

description

event

event_id

id

locator

Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

name

Operations

indico.modules.events.contributions.operations.**create_contribution**(*event*, *contrib_data*, *custom_fields_data=None*, *session_block=None*, *extend_parent=False*)

indico.modules.events.contributions.operations.**create_contribution_from_abstract**(**args*, ***kwargs*)

indico.modules.events.contributions.operations.**create_subcontribution**(*contrib*, *data*)

indico.modules.events.contributions.operations.**delete_contribution**(*contrib*)

indico.modules.events.contributions.operations.**delete_subcontribution**(*subcontrib*)

indico.modules.events.contributions.operations.**update_contribution**(**args*, ***kwargs*)

Update a contribution

Parameters

- **contrib** – The *Contribution* to update
- **contrib_data** – A dict containing the data to update
- **custom_fields_data** – A dict containing the data for custom fields.

Returns A dictionary containing information related to the update. *unscheduled* will be true if the modification resulted in the contribution being unscheduled. In this case *undo_unschedule* contains the necessary data to re-schedule it (undoing the session change causing it to be unscheduled)

indico.modules.events.contributions.operations.**update_subcontribution**(*subcontrib*, *data*)

Utilities

indico.modules.events.contributions.util.**contribution_type_row**(*contrib_type*)

indico.modules.events.contributions.util.**generate_spreadsheet_from_contributions**(*contributions*)
Return a tuple consisting of spreadsheet columns and respective contribution values

indico.modules.events.contributions.util.**get_contribution_ical_file**(*contrib*)

indico.modules.events.contributions.util.**get_contributions_with_user_as_submitter**(*event*, *user*)
Get a list of contributions in which the *user* has submission rights

indico.modules.events.contributions.util.**get_events_with_linked_contributions**(*user*, *dt=None*)
Returns a dict with keys representing event_id and the values containing data about the user rights for contributions within the event

Parameters

- **user** – A *User*
- **dt** – Only include events taking place on/after that date

indico.modules.events.contributions.util.**has_contributions_with_user_as_submitter**(*event*, *user*)

indico.modules.events.contributions.util.**make_contribution_form**(*event*)
Extends the contribution WTForm to add the extra fields.

Each extra field will use a field named `custom_ID`.

Parameters **event** – The *Event* for which to create the contribution form.

Returns A *ContributionForm* subclass.

indico.modules.events.contributions.util.**serialize_contribution_for_ical**(*contrib*)

indico.modules.events.contributions.util.**serialize_contribution_person_link**(*person_link*, *is_submitter=None*)
Serialize ContributionPersonLink to JSON-like object

Feature

Todo

Docstrings (module, utilities)

Utilities

indico.modules.events.features.util.**format_feature_names**(*names*)

indico.modules.events.features.util.**get_disallowed_features**(*event*)
Get a set containing the names of features which are not available for an event.

indico.modules.events.features.util.**get_enabled_features**(*event*, *only_explicit=False*)
Returns a set of enabled feature names for an event

indico.modules.events.features.util.**get_feature_definition**(*name*)
Gets a feature definition

indico.modules.events.features.util.**get_feature_definitions**()
Gets a dict containing all feature definitions

indico.modules.events.features.util.**is_feature_enabled**(*event*, *name*)
Checks if a feature is enabled for an event.

Parameters

- **event** – The event (or event ID) to check.
- **name** – The name of the feature.

indico.modules.events.features.util.**require_feature**(*event*, *name*)
Raises a NotFound error if a feature is not enabled

Parameters

- **event** – The event (or event ID) to check.
- **name** – The name of the feature.

indico.modules.events.features.util.**set_feature_enabled**(*event*, *name*, *state*)
Enables/disables a feature for an event

Parameters

- **event** – The event.
- **name** – The name of the feature.
- **state** – If the feature is enabled or not.

Returns Boolean indicating if the state of the feature changed.

Layout

Todo

Docstrings (module, models, utilities)

Models

class indico.modules.events.layout.models.images.**ImageFile**(***kwargs*)
Bases: indico.core.storage.models.StoredFileMixin, flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

content_type
The MIME type of the file

created_dt
The date/time when the file was uploaded

event

event_id
The event the image belongs to

filename
The name of the file

id
The ID of the file

locator

md5
An MD5 hash of the file.

Automatically assigned when *save()* is called.

size
The size of the file (in bytes).

Automatically assigned when *save()* is called.

storage_backend

storage_file_id

version_of = None

class indico.modules.events.layout.models.menu.**EventPage**(***kwargs*)
Bases: flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

event
The Event which contains the page

event_id
The ID of the event which contains the page

html
The rendered HTML of the page

id
The ID of the page

is_default

locator

class indico.modules.events.layout.models.menu.**MenuEntry**(***kwargs*)
Bases: *indico.modules.events.layout.models.menu.MenuEntryMixin*, flask_sqlalchemy.Model

children
The children menu entries and parent backref

event
The Event containing the menu entry

event_id
The ID of the event which contains the menu

static get_for_event(*event*)

id
The ID of the menu entry

insert(*parent*, *position*)

is_enabled
Whether the entry is visible in the event's menu

is_root

link_url
The target URL of a custom link

move(*to*)

name
The name of the menu entry (to uniquely identify a default entry for a given event)

new_tab
Whether the menu entry should be opened in a new tab or window

page
The page of the menu entry

page_id
The page ID if the entry is a page

parent_id
The ID of the parent menu entry (NULL if root menu entry)

plugin
The name of the plugin from which the entry comes from (NULL if the entry does not come from a plugin)

position
The relative position of the entry in the menu

title
The title of the menu entry (to be displayed to the user)

type
The type of the menu entry

class indico.modules.events.layout.models.menu.**MenuEntryMixin**(***kwargs*)
Bases: object

default_data

event_ref

is_internal_link

is_link

is_orphaned

is_page

is_plugin_link

is_separator

is_user_link

is_visible

localized_title

locator

url

class indico.modules.events.layout.models.menu.**MenuEntryType**
Bases: indico.util.struct.enum.RichIntEnum

internal_link = 2

page = 5

plugin_link = 4

separator = 1

user_link = 3

class indico.modules.events.layout.models.menu.**TransientMenuEntry**(*event*, *is_enabled*, *name*, *position*, *children*)
Bases: *indico.modules.events.layout.models.menu.MenuEntryMixin*

id

Utilities

class indico.modules.events.layout.util.**MenuEntryData**(*title*, *name*, *endpoint=None*, *position=-1*, *is_enabled=True*, *visible=None*, *parent=None*, *static_site=False*)
Bases: object

Container to transmit menu entry-related data via signals

The data contained is transmitted via the *sidemenu* signal and used to build the side menu of an event.

Parameters

- **title** – str – The title of the menu, displayed to the user. The title should be translated using the normal gettext function, i.e. _('...'), or the plugin's bound gettext function.

- **name** – str – Name used to refer to the entry internally. This is never shown to the user. The name must be unique, names from plugins are automatically prefixed with the plugin name and a colon and therefore have to be unique only within the plugin. To mark the entry as active, its name must be specified in the *menu_entry_name* class attribute of the WP class. For plugins, the plugin name must be specified via the *menu_entry_plugin* attribute as well.
- **endpoint** – str – The endpoint the entry will point to.
- **position** – int – The desired position of the menu entry. the position is indicative only, relative to the other entries and not the exact position. Entries with the same position will be sorted alphanumerically on their name. A position of *-1* will append the entry at the end of the menu.
- **is_enabled** – bool – Whether the entry should be enabled by default (Default: *True*).
- **visible** – function – Determines if the entry should be visible. This is a simple function which takes only the *event* as parameter and returns a boolean to indicate if the entry is visible or not. It is called whenever the menu is displayed, so the current state of the event/user can be taken into account.
- **parent** – str – The name of the parent entry (None for root entries).
- **static_site** – bool or str – If True, this menu item should be shown in the menu of a static site. When set to a string, the string will be used instead of a mangled version of the endpoint's URL.

name

plugin = None

visible(*event*)

indico.modules.events.layout.util.**build_menu_entry_name**(*name*, *plugin=None*)

Builds the proper name for a menu entry.

Given a menu entry's name and optionally a plugin, returns the correct name of the menu entry.

Parameters

- **name** – str – The name of the menu entry.
- **plugin** – IndicoPlugin or str – The plugin (or the name of the plugin) which created the entry.

indico.modules.events.layout.util.**get_css_file_data**(*event*)

indico.modules.events.layout.util.**get_css_url**(*event*, *force_theme=None*, *for_preview=False*)

Builds the URL of a CSS resource.

Parameters

- **event** – The *Event* to get the CSS url for
- **force_theme** – The ID of the theme to override the custom CSS resource only if it exists
- **for_preview** – Whether the URL is used in the CSS preview page

Returns The URL to the CSS resource

indico.modules.events.layout.util.**get_logo_data**(*event*)

indico.modules.events.layout.util.**get_menu_entries_from_signal**(**args*, ***kwargs*)

indico.modules.events.layout.util.**get_menu_entry_by_name**(**args*, ***kwargs*)

indico.modules.events.layout.util.**get_plugin_conference_themes**()

indico.modules.events.layout.util.**is_menu_entry_enabled**(*entry_name*, *event*)
Check whether the MenuEntry is enabled

indico.modules.events.layout.util.**menu_entries_for_event**(**args*, ***kwargs*)

Log

Todo

Docstrings (module, models, utilities)

Models

class indico.modules.events.logs.models.entries.**EventLogEntry**(***kwargs*)
Bases: flask_sqlalchemy.Model

Log entries for events

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

data
Type-specific data

event
The Event this log entry is associated with

event_id
The ID of the event

id
The ID of the log entry

kind
The general kind of operation that was performed

logged_date

logged_dt
The date/time when the reminder was created

module
The module the operation was related to (does not need to match something in indico.modules and should be human-friendly but not translated).

realm
The general area of the event the entry comes from

render()
Renders the log entry to be displayed.

If the renderer is not available anymore, e.g. because of a disabled plugin, None is returned.

renderer

summary
A short one-line description of the logged action. Should not be translated!

type
The type of the log entry. This needs to match the name of a log renderer.

user
The user associated with the log entry

user_id
The ID of the user associated with the entry

class indico.modules.events.logs.models.entries.**EventLogKind**
Bases: int, indico.util.struct.enum.IndicoEnum

change = 3

negative = 4

other = 1

positive = 2

class indico.modules.events.logs.models.entries.**EventLogRealm**
Bases: indico.util.struct.enum.RichIntEnum

emails = 5

event = 1

management = 2

participants = 3

reviewing = 4

Utilities

indico.modules.events.logs.util.**get_log_renderers**()

indico.modules.events.logs.util.**make_diff_log**(*changes*, *fields*)
Create a value for log data containing change information.

Parameters

- **changes** – a dict mapping attributes to `(old, new)` tuples
- **fields** – a dict mapping attributes to field metadata. for simple cases this may be a string with the human-friendly title, for more advanced fields it should be a dict containing `title`, a `type` string and a `convert` callback which will be invoked with a tuple containing the old and new value

indico.modules.events.logs.util.**render_changes**(*a*, *b*, *type_*)
Render the comparison of *a* and *b* as HTML.

Parameters

- **a** – old value
- **b** – new value
- **type** – the type determining how the values should be compared

class indico.modules.events.logs.renderers.**EmailRenderer**
Bases: *indico.modules.events.logs.renderers.EventLogRendererBase*

name = u'email'

template_name = u'events/logs/entry_email.html'

class indico.modules.events.logs.renderers.**EventLogRendererBase**
Bases: object

Base class for event log renderers.

classmethod get_data(*entry*)
Returns the entry data in a format suitable for the template.

This method may be overridden if the entry's data needs to be preprocessed before being passed to the template.

It MUST NOT modify *entry.data* directly.

name = None
unique name of the log renderer (matches EventLogEntry.type)

plugin = None
plugin containing this renderer - assigned automatically

classmethod render_entry(*entry*)
Renders the log entry row

Parameters **entry** – A *EventLogEntry*

template_kwargs = {}
extra kwargs passed to *render_template*

template_name = None
template used to render the log entry

class indico.modules.events.logs.renderers.**SimpleRenderer**
Bases: *indico.modules.events.logs.renderers.EventLogRendererBase*

classmethod get_data(*entry*)

name = u'simple'

template_kwargs = {u'compare': <function render_changes>}

template_name = u'events/logs/entry_simple.html'

Event Management

class indico.modules.events.management.controllers.**RHManageEventBase**
Bases: indico.legacy.webinterface.rh.conferenceBase.RHConferenceBase, indico.legacy.webinterface.rh.base.RHModificationBaseProtected

Base class for event management RHs

class indico.modules.events.management.views.**WPEventManagement**(*rh*, *event_*, *active_menu_item=None*, ***kwargs*)
Bases: indico.legacy.webinterface.pages.base.WPJinjaMixin, indico.legacy.webinterface.pages.main.WPMainBase

Base class for event management pages.

When using this class the template will always have *event* available; it is not necessary to pass it as a kwarg when calling the *render_template* classmethod.

When using the class directly, pass the menu item as a posarg:

```
return WPEventManagement.render_template('foobar.html', self.event, 'foobar',
                                         foo='bar')
```

When subclassing you can set *sidemenu_option* on the class, allowing you to omit it. This is recommended if you have many pages using the same menu item or if you already need to subclass for some other reason (e.g. to set a *template_prefix* or include additional JS/CSS bundles):

```
return WPSomething.render_template('foobar.html', self.event,
                                   foo='bar')
```

Note

Todo

Docstrings (module, models, utilities)

Models

class indico.modules.events.notes.models.notes.**EventNote**(***kwargs*)
Bases: indico.core.db.sqlalchemy.links.LinkMixin, flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

allowed_link_types = frozenset([<LinkType.event: 2>, <LinkType.contribution: 3>, <LinkType.subcontribution: 4>

category = None

category_id = None

contribution

contribution_id

create_revision(*render_mode*, *source*, *user*)
Creates a new revision if needed and marks it as undeleted if it was

Any change to the render mode or the source causes a new revision to be created. The user is not taken into account since a user "modifying" a note without changing things is not really a change.

current_revision
The currently active revision of the note

current_revision_id
The ID of the current revision

delete(*user*)
Marks the note as deleted and adds a new empty revision

event

event_id

events_backref_name = u'all_notes'

classmethod get_for_linked_object(*linked_object*, *preload_event=True*)
Gets the note for the given object.

This only returns a note that hasn't been deleted.

Parameters

- **linked_object** – An event, session, contribution or subcontribution.
- **preload_event** – If all notes for the same event should be pre-loaded and cached in the app context.

classmethod get_or_create(*linked_object*)
Gets the note for the given object or creates a new one.

If there is an existing note for the object, it will be returned even. Otherwise a new note is created.

html
The rendered HTML of the note

id
The ID of the note

is_deleted
If the note has been deleted

link_backref_name = u'note'

link_type

linked_event

linked_event_id

locator
Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

revisions
The list of all revisions for the note

session

session_id

subcontribution

subcontribution_id

unique_links = True

class indico.modules.events.notes.models.notes.**EventNoteRevision**(***kwargs*)
Bases: flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

created_dt
The date/time when the revision was created

html
The rendered HTML of the note

id
The ID of the revision

note_id
The ID of the associated note

render_mode
How the note is rendered

source
The raw source of the note as provided by the user

user
The user who created the revision

user_id
The user who created the revision

Utilities

indico.modules.events.notes.util.**build_note_api_data**(*note*)

indico.modules.events.notes.util.**build_note_legacy_api_data**(*note*)

indico.modules.events.notes.util.**can_edit_note**(*obj*, *user*)
Checks if a user can edit the object's note

indico.modules.events.notes.util.**get_scheduled_notes**(*event*)
Gets all notes of scheduled items inside an event

Paper

Todo

Docstrings (module, models, operations, utilities, settings)

Models

class indico.modules.events.papers.models.call_for_papers.**CallForPapers**(*event*)
Bases: object

Proxy class to facilitate access to the call for papers settings

announcement

assignees

can_access_judging_area(*user*)

can_access_reviewing_area(*user*)

close()

content_review_questions

content_reviewer_deadline

content_reviewers

content_reviewing_enabled

end_dt

get_questions_for_review_type(*review_type*)

get_reviewing_state(*reviewing_type*)

has_ended

has_started

is_judge(*user*)

is_manager(*user*)

is_open

is_reviewer(*user*, *role=None*)

is_staff(*user*)

judge_deadline

judges

layout_review_questions

layout_reviewer_deadline

layout_reviewers

layout_reviewing_enabled

managers

open()

rating_range

schedule(*start_dt*, *end_dt*)

set_reviewing_state(*reviewing_type*, *enable*)

start_dt

user_competences

class indico.modules.events.papers.models.comments.**PaperReviewComment**(***kwargs*)

Bases: *indico.modules.events.models.reviews.ProposalCommentMixin*, indico.core.db.sqlalchemy.review_comments.ReviewCommentMixin, flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

can_edit(*user*)

can_view(*user*)

created_dt

id

is_deleted

locator

Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

modified_by

modified_by_id

modified_dt

paper_revision

render_mode = 2

revision_id

user

user_backref_name = u'review_comments'

user_id

user_modified_backref_name = u'modified_review_comments'

visibility

class indico.modules.events.papers.models.competences.**PaperCompetence**(***kwargs*)
Bases: flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

competences

event

event_id

id

user

user_id

class indico.modules.events.papers.models.files.**PaperFile**(**args*, ***kwargs*)
Bases: indico.core.storage.models.StoredFileMixin, flask_sqlalchemy.Model

add_file_date_column = False

content_type
The MIME type of the file

created_dt = None

filename
The name of the file

id

locator
Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named locator as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The other locator can then be accessed by passing obj.locator.other to the code expecting an object with a locator.

md5
An MD5 hash of the file.

Automatically assigned when *save()* is called.

paper

paper_revision

revision_id

size
The size of the file (in bytes).

Automatically assigned when *save()* is called.

storage_backend

storage_file_id

class indico.modules.events.papers.models.papers.**Paper**(*contribution*)
Bases: *indico.modules.events.models.reviews.ProposalMixin*

Proxy class to facilitate access to all paper-related properties

accepted_revision

call_for_proposals_attr = u'cfp'

can_comment(*user*, *check_state=False*)

can_judge(*user*, *check_state=False*)

can_manage(*user*)

can_review(*user*, *check_state=False*)

can_submit(*user*)

create_comment_endpoint = u'papers.submit_comment'

create_judgment_endpoint = u'papers.judge_paper'

create_review_endpoint = u'papers.submit_review'

delete_comment_endpoint = u'papers.delete_comment'

edit_comment_endpoint = u'papers.edit_comment'

edit_review_endpoint = u'papers.edit_review'

event

files

get_last_revision()

get_revisions()

is_in_final_state

judgment_comment

last_revision

locator
Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

proposal_type = u'paper'

proxied_attr = u'contribution'

reset_state()

revision_count

revisions

revisions_enabled = True

state

title

verbose_title

class indico.modules.events.papers.models.review_questions.**PaperReviewQuestion**(***kwargs*)
Bases: indico.core.db.sqlalchemy.review_questions.ReviewQuestionMixin, flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

event

event_backref_name = u'paper_review_questions'

event_id

id

is_deleted

no_score

position

text

type

class indico.modules.events.papers.models.review_ratings.**PaperReviewRating**(***kwargs*)
Bases: indico.core.db.sqlalchemy.review_ratings.ReviewRatingMixin, flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

id

question

question_class
alias of PaperReviewQuestion

question_id

review

review_class
alias of PaperReview

review_id

value

class indico.modules.events.papers.models.reviews.**PaperAction**
Bases: indico.util.struct.enum.RichIntEnum

accept = 1

reject = 2

to_be_corrected = 3

class indico.modules.events.papers.models.reviews.**PaperCommentVisibility**
Bases: indico.util.struct.enum.RichIntEnum

Most to least restrictive visibility for paper comments

contributors = 3

judges = 1

reviewers = 2

users = 4

class indico.modules.events.papers.models.reviews.**PaperJudgmentProxy**(*paper*)
Bases: object

Represents a timeline item for the non final judgments

created_dt

timeline_item_type = u'judgment'

class indico.modules.events.papers.models.reviews.**PaperReview**(***kwargs*)
Bases: *indico.modules.events.models.reviews.ProposalReviewMixin*, indico.core.db.sqlalchemy.descriptions.RenderModeMixin, flask_sqlalchemy.Model

Represents a paper review, emitted by a layout or content reviewer

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

TIMELINE_TYPE = u'review'

can_edit(*user*, *check_state=False*)

can_view(*user*)

comment

created_dt

default_render_mode = 2

group_attr = u'type'

group_proxy_cls
alias of *PaperTypeProxy*

id

locator
Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

modified_dt

possible_render_modes = set([<RenderMode.markdown: 2>])

proposed_action

render_mode = 2

revision

revision_attr = u'revision'

revision_id

score

type

user

user_id

visibility

class indico.modules.events.papers.models.reviews.**PaperReviewType**
Bases: indico.util.struct.enum.RichIntEnum

content = 2

layout = 1

class indico.modules.events.papers.models.reviews.**PaperTypeProxy**(*group*)
Bases: *indico.modules.events.models.reviews.ProposalGroupProxy*

locator
Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

class indico.modules.events.papers.models.revisions.**PaperRevision**(**args*, ***kwargs*)
Bases: *indico.modules.events.models.reviews.ProposalRevisionMixin*, indico.core.db.sqlalchemy.descriptions.RenderModeMixin, flask_sqlalchemy.Model

default_render_mode = 2

get_reviewed_for_groups(*user*, *include_reviewed=False*)

get_reviews(*group=None*, *user=None*)

get_spotlight_file()

get_timeline(*user=None*)

has_user_reviewed(*user*, *review_type=None*)

id

is_last_revision

judge

judge_id

judgment_comment

judgment_dt

locator
Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

number

paper

possible_render_modes = set([<RenderMode.markdown: 2>])

proposal_attr = u'paper'

render_mode = 2

state

submitted_dt

submitter

submitter_id

class indico.modules.events.papers.models.revisions.**PaperRevisionState**
Bases: indico.util.struct.enum.RichIntEnum

accepted = 2

rejected = 3

submitted = 1

to_be_corrected = 4

class indico.modules.events.papers.models.templates.**PaperTemplate**(***kwargs*)
Bases: indico.core.storage.models.StoredFileMixin, flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

add_file_date_column = False

content_type
The MIME type of the file

created_dt = None

description

event

event_id

filename
The name of the file

id

locator
Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

md5
An MD5 hash of the file.

Automatically assigned when *save()* is called.

name

size
The size of the file (in bytes).

Automatically assigned when *save()* is called.

storage_backend

storage_file_id

Operations

indico.modules.events.papers.operations.**close_cfp**(*event*)

indico.modules.events.papers.operations.**create_comment**(**args*, ***kwargs*)

indico.modules.events.papers.operations.**create_competences**(*event*, *user*, *competences*)

indico.modules.events.papers.operations.**create_paper_revision**(*paper*, *submitter*, *files*)

indico.modules.events.papers.operations.**create_paper_template**(*event*, *data*)

indico.modules.events.papers.operations.**create_review**(*paper*, *review_type*, *user*, *review_data*, *questions_data*)

indico.modules.events.papers.operations.**delete_comment**(*comment*)

indico.modules.events.papers.operations.**delete_paper_template**(*template*)

indico.modules.events.papers.operations.**judge_paper**(**args*, ***kwargs*)

indico.modules.events.papers.operations.**open_cfp**(*event*)

indico.modules.events.papers.operations.**reset_paper_state**(*paper*)

indico.modules.events.papers.operations.**schedule_cfp**(*event*, *start_dt*, *end_dt*)

indico.modules.events.papers.operations.**set_deadline**(*event*, *role*, *deadline*, *enforce=True*)

indico.modules.events.papers.operations.**set_reviewing_state**(*event*, *reviewing_type*, *enable*)

indico.modules.events.papers.operations.**update_comment**(*comment*, *text*, *visibility*)

indico.modules.events.papers.operations.**update_competences**(*user_competences*, *competences*)

indico.modules.events.papers.operations.**update_paper_template**(*template*, *data*)

indico.modules.events.papers.operations.**update_review**(*review*, *review_data*, *questions_data*)

indico.modules.events.papers.operations.**update_reviewing_roles**(*event*, *users*, *contributions*, *role*, *assign*)

indico.modules.events.papers.operations.**update_team_members**(*event*, *managers*, *judges*, *content_reviewers=None*, *layout_reviewers=None*)

Utilities

indico.modules.events.papers.util.**get_contributions_with_paper_submitted_by_user**(*event*, *user*)

indico.modules.events.papers.util.**get_contributions_with_user_paper_submission_rights**(*event*, *user*)

indico.modules.events.papers.util.**get_events_with_paper_roles**(*user*, *dt=None*)
Get the IDs and PR roles of events where the user has any kind of paper reviewing privileges.

Parameters

- **user** – A *User*
- **dt** – Only include events taking place on/after that date

Returns A dict mapping event IDs to a set of roles

indico.modules.events.papers.util.**get_user_contributions_to_review**(*event*, *user*)
Get the list of contributions where user has paper to review

indico.modules.events.papers.util.**get_user_reviewed_contributions**(*event*, *user*)
Get the list of contributions where user already reviewed paper

indico.modules.events.papers.util.**has_contributions_with_user_paper_submission_rights**(*event*, *user*)

Settings

class indico.modules.events.settings.**EventACLProxy**(*proxy*)
Bases: indico.core.settings.proxy.ACLProxyBase

Proxy class for event-specific ACL settings

add_principal(*event*, **args*, ***kwargs*)
Adds a principal to an ACL

Parameters

- **event** – Event (or its ID)
- **name** – Setting name
- **principal** – A *User* or a *GroupProxy*

contains_user(*event*, **args*, ***kwargs*)
Checks if a user is in an ACL.

To pass this check, the user can either be in the ACL itself or in a group in the ACL.

Parameters

- **event** – Event (or its ID)
- **name** – Setting name
- **user** – A *User*

get(*event*, **args*, ***kwargs*)
Retrieves an ACL setting

Parameters

- **event** – Event (or its ID)
- **name** – Setting name

merge_users(*target*, *source*)
Replaces all ACL user entries for *source* with *target*

remove_principal(*event*, **args*, ***kwargs*)
Removes a principal from an ACL

Parameters

- **event** – Event (or its ID)
- **name** – Setting name
- **principal** – A *User* or a *GroupProxy*

set(*event*, **args*, ***kwargs*)
Replaces an ACL with a new one

Parameters

- **event** – Event (or its ID)
- **name** – Setting name
- **acl** – A set containing principals (users/groups)

class indico.modules.events.settings.**EventSettingProperty**(*proxy*, *name*, *default=<object object>*, *attr=None*)
Bases: indico.core.settings.proxy.SettingProperty

attr = u'event'

class indico.modules.events.settings.**EventSettingsProxy**(*module*, *defaults=None*, *strict=True*, *acls=None*, *converters=None*)

Bases: indico.core.settings.proxy.SettingsProxyBase

Proxy class to access event-specific settings for a certain module

acl_proxy_class
alias of *EventACLProxy*

delete(*event*, **args*, ***kwargs*)
Deletes settings.

Parameters

- **event** – Event (or its ID)
- **names** – One or more names of settings to delete

delete_all(*event*, **args*, ***kwargs*)
Deletes all settings.

Parameters **event** – Event (or its ID)

get(*event*, **args*, ***kwargs*)
Retrieves the value of a single setting.

Parameters

- **event** – Event (or its ID)
- **name** – Setting name
- **default** – Default value in case the setting does not exist

Returns The settings's value or the default value

get_all(*event*, **args*, ***kwargs*)
Retrieves all settings

Parameters

- **event** – Event (or its ID)
- **no_defaults** – Only return existing settings and ignore defaults.

Returns Dict containing the settings

query
Returns a query object filtering by the proxy's module.

set(*event*, **args*, ***kwargs*)
Sets a single setting.

Parameters

- **event** – Event (or its ID)
- **name** – Setting name
- **value** – Setting value; must be JSON-serializable

set_multi(*event*, **args*, ***kwargs*)
Sets multiple settings at once.

Parameters

- **event** – Event (or its ID)

- **items** – Dict containing the new settings

class indico.modules.events.settings.**ThemeSettingsProxy**
Bases: object

defaults

get_themes_for(**args*, ***kwargs*)

settings

themes

indico.modules.events.settings.**event_or_id**(*f*)

Payment

Todo

Docstrings (module, models, plugins)

Models

exception indico.modules.events.payment.models.transactions.**DoublePaymentTransaction**
Bases: exceptions.Exception

exception indico.modules.events.payment.models.transactions.**IgnoredTransactionAction**
Bases: exceptions.Exception

exception indico.modules.events.payment.models.transactions.**InvalidManualTransactionAction**
Bases: exceptions.Exception

exception indico.modules.events.payment.models.transactions.**InvalidTransactionAction**
Bases: exceptions.Exception

exception indico.modules.events.payment.models.transactions.**InvalidTransactionStatus**
Bases: exceptions.Exception

class indico.modules.events.payment.models.transactions.**PaymentTransaction**(***kwargs*)
Bases: flask_sqlalchemy.Model

Payment transactions

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

amount
the base amount the user needs to pay (without payment-specific fees)

classmethod create_next(*registration*, *amount*, *currency*, *action*, *provider=None*, *data=None*)

currency
the currency of the payment (ISO string, e.g. EUR or USD)

data
plugin-specific data of the payment

id
Entry ID

is_manual

plugin

provider
the provider of the payment (e.g. manual, PayPal etc.)

registration
The associated registration

registration_id
ID of the associated registration

render_details()
Renders the transaction details

status
a *TransactionStatus*

timestamp
the date and time the transaction was recorded

class indico.modules.events.payment.models.transactions.**TransactionAction**
Bases: int, indico.util.struct.enum.IndicoEnum

cancel = 2

complete = 1

pending = 3

reject = 4

class indico.modules.events.payment.models.transactions.**TransactionStatus**
Bases: int, indico.util.struct.enum.IndicoEnum

cancelled = 2
payment cancelled manually

failed = 3
payment attempt failed

pending = 4
payment on hold pending approval of merchant

rejected = 5
payment rejected after being pending

successful = 1
payment attempt succeeded

class indico.modules.events.payment.models.transactions.**TransactionStatusTransition**
Bases: object

initial_statuses = [<TransactionStatus.cancelled: 2>, <TransactionStatus.failed: 3>, <TransactionStatus.rejected: 5

classmethod next(*transaction*, *action*, *provider=None*)

Utilities

indico.modules.events.payment.util.**get_active_payment_plugins**(*event*)
Returns a dict containing the active payment plugins of an event.

indico.modules.events.payment.util.**get_payment_plugins**()
Returns a dict containing the available payment plugins.

indico.modules.events.payment.util.**register_transaction**(*registration*, *amount*, *currency*, *action*, *provider=None*, *data=None*)
Creates a new transaction for a certain transaction action.

Parameters

- **registration** – the *Registration* associated to the transaction
- **amount** – the (strictly positive) amount of the transaction
- **currency** – the currency used for the transaction
- **action** – the *TransactionAction* of the transaction
- **provider** – the payment method name of the transaction, or '_manual' if no payment method has been used
- **data** – arbitrary JSON-serializable data specific to the transaction's provider

Plugins

class indico.modules.events.payment.plugins.**PaymentPluginMixin**
Bases: object

adjust_payment_form_data(*data*)
Updates the payment form data if necessary.

This method can be overridden to update e.g. the amount based on choices the user makes in the payment form or to provide additional data to the form. To do so, *data* must be modified.

Parameters **data** – a dict containing event, registration, amount, currency, settings and event_settings

can_be_modified(*user*, *event*)
Checks if the user is allowed to enable/disable/modify the payment method.

Parameters

- **user** – the *User* repesenting the user
- **event** – the Event

category = u'Payment'

default_settings

event_settings_form
alias of PaymentEventSettingsFormBase

get_event_management_url(*event*, ***kwargs*)

get_invalid_regforms(*event*)
Return registration forms with incompatible currencies

get_method_name(*event*)
Returns the (customized) name of the payment method.

init()

logo_url

render_payment_form(*registration*)
Returns the payment form shown to the user.

Parameters **registration** – a `Registration` object

render_transaction_details(*transaction*)
Renders the transaction details in event management

Override this (or inherit from the template) to show more useful data such as transaction IDs

Parameters **transaction** – the `PaymentTransaction`

settings_form
alias of `PaymentPluginSettingsFormBase`

supports_currency(*currency*)

valid_currencies = None
Set containing all valid currencies. Set to *None* to allow all.

Person

Todo

Docstrings (module, operations)

Operations

`indico.modules.events.persons.operations.`**update_person**(*person*, *data*)

Placeholders

class `indico.modules.events.persons.placeholders.`**EmailPlaceholder**
Bases: `indico.util.placeholders.Placeholder`

description = lu'Email of the person'

name = u'email'

classmethod render(*person*, *event*, ***kwargs*)

class `indico.modules.events.persons.placeholders.`**EventLinkPlaceholder**
Bases: `indico.util.placeholders.Placeholder`

description = lu'Link to the event'

name = u'event_link'

classmethod render(*person*, *event*, ***kwargs*)

class `indico.modules.events.persons.placeholders.`**EventTitlePlaceholder**
Bases: `indico.util.placeholders.Placeholder`

description = lu'The title of the event'

name = u'event_title'

classmethod render(*person*, *event*, ***kwargs*)

class indico.modules.events.persons.placeholders.**FirstNamePlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'First name of the person'

name = u'first_name'

classmethod render(*person*, *event*, ***kwargs*)

class indico.modules.events.persons.placeholders.**LastNamePlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'Last name of the person'

name = u'last_name'

classmethod render(*person*, *event*, ***kwargs*)

class indico.modules.events.persons.placeholders.**RegisterLinkPlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'The link for the registration page'

name = u'register_link'

classmethod render(*person*, *event*, ***kwargs*)

Registration

Todo

Docstrings (module, models, utilities, statistics)

Models

class indico.modules.events.registration.models.registrations.**Registration**(***kwargs*)
Bases: flask_sqlalchemy.Model

Somebody's registration for an event through a registration form

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

base_price
The base registration fee (that is not specific to form items)

billable_data

can_be_modified

checked_in
Whether the person has checked in. Setting this also sets or clears *checked_in_dt*.

checked_in_dt
The date/time when the person has checked in

currency
Registration price currency

data
The registration this data is associated with

data_by_field

display_full_name
Return the full name using the user's preferred name format.

email
The email of the registrant

event
The Event containing this registration

event_id
The ID of the event

first_name
The first name of the registrant

friendly_id
The human-friendly ID for the object

full_name
Returns the user's name in 'Firstname Lastname' notation.

classmethod get_all_for_event(*event*)
Retrieve all registrations in all registration forms of an event.

get_full_name(*last_name_first=True*, *last_name_upper=False*, *abbrev_first_name=False*)
Returns the user's in the specified notation.

If not format options are specified, the name is returned in the 'Lastname, Firstname' notation.

Note: Do not use positional arguments when calling this method. Always use keyword arguments!

Parameters

- **last_name_first** – if "lastname, firstname" instead of "firstname lastname" should be used
- **last_name_upper** – if the last name should be all-uppercase
- **abbrev_first_name** – if the first name should be abbreviated to use only the first character

get_personal_data()

has_files

id
The ID of the object

is_active

is_cancelled

is_deleted
If the registration has been deleted

is_paid
Returns whether the registration has been paid for.

last_name
The last name of the registrant

locator
Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

order_by_name = (<sqlalchemy.sql.functions.Function at 0x7faac8464350; lower>, <sqlalchemy.sql.functions.Function

payment_dt
The date/time when the registration has been paid for

price
The total price of the registration.

This includes the base price, the field-specific price, and the custom price adjustment for the registrant.

Return type Decimal

price_adjustment
The price modifier applied to the final calculated price

registration_form_id
The ID of the registration form

render_base_price()

render_price()

render_price_adjustment()

sections_with_answered_fields

state
The state a registration is in

submitted_dt
The date/time when the registration was recorded

summary_data
Export registration data nested in sections and fields

sync_state(*_skip_moderation=True*)
Sync the state of the registration

ticket_uuid
The unique token used in tickets

transaction
The latest payment transaction associated with this registration

transaction_id
The ID of the latest payment transaction associated with this registration

update_state(*approved=None*, *paid=None*, *rejected=None*, *_skip_moderation=False*)
Update the state of the registration for a given action

The accepted kwargs are the possible actions. `True` means that the action occured and `False` that it was reverted.

user

user_id
The ID of the user who registered

uuid
The unguessable ID for the object

class indico.modules.events.registration.models.registrations.**RegistrationData**(***kwargs*)
Bases: indico.core.storage.models.StoredFileMixin, flask_sqlalchemy.Model

Data entry within a registration for a field in a registration form

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

add_file_date_column = False

content_type
The MIME type of the file

created_dt = None

data
The submitted data for the field

field_data
The associated field data object

field_data_id
The ID of the field data

file

file_required = False

filename
The name of the file

friendly_data

get_friendly_data(***kwargs*)

locator
Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

md5
An MD5 hash of the file.

Automatically assigned when *save()* is called.

price

registration_id
The ID of the registration

render_price()

size
The size of the file (in bytes).

Automatically assigned when *save()* is called.

storage_backend

storage_file_id

summary_data

user_data

class indico.modules.events.registration.models.registrations.**RegistrationState**
Bases: indico.util.struct.enum.RichIntEnum

complete = 1

pending = 2

rejected = 3

unpaid = 5

withdrawn = 4

class indico.modules.events.registration.models.form_fields.**RegistrationFormField**(***kwargs*)
Bases: *indico.modules.events.registration.models.items.RegistrationFormItem*

A registration form field

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

calculate_price(*registration_data*)

children

current_data

current_data_id

data

data_versions

description

field_impl
Gets the implementation of the field.

Returns An instance of a *RegistrationFormFieldBase* subclass

get_friendly_data(*registration_data*, ***kwargs*)

html_field_name

id

input_type

is_deleted

is_enabled

is_manager_only

is_required

locator

parent_id

personal_data_type

position

registration_form_id

title

type

versioned_data

view_data

class indico.modules.events.registration.models.form_fields.**RegistrationFormFieldData**(***kwargs*
Bases: flask_sqlalchemy.Model

Description of a registration form field

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

field_id
The ID of the registration form field

id
The ID of the object

versioned_data
Data describing the field

class indico.modules.events.registration.models.form_fields.**RegistrationFormPersonalDataField**
Bases: *indico.modules.events.registration.models.form_fields.RegistrationFormField*

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

children

current_data

current_data_id

data

data_versions

description

html_field_name

id

input_type

is_deleted

is_enabled

is_manager_only

is_required

parent_id

personal_data_type

position

registration_form_id

title

type

view_data

class indico.modules.events.registration.models.forms.**ModificationMode**
Bases: indico.util.struct.enum.RichIntEnum

allowed_always = 1

allowed_until_payment = 2

not_allowed = 3

class indico.modules.events.registration.models.forms.**RegistrationForm**(***kwargs*)
Bases: flask_sqlalchemy.Model

A registration form for an event

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

active_fields

active_registrations

base_price
The base fee users have to pay when registering

can_submit(*user*)

contact_info
Contact information for registrants

currency
Currency for prices in the registration form

disabled_sections

end_dt
Datetime when the registration form is closed

event
The Event containing this registration form

event_id
The ID of the event

form_items

get_personal_data_field_id(*personal_data_type*)
Returns the field id corresponding to the personal data field with the given name.

get_registration(**args*, ***kwargs*)
Retrieves registrations for this registration form by user or uuid

has_ended

has_started

id
The ID of the object

introduction

invitations
The registration invitations associated with this form

is_active

is_deleted
Whether the registration has been marked as deleted

is_modification_allowed(*registration*)
Checks whether a registration may be modified

is_modification_open

is_open

is_participation
Whether it's the 'Participants' form of a meeting/lecture

is_scheduled

limit_reached

locator

manager_notification_recipients
List of emails that should receive management notifications

manager_notifications_enabled
Whether the manager notifications for this event are enabled

message_complete
Custom message to include in emails for complete registrations

message_pending
Custom message to include in emails for pending registrations

message_unpaid
Custom message to include in emails for unpaid registrations

moderation_enabled
Whether registrations must be approved by a manager

modification_end_dt
Datetime when the modification period is over

modification_mode
Whether registration modifications are allowed

notification_sender_address
Notifications sender address

publish_checkin_enabled
Whether checked-in status should be displayed in the event pages and participant list

publish_registration_count
Whether to display the number of registrations

publish_registrations_enabled
Whether registrations should be displayed in the participant list

registration_limit
Maximum number of registrations allowed

registrations
The registrations associated with this form

render_base_price()

require_login
Whether users must be logged in to register

require_user
Whether registrations must be associated with an Indico account

sections

sender_address

start_dt
Datetime when the registration form is open

ticket_on_email
Whether to send tickets by e-mail

ticket_on_event_page
Whether to show a ticket download link on the event homepage

ticket_on_summary_page
Whether to show a ticket download link on the registration summary page

ticket_template
The template used to generate tickets

ticket_template_id
The ID of the template used to generate tickets

tickets_enabled
Whether tickets are enabled for this form

title
The title of the registration form

class indico.modules.events.registration.models.invitations.**InvitationState**
Bases: indico.util.struct.enum.RichIntEnum

accepted = 1

declined = 2

pending = 0

class indico.modules.events.registration.models.invitations.**RegistrationInvitation**(***kwargs*)
Bases: flask_sqlalchemy.Model

An invitation for someone to register

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

affiliation
The affiliation of the invited person

email
The email of the invited person

first_name
The first name of the invited person

id
The ID of the invitation

last_name
The last name of the invited person

locator
Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

registration
The associated registration

registration_form_id
The ID of the registration form

registration_id
The ID of the registration (if accepted)

skip_moderation
Whether registration moderation should be skipped

state
The state of the invitation

uuid
The UUID of the invitation

class indico.modules.events.registration.models.items.**PersonalDataType**
Bases: int, indico.util.struct.enum.IndicoEnum

Description of the personal data items that exist on every registration form

FIELD_DATA = [(<PersonalDataType.title: 5>, {u'input_type': u'single_choice', u'data': {u'item_type': u'dropdown', u'

address = 6

affiliation = 4

column
The Registration column in which the value is stored in addition to the regular registration data entry.

country = 8

email = 1

first_name = 2

get_title()

is_required

last_name = 3

phone = 7

position = 9

title = 5

class indico.modules.events.registration.models.items.**RegistrationFormItem**(***kwargs*)
Bases: flask_sqlalchemy.Model

Generic registration form item

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

children

current_data
The latest value of the field

current_data_id
The ID of the latest data

data
unversioned field data

data_versions
The list of all versions of the field data

description
Description of this field

id
The ID of the object

input_type
input type of this field

is_deleted
Whether field has been "deleted"

is_enabled
Whether the field is enabled

is_field

is_manager_only
if the section is only accessible to managers

is_required
determines if the field is mandatory

is_section

is_visible

parent_id
The ID of the parent form item

personal_data_type
The type of a personal data field

position

registration_form_id
The ID of the registration form

title
The title of this field

type
The type of the registration form item

view_data
Returns object with data that Angular can understand

class indico.modules.events.registration.models.items.**RegistrationFormItemType**
Bases: int, indico.util.struct.enum.IndicoEnum

field = 2

field_pd = 5

section = 1

section_pd = 4

text = 3

class indico.modules.events.registration.models.items.**RegistrationFormPersonalDataSection**(**kw
Bases: *indico.modules.events.registration.models.items.RegistrationFormSection*

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

children

current_data

current_data_id

data

data_versions

description

id

input_type

is_deleted

is_enabled

is_manager_only

is_required

parent_id

personal_data_type

position

registration_form_id

title

type

view_data

class indico.modules.events.registration.models.items.**RegistrationFormSection**(***kwargs*)
Bases: *indico.modules.events.registration.models.items.RegistrationFormItem*

Registration form section that can contain fields and text

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

active_fields

children

current_data

current_data_id

data

data_versions

description

fields

id

input_type

is_deleted

is_enabled

is_manager_only

is_required

locator

own_data

parent_id

personal_data_type

position

registration_form_id

title

type

view_data

class indico.modules.events.registration.models.items.**RegistrationFormText**(***kwargs*)
Bases: *indico.modules.events.registration.models.items.RegistrationFormItem*

Text to be displayed in registration form sections

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

children

current_data

current_data_id

data

data_versions

description

id

input_type

is_deleted

is_enabled

is_manager_only

is_required

locator

parent_id

personal_data_type

position

registration_form_id

title

type

view_data

Utilities

indico.modules.events.registration.util.**build_registration_api_data**(*registration*)

indico.modules.events.registration.util.**build_registrations_api_data**(*event*)

indico.modules.events.registration.util.**check_registration_email**(*regform*, *email*, *registration=None*, *management=False*)

Checks whether an email address is suitable for registration.

Parameters

- **regform** – The registration form
- **email** – The email address
- **registration** – The existing registration (in case of modification)
- **management** – If it's a manager adding a new registration

indico.modules.events.registration.util.**create_personal_data_fields**(*regform*)

Creates the special section/fields for personal data.

indico.modules.events.registration.util.**create_registration**(*regform*, *data*, *invitation=None*, *management=False*, *notify_user=True*)

indico.modules.events.registration.util.**generate_spreadsheet_from_registrations**(*registrations*, *regform_items*, *static_items*)

Generates a spreadsheet data from a given registration list.

Parameters

- **registrations** – The list of registrations to include in the file
- **regform_items** – The registration form items to be used as columns
- **static_items** – Registration form information as extra columns

indico.modules.events.registration.util.**generate_ticket_qr_code**(*registration*)

Generate a Pillow *Image* with a QR Code encoding a check-in ticket.

Parameters **registration** – corresponding *Registration* object

indico.modules.events.registration.util.**get_event_regforms**(*event*, *user*, *with_registrations=False*)

Get registration forms with information about user registrations.

Parameters

- **event** – the *Event* to get registration forms for
- **user** – A *User*
- **with_registrations** – Whether to return the user's registration instead of just whether they have one

indico.modules.events.registration.util.**get_event_section_data**(*regform*, *management=False*, *registration=None*)

indico.modules.events.registration.util.**get_events_registered**(*user*, *dt=None*)

Gets the IDs of events where the user is registered.

Parameters

- **user** – A *User*
- **dt** – Only include events taking place on/after that date

Returns A set of event ids

indico.modules.events.registration.util.**get_published_registrations**(*event*)

Get a list of published registrations for an event.

Parameters **event** – the *Event* to get registrations for

Returns list of *Registration* objects

indico.modules.events.registration.util.**get_registrations_with_tickets**(*user*, *event*)

indico.modules.events.registration.util.**get_title_uuid**(*regform*, *title*)

Convert a string title to its UUID value

If the title does not exist in the title PD field, it will be ignored and returned as `None`.

indico.modules.events.registration.util.**make_registration_form**(*regform*, *management=False*, *registration=None*)
Creates a WTForm based on registration form fields

indico.modules.events.registration.util.**modify_registration**(*registration*, *data*, *management=False*, *notify_user=True*)

indico.modules.events.registration.util.**url_rule_to_angular**(*endpoint*)
Converts a flask-style rule to angular style

Placeholders

class indico.modules.events.registration.placeholders.registrations.**EventLinkPlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'Link to the event'

name = u'event_link'

classmethod render(*regform*, *registration*)

class indico.modules.events.registration.placeholders.registrations.**EventTitlePlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'The title of the event'

name = u'event_title'

classmethod render(*regform*, *registration*)

class indico.modules.events.registration.placeholders.registrations.**FieldPlaceholder**
Bases: indico.util.placeholders.ParametrizedPlaceholder

advanced = True

description = None

classmethod iter_param_info(*regform*, *registration*)

name = u'field'

param_required = True

param_restricted = True

classmethod render(*param*, *regform*, *registration*)

class indico.modules.events.registration.placeholders.registrations.**FirstNamePlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'First name of the person'

name = u'first_name'

classmethod render(*regform*, *registration*)

class indico.modules.events.registration.placeholders.registrations.**IDPlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'The ID of the registration'

name = u'id'

classmethod render(*regform*, *registration*)

class indico.modules.events.registration.placeholders.registrations.**LastNamePlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'Last name of the person'

name = u'last_name'

classmethod render(*regform*, *registration*)

class indico.modules.events.registration.placeholders.registrations.**LinkPlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'The link to the registration details'

name = u'link'

classmethod render(*regform*, *registration*)

class indico.modules.events.registration.placeholders.invitations.**FirstNamePlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'First name of the person'

name = u'first_name'

classmethod render(*invitation*)

class indico.modules.events.registration.placeholders.invitations.**InvitationLinkPlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'Link to accept/decline the invitation'

name = u'invitation_link'

classmethod render(*invitation*)

required = True

class indico.modules.events.registration.placeholders.invitations.**LastNamePlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'Last name of the person'

name = u'last_name'

classmethod render(*invitation*)

Settings

class indico.modules.events.registration.settings.**RegistrationSettingsProxy**(*module*, *defaults=None*, *strict=True*, *acls=None*, *converters=None*)
Bases: *indico.modules.events.settings.EventSettingsProxy*

Store per-event registration settings

get_participant_list_columns(*event*, *form=None*)

get_participant_list_form_ids(*event*)

set_participant_list_columns(*event*, *columns*, *form=None*)

set_participant_list_form_ids(*event*, *form_ids*)

Statistics

class indico.modules.events.registration.stats.**AccommodationStats**(*field*)
Bases: *indico.modules.events.registration.stats.FieldStats*, *indico.modules.events.registration.stats.StatsBase*

class indico.modules.events.registration.stats.**Cell**
Bases: *indico.modules.events.registration.stats.Cell*

Hold data and type for a cell of a stats table

The table below indicates the valid types and expected data.

type	data
str	*str* – string value
progress	*(int, str)* – a tuple with the progress (a value between 0 and 1) and a label
progress-stacked	*([int], str)* – a tuple with a list of progresses (values which must sum up to 1) and a label
currency	*float* – numeric value
icon	*str* – icon name from *_icons.scss*
default	*None* – renders a default cell with an *—* (use *Cell(type='str')* for an empty cell)

Parameters

- **type** – str – The type of data in the cell
- **data** – The data for the cell
- **colspan** – int – HTML colspan value for the cell
- **classes** – [str] – HTML classes to apply to the cell
- **qtip** – str – content for qtip

class indico.modules.events.registration.stats.**DataItem**
Bases: *indico.modules.events.registration.stats.DataItem*

Holds the aggregation of some data, intended for stats tables as a aggregation from which to generate cells.

Parameters

- **regs** – int – number of registrant
- **attendance** – int – number of people attending
- **capacity** – int – maximum number of people allowed to attend (*0* if unlimited)
- **billable** – bool – whether the item is billable to the or not
- **cancelled** – bool – whether the item is cancelled or not
- **price** – str – the price of the item
- **fixed_price** – bool – *True* if the price is per registrant, *False* if accompanying guests must pay as well.
- **paid** – int – number of registrants who paid
- **paid_amount** – float – amount already paid by registrants

- **unpaid** – int – number of registrants who haven't paid
- **unpaid_amount** – float – amount not already paid by registrants

class indico.modules.events.registration.stats.**FieldStats**(*field*, ***kwargs*)
Bases: object

Holds stats for a registration form field

get_table()
Returns a table containing the stats for each item.

Returns dict – A table with a list of head cells (key: *'head'*) and a list of rows (key: *'rows'*) where each row is a list of cells.

is_currency_shown

class indico.modules.events.registration.stats.**OverviewStats**(*regform*)
Bases: *indico.modules.events.registration.stats.StatsBase*

Generic stats for a registration form

class indico.modules.events.registration.stats.**StatsBase**(*title*, *subtitle*, *type*, ***kwargs*)
Bases: object

Base class for registration form statistics

Parameters

- **title** – str – the title for the stats box
- **subtitle** – str – the subtitle for the stats box
- **type** – str – the type used in Jinja to display the stats

is_currency_shown

Reminder

Todo

Docstrings (module)

Models

class indico.modules.events.reminders.models.reminders.**EventReminder**(***kwargs*)
Bases: flask_sqlalchemy.Model

Email reminders for events

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

all_recipients
Returns all recipients of the notifications.

This includes both explicit recipients and, if enabled, participants of the event.

created_dt
The date/time when the reminder was created

creator
The user who created the reminder

creator_id
The ID of the user who created the reminder

event
The Event this reminder is associated with

event_id
The ID of the event

event_start_delta
How long before the event start the reminder should be sent This is needed to update the *scheduled_dt* when changing the start time of the event.

id
The ID of the reminder

include_summary
If the notification should include a summary of the event's schedule.

is_overdue

is_relative
Returns if the reminder is relative to the event time

is_sent
If the reminder has been sent

locator

message
Custom message to include in the email

recipients
The recipients of the notification

reply_to_address
The address to use as Reply-To in the notification email.

scheduled_dt
The date/time when the reminder should be sent

send()
Sends the reminder to its recipients.

send_to_participants
If the notification should also be sent to all event participants

Utilities

indico.modules.events.reminders.util.**make_reminder_email**(*event*, *with_agenda*, *note*)
Returns the template module for the reminder email.

Parameters

- **event** – The event

- **with_agenda** – If the event's agenda should be included
- **note** – A custom message to include in the email

Request

Todo

Docstrings (module)

Models

class indico.modules.events.requests.models.requests.**Request**(***kwargs*)
Bases: flask_sqlalchemy.Model

Event-related requests, e.g. for a webcast

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

can_be_modified
Determines if the request can be modified or if a new one must be sent

comment
an optional comment for an accepted/rejected request

created_by_id
ID of the user creating the request

created_by_user
The user who created the request

created_dt
the date/time the request was created

data
plugin-specific data of the payment

definition

event
The Event this agreement is associated with

event_id
ID of the event

classmethod find_latest_for_event(*event*, *type_=None*)
Returns the latest requests for a given event.

Parameters

- **event** – the event to find the requests for
- **type** – the request type to retrieve, or *None* to get all

Returns a dict mapping request types to a *Request* or if *type_* was specified, a single *Request* or *None*

id
request ID

locator

processed_by_id
ID of the user processing the request

processed_by_user
The user who processed the request

processed_dt
the date/time the request was accepted/rejected

state
the requests's date, a *RequestState* value

type
the request type name

class indico.modules.events.requests.models.requests.**RequestState**
Bases: indico.util.struct.enum.RichIntEnum

accepted = 1

pending = 0

rejected = 2

withdrawn = 3

Utilities

indico.modules.events.requests.util.**get_request_definitions**()
Returns a dict of request definitions

indico.modules.events.requests.util.**is_request_manager**(*user*)
Checks if the user manages any request types

class indico.modules.events.requests.base.**RequestDefinitionBase**
Bases: object

Defines a service request which can be sent by event managers.

classmethod accept(*req*, *data*, *user*)
Accepts the request

To ensure that additional data is saved, this method should call **:method:'manager_save'**.

Parameters

- **req** – the Request of the request
- **data** – the form data from the management form
- **user** – the user processing the request

classmethod can_be_managed(*user*)
Checks whether the user is allowed to manage this request type

Parameters **user** – a *User*

classmethod create_form(*event*, *existing_request=None*)
Creates the request form

Parameters

- **event** – the event the request is for
- **existing_request** – the `Request` if there's an existing request of this type

Returns an instance of an `IndicoForm` subclass

classmethod create_manager_form(*req*)
Creates the request management form

Parameters **req** – the `Request` of the request

Returns an instance of an `IndicoForm` subclass

form = None
the `IndicoForm` to use for the request form

form_defaults = {}
default values to use if there's no existing request

classmethod get_manager_notification_emails()
Returns the email addresses of users who manage requests of this type

The email addresses are used only for notifications. It usually makes sense to return the email addresses of the users who pass the **:method:'can_be_managed'** check.

Returns set of email addresses

classmethod get_notification_template(*name*, ***context*)
Gets the template module for a notification email

Parameters

- **name** – the template name
- **context** – data passed to the template

manager_form
the `IndicoForm` to use for the request manager form

alias of `RequestManagerForm`

classmethod manager_save(*req*, *data*)
Saves management-specific data

This method is called when the management form is submitted without accepting/rejecting the request (which is guaranteed to be already accepted or rejected).

Parameters

- **req** – the `Request` of the request
- **data** – the form data from the management form

name = None
the unique internal name of the request type

plugin = None
the plugin containing this request definition - assigned automatically

classmethod reject(*req*, *data*, *user*)
Rejects the request

To ensure that additional data is saved, this method should call **:method:‘manager_save‘**.

Parameters

- **req** – the `Request` of the request
- **data** – the form data from the management form
- **user** – the user processing the request

classmethod render_form(*event*, ***kwargs*)
Renders the request form

Parameters

- **event** – the event the request is for
- **kwargs** – arguments passed to the template

classmethod send(*req*, *data*)
Sends a new/modified request

Parameters

- **req** – the `Request` of the request
- **data** – the form data from the request form

title = None
the title of the request type as shown to users

classmethod withdraw(*req*, *notify_event_managers=True*)
Withdraws the request

Parameters

- **req** – the `Request` of the request
- **notify_event_managers** – if event managers should be notified

Session

Todo

Docstrings (module, models, operations, utilities)

Models

class `indico.modules.events.sessions.models.sessions.`**Session**(***kwargs*)
Bases: `indico.core.db.sqlalchemy.descriptions.DescriptionMixin`, `indico.core.db.sqlalchemy.colors.ColorMixin`, `indico.core.db.sqlalchemy.protection.ProtectionManagersMixin`, `indico.core.db.sqlalchemy.locations.LocationMixin`, `indico.core.db.sqlalchemy.attachments.AttachedItemsMixin`, `indico.core.db.sqlalchemy.notes.AttachedNotesMixin`, `flask_sqlalchemy.Model`

ATTACHMENT_FOLDER_ID_COLUMN = u’session_id’

PRELOAD_EVENT_ATTACHED_ITEMS = True

PRELOAD_EVENT_NOTES = True

access_key = None

acl_entries

allow_relationship_preloading = True

background_color

blocks

can_manage_blocks(*user*, *allow_admin=True*)
Check whether a user can manage session blocks.

This only applies to the blocks themselves, not to contributions inside them.

can_manage_contributions(*user*, *allow_admin=True*)
Check whether a user can manage contributions within the session.

code

conveners

default_colors = ColorTuple(text=u'202020', background=u'e3f2d3')

default_contribution_duration

default_render_mode = 2

disallowed_protection_modes = frozenset([])

end_dt

event

event_id

friendly_id
The human-friendly ID for the session

get_non_inheriting_objects()
Get a set of child objects that do not inherit protection

id

inherit_location

inheriting_have_acl = True

is_deleted

is_poster

location_backref_name = u'sessions'

location_parent

locator
Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

own_address

own_no_access_contact = None

own_room

own_room_id

own_room_name

own_venue

own_venue_id

own_venue_name

possible_render_modes = set([<RenderMode.markdown: 2>])

classmethod preload_acl_entries(*event*)

protection_mode

protection_parent

render_mode = 2

session
Convenience property so all event entities have it

start_dt

text_color

title

class indico.modules.events.sessions.models.blocks.**SessionBlock**(***kwargs*)
Bases: indico.core.db.sqlalchemy.locations.LocationMixin, flask_sqlalchemy.Model

can_access(*user*, *allow_admin=True*)

can_edit_note(*user*)

can_manage(*user*, *allow_admin=True*)

can_manage_attachments(*user*)

contribution_count

duration

end_dt

event

full_title

has_note

id

inherit_location

location_backref_name = u'session_blocks'

location_parent

locator

Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

note

own_address

own_room

own_room_id

own_room_name

own_venue

own_venue_id

own_venue_name

person_links

Persons associated with this session block

session_id

start_dt

title

class indico.modules.events.sessions.models.persons.**SessionBlockPersonLink**(**args*, ***kwargs*)

Bases: *indico.modules.events.models.persons.PersonLinkBase*

Association between EventPerson and SessionBlock.

Also known as a 'session convener'

display_order

id

object_relationship_name = u'session_block'

person

person_id

person_link_backref_name = u'session_block_links'

person_link_unique_columns = (u'session_block_id',)

session_block_id

class indico.modules.events.sessions.models.principals.**SessionPrincipal**(***kwargs*)

Bases: indico.core.db.sqlalchemy.principals.PrincipalRolesMixin, flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

allow_emails = True

disallowed_protection_modes = frozenset([])

email

full_access

id
The ID of the acl entry

ip_network_group = None

ip_network_group_id = None

local_group

local_group_id

multipass_group_name

multipass_group_provider

principal_backref_name = u'in_session_acls'

principal_for = u'Session'

read_access

roles

session_id
The ID of the associated session

type

unique_columns = (u'session_id',)

user

user_id

Operations

indico.modules.events.sessions.operations.**create_session**(*event*, *data*)
Create a new session with the information passed in the *data* argument

indico.modules.events.sessions.operations.**create_session_block**(*session_*, *data*)

indico.modules.events.sessions.operations.**delete_session**(*event_session*)
Delete session from the event

indico.modules.events.sessions.operations.**delete_session_block**(*session_block*)

indico.modules.events.sessions.operations.**update_session**(*event_session*, *data*)
Update a session based on the information in the *data*

indico.modules.events.sessions.operations.**update_session_block**(*session_block*, *data*)
Update a session block with data passed in the *data* argument

indico.modules.events.sessions.operations.**update_session_coordinator_privs**(*event*, *data*)

Utilities

class indico.modules.events.sessions.util.**SessionListToPDF**(*conf*, *sessions*)
Bases: indico.legacy.pdfinterface.base.PDFBase

getBody(*story=None*)

indico.modules.events.sessions.util.**can_manage_sessions**(*user*, *event*, *role=None*)
Check whether a user can manage any sessions in an event

indico.modules.events.sessions.util.**generate_pdf_from_sessions**(*event*, *sessions*)
Generate a PDF file from a given session list

indico.modules.events.sessions.util.**generate_spreadsheet_from_sessions**(*sessions*)
Generate spreadsheet data from a given session list.

Parameters **sessions** – The sessions to include in the spreadsheet

indico.modules.events.sessions.util.**get_events_with_linked_sessions**(*user*, *dt=None*)
Returns a dict with keys representing event_id and the values containing data about the user rights for sessions within the event

Parameters

- **user** – A *User*
- **dt** – Only include events taking place on/after that date

indico.modules.events.sessions.util.**get_session_ical_file**(*sess*)

indico.modules.events.sessions.util.**get_session_timetable_pdf**(*sess*, ***kwargs*)

indico.modules.events.sessions.util.**get_sessions_for_user**(*event*, *user*)

indico.modules.events.sessions.util.**has_sessions_for_user**(*event*, *user*)

indico.modules.events.sessions.util.**serialize_session_for_ical**(*sess*)

indico.modules.events.sessions.util.**session_coordinator_priv_enabled**(*event*, *priv*)
Check whether a coordinator privilege is enabled.

Currently the following privileges are available:

- manage-contributions
- manage-blocks

Parameters

- **event** – The *Event* to check for
- **priv** – The name of the privilege

Survey

Todo

Docstrings (module, models)

Models

class indico.modules.events.surveys.models.surveys.**Survey**(***kwargs*)
Bases: flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

anonymous
Whether submissions will not be linked to a user

can_submit(*user*)

close()

end_dt
Datetime when the survey is closed

event
The Event containing this survey

event_id
The ID of the event

has_ended

has_started

id
The ID of the survey

introduction

is_active

is_deleted
Whether the survey has been marked as deleted

is_visible

items
The list of items

locator
Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

new_submission_emails
Email addresses to notify about new submissions

notifications_enabled
Whether to send survey related notifications to users

notify_participants
Whether include Participants / Registrants when sending start notifications

open()

partial_completion
Whether answers can be saved without submitting the survey

private

questions
The list of questions

require_user
Whether submissions must be done by logged users

sections
The list of sections

send_start_notification()

send_submission_notification(*submission*)

start_dt
Datetime when the survey is open

start_notification_emails
Email addresses to notify about the start of a survey

start_notification_recipients
Returns all recipients of the notifications.

This includes both explicit recipients and, if enabled, participants of the event.

start_notification_sent
Whether start notification has been already sent

state

submission_limit
Maximum number of submissions allowed

submissions
The list of submissions

title
The title of the survey

uuid

class indico.modules.events.surveys.models.surveys.**SurveyState**
Bases: indico.util.struct.enum.IndicoEnum

active_and_answered = 4

active_and_clean = 3

finished = 5

not_ready = 1

ready_to_open = 2

class indico.modules.events.surveys.models.items.**SurveyItem**(***kwargs*)
Bases: flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

description
The description of the item

display_as_section
If a section should be rendered as a section

field_data
Field-specific data (such as choices for multi-select fields)

field_type
The type of the field used for the question

id
The ID of the item

is_required
If the question must be answered (wtforms DataRequired)

parent_id
The ID of the parent section item (NULL for top-level items, i.e. sections)

position
The position of the item in the survey form

survey_id
The ID of the survey

title
The title of the item

to_dict()
Return a json-serializable representation of this object.

Subclasses must add their own data to the dict.

type
The type of the survey item

class indico.modules.events.surveys.models.items.**SurveyItemType**
Bases: int, indico.util.struct.enum.IndicoEnum

question = 1

section = 2

text = 3

class indico.modules.events.surveys.models.items.**SurveyQuestion**(***kwargs*)
Bases: *indico.modules.events.surveys.models.items.SurveyItem*

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

description

display_as_section

field

field_data

field_type

get_summary(***kwargs*)
Returns the summary of answers submitted for this question.

id

is_required

locator

not_empty_answers

parent_id

position

survey_id

title

to_dict()

type

class indico.modules.events.surveys.models.items.**SurveySection**(***kwargs*)
Bases: *indico.modules.events.surveys.models.items.SurveyItem*

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

children
The child items of this section

description

display_as_section

field_data

field_type

id

is_required

locator

parent_id

position

survey_id

title

to_dict()

type

class indico.modules.events.surveys.models.items.**SurveyText**(***kwargs*)
Bases: *indico.modules.events.surveys.models.items.SurveyItem*

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

description

display_as_section

field_data

field_type

id

is_required

locator

parent_id

position

survey_id

title

to_dict()

type

class indico.modules.events.surveys.models.submissions.**SurveyAnswer**(***kwargs*)
Bases: flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

answer_data

data
The user's answer (no, not 42!) to the question

is_empty

question
The list of answers

question_id
The ID of the question

submission_id
The ID of the submission

class indico.modules.events.surveys.models.submissions.**SurveySubmission**(***kwargs*)
Bases: flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

answers
The list of answers

friendly_id
The human-friendly ID of the submission

id
The ID of the submission

is_anonymous
Whether the survey submission is anonymous

is_submitted
Whether the survey was submitted

locator

pending_answers
List of non-submitted answers

submitted_dt
The date/time when the survey was submitted

survey_id
The ID of the survey

user
The user who submitted the survey

user_id
The ID of the user who submitted the survey

Operations

indico.modules.events.surveys.operations.**add_survey_question**(*section*, *field_cls*, *data*)

Add a question to a survey.

Parameters

- **section** – The *SurveySection* to which the question will be added.
- **field_cls** – The field class of this question.
- **data** – The *FieldConfigForm.data* to populate the question with.

Returns The added *SurveyQuestion*.

indico.modules.events.surveys.operations.**add_survey_section**(*survey*, *data*)

Add a section to a survey.

Parameters

- **survey** – The *Survey* to which the section will be added.
- **data** – Attributes of the new *SurveySection*.

Returns The added *SurveySection*.

indico.modules.events.surveys.operations.**add_survey_text**(*section*, *data*)

Add a text item to a survey.

Parameters

- **section** – The *SurveySection* to which the question will be added.
- **data** – The *TextForm.data* to populate the question with.

Returns The added *SurveyText*.

Utilities

indico.modules.events.surveys.util.**generate_spreadsheet_from_survey**(*survey*, *submission_ids*)

Generates spreadsheet data from a given survey.

Parameters

- **survey** – *Survey* for which the user wants to export submissions
- **submission_ids** – The list of submissions to include in the file

indico.modules.events.surveys.util.**get_events_with_submitted_surveys**(*user*, *dt=None*)

Gets the IDs of events where the user submitted a survey.

Parameters

- **user** – A *User*
- **dt** – Only include events taking place on/after that date

Returns A set of event ids

indico.modules.events.surveys.util.**is_submission_in_progress**(*survey*)
Check whether the current user has a survey submission in progress

indico.modules.events.surveys.util.**make_survey_form**(*survey*)
Creates a WTForm from survey questions.

Each question will use a field named `question_ID`.

Parameters **survey** – The *Survey* for which to create the form.

Returns An *IndicoForm* subclass.

indico.modules.events.surveys.util.**query_active_surveys**(*event*)

indico.modules.events.surveys.util.**save_submitted_survey_to_session**(*submission*)
Save submission of a survey to session for further checks

indico.modules.events.surveys.util.**was_survey_submitted**(**args*, ***kwargs*)
Check whether the current user has submitted a survey

Timetable

Todo

Docstring (module, models, operations, utilities)

Models

class indico.modules.events.timetable.models.breaks.**Break**(***kwargs*)
Bases: indico.core.db.sqlalchemy.descriptions.DescriptionMixin, indico.core.db.sqlalchemy.colors.ColorMixin, indico.core.db.sqlalchemy.locations.LocationMixin, flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

background_color

can_access(*user*)

default_colors = ColorTuple(text=u'202020', background=u'90c0f0')

default_render_mode = 2

duration

end_dt

event

id

inherit_location

location_backref_name = u'breaks'

location_parent

locator
Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

own_address

own_room

own_room_id

own_room_name

own_venue

own_venue_id

own_venue_name

possible_render_modes = set([<RenderMode.markdown: 2>])

render_mode = 2

start_dt

text_color

title

class `indico.modules.events.timetable.models.entries.`**TimetableEntry**(***kwargs*)
Bases: `flask_sqlalchemy.Model`

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

break_

break_id

can_view(*user*)
Checks whether the user will see this entry in the timetable.

children

contribution

contribution_id

duration

end_dt

event

event_id

extend_end_dt(*end_dt*)

extend_parent(*by_start=True*, *by_end=True*)
Extend start/end of parent objects if needed.

No extension if performed for entries crossing a day boundary in the event timezone.

Parameters

- **by_start** – Extend parent by start datetime.
- **by_end** – Extend parent by end datetime.

extend_start_dt(*start_dt*)

id

is_parallel(*in_session=False*)

locator
Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

move(*start_dt*)
Move the entry to start at a different time.

This method automatically moves children of the entry to preserve their start time relative to the parent's start time.

move_next_to(*sibling*, *position=u'before'*)

object

parent_id

session_block

session_block_id

session_siblings

siblings

siblings_query

start_dt

type

class indico.modules.events.timetable.models.entries.**TimetableEntryType**
Bases: indico.util.struct.enum.RichIntEnum

BREAK = 3

CONTRIBUTION = 2

SESSION_BLOCK = 1

Operations

indico.modules.events.timetable.operations.**can_swap_entry**(*entry*, *direction*, *in_session=False*)

indico.modules.events.timetable.operations.**create_break_entry**(*event*, *data*, *session_block=None*)

indico.modules.events.timetable.operations.**create_session_block_entry**(*session_*, *data*)

indico.modules.events.timetable.operations.**create_timetable_entry**(*event*, *data*, *parent=None*, *extend_parent=False*)

indico.modules.events.timetable.operations.**delete_timetable_entry**(*entry*, *log=True*)

indico.modules.events.timetable.operations.**fit_session_block_entry**(*entry*, *log=True*)

indico.modules.events.timetable.operations.**get_sibling_entry**(*entry*, *direction*, *in_session=False*)

indico.modules.events.timetable.operations.**move_timetable_entry**(*entry*, *parent=None*, *day=None*)

Move the *entry* to another session or top-level timetable

Parameters

- **entry** – *TimetableEntry* to be moved
- **parent** – If specified then the entry will be set as a child of parent
- **day** – If specified then the entry will be moved to the top-level timetable on this day

indico.modules.events.timetable.operations.**schedule_contribution**(*contribution*, *start_dt*, *session_block=None*, *extend_parent=False*)

indico.modules.events.timetable.operations.**swap_timetable_entry**(*entry*, *direction*, *session_=None*)

Swap entry with closest gap or non-parallel sibling

indico.modules.events.timetable.operations.**update_break_entry**(*break_*, *data*)

indico.modules.events.timetable.operations.**update_timetable_entry**(*entry*, *data*)

indico.modules.events.timetable.operations.**update_timetable_entry_object**(*entry*, *data*)

Update the *object* of a timetable entry according to its type

Utilities

indico.modules.events.timetable.util.**find_latest_entry_end_dt**(*obj*, *day=None*)

Get the latest end datetime for timetable entries within the object.

Parameters

- **obj** – The `Event` or `SessionBlock` that will be used to look for timetable entries.
- **day** – The local event date to look for timetable entries. Applicable only to `Event`.

Returns The end datetime of the timetable entry finishing the latest. `None` if no entry was found.

indico.modules.events.timetable.util.**find_next_start_dt**(*duration*, *obj*, *day=None*, *force=False*)

Find the next most convenient start date fitting a duration within an object.

Parameters

- **duration** – Duration to fit into the event/session-block.
- **obj** – The `Event` or `SessionBlock` the duration needs to fit into.
- **day** – The local event date where to fit the duration in case the object is an event.
- **force** – Gives earliest datetime if the duration doesn't fit.

Returns The end datetime of the latest scheduled entry in the object if the duration fits then. It it doesn't, the latest datetime that fits it. `None` if the duration cannot fit in the object, earliest datetime if `force` is `True`.

indico.modules.events.timetable.util.**get_category_timetable**(*categ_ids*, *start_dt*, *end_dt*, *detail_level=u'event'*, *tz=<UTC>*, *from_categ=None*, *grouped=True*)

Retrieve time blocks that fall within a specific time interval for a given set of categories.

Parameters

- **categ_ids** – iterable containing list of category IDs
- **start_dt** – start of search interval (`datetime`, expected to be in display timezone)
- **end_dt** – end of search interval (`datetime` in expected to be in display timezone)
- **detail_level** – the level of detail of information (`event|session|contribution`)
- **tz** – the `timezone` information should be displayed in

- **from_categ** – Category that will be taken into account to calculate visibility
- **grouped** – Whether to group results by start date

Returns a dictionary containing timetable information in a structured way. See source code for examples.

indico.modules.events.timetable.util.**get_nested_entries**(**args*, ***kwargs*)

indico.modules.events.timetable.util.**get_session_block_entries**(*event*, *day*)
Returns a list of event top-level session blocks for the given *day*

indico.modules.events.timetable.util.**get_time_changes_notifications**(*changes*, *tzinfo*, *entry=None*)

indico.modules.events.timetable.util.**get_timetable_offline_pdf_generator**(*event*)

indico.modules.events.timetable.util.**get_top_level_entries**(**args*, ***kwargs*)

indico.modules.events.timetable.util.**render_entry_info_balloon**(*entry*, *editable=False*, *sess=None*)

indico.modules.events.timetable.util.**render_session_timetable**(*session*, *timetable_layout=None*, *management=False*)

indico.modules.events.timetable.util.**shift_following_entries**(*entry*, *shift*, *session_=None*)
Reschedules entries starting after the given entry by the given shift.

class indico.modules.events.timetable.reschedule.**RescheduleMode**
Bases: unicode, indico.util.struct.enum.RichEnum

duration = u'duration'

none = u'none'

time = u'time'

title

class indico.modules.events.timetable.reschedule.**Rescheduler**(*event*, *mode*, *day*, *session=None*, *session_block=None*, *fit_blocks=False*, *gap=datetime.timedelta(0)*)
Bases: object

Compacts the the schedule of an event day by either adjusting start times or durations of timetable entries.

Parameters

- **event** – The event of which the timetable entries should be rescheduled.
- **mode** – A *RescheduleMode* value specifying whether the duration or start time should be adjusted.
- **day** – A *date* specifying the day to reschedule (the day of the timetable entries are determined using the event's timezone)
- **session** – If specified, only blocks of that session will be rescheduled, ignoring any other timetable entries. Cannot be combined with *session_block*.

- **session_block`** – If specified, only entries inside that block will be rescheduled. Cannot be combined with *session*.
- **fit_blocks** – Whether session blocks should be resized to exactly fit their contents before the actual rescheduling operation.
- **gap** – A timedelta specifying the cap between rescheduled timetable entries.

run()
Perform the rescheduling

Track

Todo

Docstring (module, models, operations)

Models

class indico.modules.events.tracks.models.tracks.**Track**(**kwargs*)
Bases: indico.core.db.sqlalchemy.descriptions.DescriptionMixin, flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

abstract_reviewers

abstracts_accepted

abstracts_reviewed

abstracts_submitted

can_convene(*user*)

can_delete(*user*)

can_review_abstracts(*user*)

code

conveners

default_render_mode = 2

event

event_id

full_title

id

locator
Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

position

possible_render_modes = set([<RenderMode.markdown: 2>])

render_mode = 2

short_title

title

Operations

indico.modules.events.tracks.operations.**create_track**(*event*, *data*)

indico.modules.events.tracks.operations.**delete_track**(*track*)

indico.modules.events.tracks.operations.**update_program**(*event*, *data*)

indico.modules.events.tracks.operations.**update_track**(*track*, *data*)

Static site

Todo

Doctrings (module, utilities)

Models

class indico.modules.events.static.models.static.**StaticSite**(***kwargs*)
Bases: indico.core.storage.models.StoredFileMixin, flask_sqlalchemy.Model

Static site for an Indico event.

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

add_file_date_column = False

content_type
The MIME type of the file

created_dt = None

creator
The user who created the static site

creator_id
ID of the user who created the static site

event
The Event this static site is associated with

event_id
ID of the event

file_required = False

filename
The name of the file

id
Entry ID

locator

md5
An MD5 hash of the file.

Automatically assigned when *save()* is called.

requested_dt
The date and time the static site was requested

size
The size of the file (in bytes).

Automatically assigned when *save()* is called.

state
The state of the static site (a *StaticSiteState* member)

storage_backend

storage_file_id

class indico.modules.events.static.models.static.**StaticSiteState**
Bases: indico.util.struct.enum.RichIntEnum

expired = 4

failed = 3

pending = 0

running = 1

success = 2

Utilities

indico.modules.events.static.util.**url_to_static_filename**(*endpoint*, *url*)

Category

Todo

Docstrings (module, model, operations, utilities)

Models

class indico.modules.categories.models.categories.**Category**(***kwargs*)

Bases: indico.core.db.sqlalchemy.searchable_titles.SearchableTitleMixin, indico.core.db.sqlalchemy.descriptions.DescriptionMixin, indico.core.db.sqlalchemy.protection.ProtectionManagersMixin, indico.core.db.sqlalchemy.attachments.AttachedItemsMixin, flask_sqlalchemy.Model

An Indico category

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

ATTACHMENT_FOLDER_ID_COLUMN = u'category_id'

access_key = None

acl_entries

allow_no_access_contact = True

can_create_events(*user*)

Check whether the user can create events in the category.

chain_query

Get a query object for the category chain.

The query retrieves the root category first and then all the intermediate categories up to (and including) this category.

children

deep_children_query

Get a query object for all subcategories.

This includes subcategories at any level of nesting.

default_event_themes

default_render_mode = 2

default_ticket_template

default_ticket_template_id

disallowed_protection_modes = frozenset([])

display_tzinfo

The tzinfo of the category or the one specified by the user

effective_icon_url

Get the HTTP URL of the icon (possibly inherited).

event_creation_notification_emails

event_creation_restricted

event_message

event_message_mode

classmethod get_icon_data_cte()

classmethod get_protection_cte()

get_protection_parent_cte()

classmethod get_root()
Get the root category

classmethod get_tree_cte(*col=u'id'*)
Create a CTE for the category tree.

The CTE contains the following columns:

- id – the category id
- **path – an array containing the path from the root to** the category itself
- is_deleted – whether the category is deleted

Parameters **col** – The name of the column to use in the path or a callable receiving the category alias that must return the expression used for the 'path' retrieved by the CTE.

has_effective_icon

has_icon

has_logo

has_only_events

icon

icon_metadata

icon_url
Get the HTTP URL of the icon.

id

inheriting_have_acl = True

is_deleted

is_descendant_of(*categ*)

is_empty

is_root

locator
Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named locator as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

logo

logo_metadata

logo_url
Get the HTTP URL of the logo.

move (*target*)
Move the category into another category.

notify_managers

nth_parent (*n_categs*, *fail_on_overflow=True*)
Return the nth parent of the category.

Parameters

- **n_categs** – the number of categories to go up
- **fail_on_overflow** – whether to fail if we try to go above the root category

Returns *Category* object or None (only if `fail_on_overflow` is not set)

own_no_access_contact

own_visibility_horizon
Get the highest category this one would like to be visible from (configured visibility).

parent_chain_query
Get a query object for the category's parent chain.

The query retrieves the root category first and then all the intermediate categories up to (excluding) this category.

parent_id

position

possible_render_modes = set([<RenderMode.markdown: 2>])

protection_mode

protection_parent

real_visibility_horizon
Get the highest category this one is actually visible from (as limited by categories above).

render_mode = 2

suggestions_disabled

timezone

title

tzinfo

url

visibility

visibility_horizon_query
Get a query object that returns the highest category this one is visible from.

visible_categories_cte
Get a sqlalchemy select for the visible categories within this category, including the category itself.

visible_categories_query
Get a query object for the visible categories within this category, including the category itself.

class indico.modules.categories.models.categories.**EventMessageMode**
Bases: indico.util.struct.enum.RichIntEnum

danger = 3

disabled = 0

info = 1

warning = 2

class indico.modules.categories.models.principals.**CategoryPrincipal**(***kwargs*)
Bases: indico.core.db.sqlalchemy.principals.PrincipalRolesMixin, flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

allow_networks = True

category_id
The ID of the associated event

email = None

full_access

id
The ID of the acl entry

ip_network_group

ip_network_group_id

local_group

local_group_id

multipass_group_name

multipass_group_provider

principal_backref_name = u'in_category_acls'

principal_for = u'Category'

read_access

roles

type

unique_columns = (u'category_id',)

user

user_id

class indico.modules.categories.models.settings.**CategorySetting**(***kwargs*)
Bases: indico.core.settings.models.base.JSONSettingsBase, flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

category

category_id

id

module

name

value

Operations

indico.modules.categories.operations.**create_category**(*parent*, *data*)

indico.modules.categories.operations.**delete_category**(*category*)

indico.modules.categories.operations.**move_category**(*category*, *target_category*)

indico.modules.categories.operations.**update_category**(*category*, *data*, *skip=()*)

Utilities

indico.modules.categories.util.**get_attachment_count**(*category_id=None*)
Get the number of attachments in events in a category.

Parameters **category_id** – The category ID to get statistics for. Attachments from subcategories are also included.

Returns The number of attachments

indico.modules.categories.util.**get_category_stats**(**args*, ***kwargs*)
Get category statistics.

This function is mainly a helper so we can get and cache all values at once and keep a last-update timestamp.

Parameters **category_id** – The category ID to get statistics for. Subcategories are also included.

indico.modules.categories.util.**get_contribs_by_year**(*category_id=None*)
Get the number of contributions for each year.

Parameters **category_id** – The category ID to get statistics for. Contributions from subcategories are also included.

Returns An *OrderedDict* mapping years to contribution counts.

indico.modules.categories.util.**get_events_by_year**(*category_id=None*)
Get the number of events for each year.

Parameters **category_id** – The category ID to get statistics for. Events from subcategories are also included.

Returns An *OrderedDict* mapping years to event counts.

indico.modules.categories.util.**get_image_data**(*image_type*, *category*)

indico.modules.categories.util.**get_upcoming_events**(**args*, ***kwargs*)
Get the global list of upcoming events

indico.modules.categories.util.**get_visibility_options**(*category_or_event*, *allow_invisible=True*)
Return the visibility options available for the category or event.

indico.modules.categories.serialize.**serialize_categories_ical**(*category_ids*, *user*, *event_filter=True*, *event_filter_fn=None*, *update_query=None*)
Export the events in a category to iCal

Parameters

- **category_ids** – Category IDs to export
- **user** – The user who needs to be able to access the events
- **event_filter** – A SQLalchemy criterion to restrict which events will be returned. Usually something involving the start/end date of the event.
- **event_filter_fn** – A callable that determines which events to include (after querying)
- **update_query** – A callable that can update the query used to retrieve the events. Must return the updated query object.

indico.modules.categories.serialize.**serialize_category**(*category*, *with_favorite=False*, *with_path=False*, *parent_path=None*, *child_path=None*)

indico.modules.categories.serialize.**serialize_category_atom**(*category*, *url*, *user*, *event_filter*)
Export the events in a category to Atom

Parameters

- **category** – The category to export
- **url** – The URL of the feed
- **user** – The user who needs to be able to access the events
- **event_filter** – A SQLalchemy criterion to restrict which events will be returned. Usually something involving the start/end date of the event.

indico.modules.categories.serialize.**serialize_category_chain**(*category*, *include_children=False*, *include_parents=False*)

Settings

class indico.modules.categories.settings.**CategorySettingsProxy**(*module*, *defaults=None*, *strict=True*, *acls=None*, *converters=None*)

Bases: indico.core.settings.proxy.SettingsProxyBase

Proxy class to access category-specific settings for a certain module.

delete(*category*, **args*, ***kwargs*)
Delete settings.

Parameters

- **category** – Category (or its ID)
- **names** – One or more names of settings to delete

delete_all(*category*, **args*, ***kwargs*)
Delete all settings.

Parameters **category** – Category (or its ID)

get(*category*, **args*, ***kwargs*)
Retrieve the value of a single setting.

Parameters

- **category** – Category (or its ID)
- **name** – Setting name
- **default** – Default value in case the setting does not exist

Returns The settings's value or the default value

get_all(*category*, **args*, ***kwargs*)
Retrieve all settings.

Parameters

- **category** – Category (or its ID)
- **no_defaults** – Only return existing settings and ignore defaults.

Returns Dict containing the settings

query
Return a query object filtering by the proxy's module.

set(*category*, **args*, ***kwargs*)
Set a single setting.

Parameters

- **category** – Category (or its ID)
- **name** – Setting name
- **value** – Setting value; must be JSON-serializable

set_multi(*category*, **args*, ***kwargs*)
Set multiple settings at once.

Parameters

- **category** – Category (or its ID)
- **items** – Dict containing the new settings

User

Todo

Docstrings (module, models, utilities)

Models

class indico.modules.users.models.users.**NameFormat**
Bases: indico.util.struct.enum.RichIntEnum

f_last = 3

f_last_upper = 7

first_last = 0

first_last_upper = 4

last_f = 2

last_f_upper = 6

last_first = 1

last_first_upper = 5

class indico.modules.users.models.users.**PersonMixin**
Bases: object

Add convenience properties and methods to person classes.

Assumes the following attributes exist: * first_name * last_name * title

display_full_name
Return the full name using the user's preferred name format.

full_name
Return the person's name in 'Firstname Lastname' notation.

get_full_name(*last_name_first=True*, *last_name_upper=True*, *abbrev_first_name=True*, *show_title=False*, *_show_empty_names=False*)
Return the person's name in the specified notation.

Note: Do not use positional arguments when calling this method. Always use keyword arguments!

Parameters

- **last_name_first** – if "lastname, firstname" instead of "firstname lastname" should be used
- **last_name_upper** – if the last name should be all-uppercase
- **abbrev_first_name** – if the first name should be abbreviated to use only the first character
- **show_title** – if the title of the person should be included

name
Return the person's name in 'Firstname Lastname' notation.

title
The title of the user

class indico.modules.users.models.users.**User**(***kwargs*)
Bases: *indico.modules.users.models.users.PersonMixin*, flask_sqlalchemy.Model

Indico users

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

abstract_email_log_entries

abstracts

address
the address of the user

affiliation
the affiliation of the user

all_emails
all emails of the user. read-only; use it only for searching by email! also, do not use it between modifying *email* or *secondary_emails* and a session expire/commit!

api_key
the active API key of the user

as_avatar

as_legacy

as_principal
The serializable principal identifier of this user

avatar_css

can_be_modified(*user*)
If this user can be modified by the given user

created_events

email
the primary email address of the user

external_identities
The external identities of the user

favorite_categories
the users's favorite categories

favorite_users
the users's favorite users

first_name
the first name of the user

get_full_name(**args*, ***kwargs*)

static get_system_user()

id
the unique id of the user

identities
the identities used by this user

in_attachment_acls

in_attachment_folder_acls

is_admin
if the user is an administrator with unrestricted access to everything

is_blocked
if the user has been blocked

is_deleted
if the user is deleted (e.g. due to a merge)

is_group = False

is_network = False

is_pending
if the user is pending (e.g. never logged in, only added to some list)

is_single_person = True

is_system
if the user is the default system user

iter_identifiers(*check_providers=False*, *providers=None*)
Yields `(provider, identifier)` tuples for the user.

Parameters

- **check_providers** – If True, providers are searched for additional identifiers once all existing identifiers have been yielded.
- **providers** – May be a set containing provider names to get only identifiers from the specified providers.

judged_abstracts

last_name
the last/family name of the user

local_identities
The local identities of the user

local_identity
The main (most recently used) local identity

locator
Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

make_email_primary(*email*)
Promotes a secondary email address to the primary email address

Parameters **email** – an email address that is currently a secondary email

merged_into_id
the id of the user this user has been merged into

merged_into_user
the user this user has been merged into

modified_abstracts

old_api_keys
the previous API keys of the user

phone
the phone number of the user

principal_order = 0

principal_type = 1

secondary_emails
any additional emails the user might have

secondary_local_identities
The local identities of the user except the main one

settings
Returns the user settings proxy for this user

suggested_categories
the user's category suggestions

synced_fields
The fields of the user whose values are currently synced.

This set is always a subset of the synced fields define in synced fields of the idp in 'indico.conf'.

synced_values
The values from the synced identity for the user.

Those values are not the actual user's values and might differ if they are not set as synchronized.

synchronize_data(*refresh=False*)
Synchronize the fields of the user from the sync identity.

This will take only into account *synced_fields*.

Parameters **refresh** – bool – Whether to refresh the synced identity with the sync provider before instead of using the stored data. (Only if the sync provider supports refresh.)

class indico.modules.users.models.users.**UserTitle**
Bases: indico.util.struct.enum.RichIntEnum

dr = 4

mr = 1

mrs = 3

ms = 2

none = 0

prof = 5

indico.modules.users.models.users.**format_display_full_name**(*user*, *obj*)

indico.modules.users.models.users.**syncable_fields = {u'affiliation': lu'affiliation', u'first_name': lu'first nam**
Fields which can be synced as keys and a mapping to a more human readable version, used for flashing messages

class indico.modules.users.models.affiliations.**UserAffiliation**(***kwargs*)
Bases: flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

id
the unique id of the affiliations

name
the affiliation

user_id
the id of the associated user

class indico.modules.users.models.emails.**UserEmail**(***kwargs*)
Bases: flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

email
the email address

id
the unique id of the email address

is_primary
if the email is the user's primary email

is_user_deleted
if the user is marked as deleted (e.g. due to a merge). DO NOT use this flag when actually deleting an email

user_id
the id of the associated user

class indico.modules.users.models.suggestions.**SuggestedCategory**(***kwargs*)
Bases: flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

category

category_id

is_ignored

classmethod merge_users(*target*, *source*)
Merge the suggestions for two users.

Parameters

- **target** – The target user of the merge.
- **source** – The user that is being merged into *target*.

score

user_id

class indico.modules.users.models.settings.**UserSetting**(***kwargs*)
Bases: indico.core.settings.models.base.JSONSettingsBase, flask_sqlalchemy.Model

User-specific settings

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

id

module

name

user

user_id

value

class indico.modules.users.models.settings.**UserSettingsProxy**(*module*, *defaults=None*, *strict=True*, *acls=None*, *converters=None*)
Bases: indico.core.settings.proxy.SettingsProxyBase

Proxy class to access user-specific settings for a certain module

delete(*user*, **args*, ***kwargs*)
Deletes settings.

Parameters

- **user** – {'user': user} or {'user_id': id}
- **names** – One or more names of settings to delete

delete_all(*user*, **args*, ***kwargs*)
Deletes all settings.

Parameters **user** – {'user': user} or {'user_id': id}

get(*user*, **args*, ***kwargs*)
Retrieves the value of a single setting.

Parameters

- **user** – {'user': user} or {'user_id': id}
- **name** – Setting name
- **default** – Default value in case the setting does not exist

Returns The settings's value or the default value

get_all(*user*, **args*, ***kwargs*)
Retrieves all settings

Parameters

- **user** – {'user': user} or {'user_id': id}
- **no_defaults** – Only return existing settings and ignore defaults.

Returns Dict containing the settings

query
Returns a query object filtering by the proxy's module.

set(*user*, **args*, ***kwargs*)
Sets a single setting.

Parameters

- **user** – {'user': user} or {'user_id': id}
- **name** – Setting name
- **value** – Setting value; must be JSON-serializable

set_multi(*user*, **args*, ***kwargs*)
Sets multiple settings at once.

Parameters

- **user** – {'user': user} or {'user_id': id}
- **items** – Dict containing the new settings

indico.modules.users.models.settings.**user_or_id**(*f*)

Operations

indico.modules.users.operations.**create_user**(*email*, *data*, *identity=None*, *settings=None*, *other_emails=None*, *from_moderation=True*)
Create a new user.

This may also convert a pending user to a proper user in case the email address matches such a user.

Parameters

- **email** – The primary email address of the user.
- **data** – The data used to populate the user.
- **identity** – An *Identity* to associate with the user.
- **settings** – A dict containing user settings.
- **other_emails** – A set of email addresses that are also used to check for a pending user. They will also be added as secondary emails to the user.
- **from_moderation** – Whether the user was created through the moderation process or manually by an admin.

Utilities

indico.modules.users.util.**get_admin_emails**()
Get the email addresses of all Indico admins

indico.modules.users.util.**get_color_for_username**(*username*)

indico.modules.users.util.**get_linked_events**(*user*, *dt*, *limit=None*)
Get the linked events and the user's roles in them

Parameters

- **user** – A *User*
- **dt** – Only include events taking place on/after that date
- **limit** – Max number of events

indico.modules.users.util.**get_related_categories**(*user*, *detailed=True*)
Gets the related categories of a user for the dashboard

indico.modules.users.util.**get_suggested_categories**(*user*)
Gets the suggested categories of a user for the dashboard

indico.modules.users.util.**get_user_by_email**(*email*, *create_pending=False*)
finds a user based on his email address.

Parameters

- **email** – The email address of the user.
- **create_pending** – If True, this function searches for external users and creates a new pending User in case no existing user was found.

Returns A *User* instance or `None` if not exactly one user was found.

indico.modules.users.util.**merge_users**(*source*, *target*, *force=False*)
Merge two users together, unifying all related data

Parameters

- **source** – source user (will be set as deleted)
- **target** – target user (final)

indico.modules.users.util.**search_users**(*exact=False*, *include_deleted=False*, *include_pending=False*, *external=False*, *allow_system_user=False*, ***criteria*)
Searches for users.

Parameters

- **exact** – Indicates if only exact matches should be returned. This is MUCH faster than a non-exact saerch, especially when searching external users.
- **include_deleted** – Indicates if also users marked as deleted should be returned.
- **include_pending** – Indicates if also users who are still pending should be returned.
- **external** – Indicates if identity providers should be searched for matching users.
- **allow_system_user** – Whether the system user may be returned in the search results.
- **criteria** – A dict containing any of the following keys: first_name, last_name, email, affiliation, phone, address

Returns A set of matching users. If *external* was set, it may contain both `IdentityInfo` objects for external users not yet in Indico and *User* objects for existing users.

indico.modules.users.util.**serialize_user**(*user*)
Serialize user to JSON-like object

class indico.modules.users.ext.**ExtraUserPreferences**(*user*)
Bases: `object`

Defines additional user preferences

To use this class, subclass it and override *defaults*, *fields* and *save* to implement your custom logic.

extend_defaults(*defaults*)
Adds values to the FormDefaults.

extend_form(*form_class*)
Creates a subclass of the form containing the extra field

fields = {}
a dict containing all the fields that should be added to the user preferences

load()
Returns a dict with the current values for the user

process_form_data(*data*)
Processes and saves submitted data.

This modifies *data* so the core code doesn't receive any extra data it doesn't expect.

save(*data*)
Saves the updated settings

Attachment

Todo

Docstrings (module, models, operations)

Models

class indico.modules.attachments.models.attachments.**Attachment**(***kwargs*)
Bases: `indico.core.db.sqlalchemy.protection.ProtectionMixin`, `indico.core.storage.models.VersionedResourceMixin`, `flask_sqlalchemy.Model`

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

absolute_download_url
The absolute download url for the attachment

access_key = None

acl
The ACL of the folder (used for ProtectionMode.protected)

acl_entries

all_files

can_access(*user*, **args*, ***kwargs*)
Checks if the user is allowed to access the attachment.

This is the case if the user has access to see the attachment or if the user can manage attachments for the linked object.

description
The description of the attachment

download_url
The download url for the attachment

file

file_id

folder
The folder containing the attachment

folder_id
The ID of the folder the attachment belongs to

get_download_url(*absolute=False*)
Returns the download url for the attachment.

During static site generation this returns a local URL for the file or the target URL for the link.

Parameters **absolute** – If the returned URL should be absolute.

id
The ID of the attachment

is_deleted
If the attachment has been deleted

link_url
The target URL for a link attachment

locator

modified_dt
The date/time when the attachment was created/modified

own_no_access_contact = None

protection_mode

protection_parent

stored_file_class
alias of *AttachmentFile*

stored_file_fkey = u'attachment_id'

stored_file_table = u'attachments.files'

title
The name of the attachment

type
The type of the attachment (file or link)

user
The user who created the attachment

user_id
The ID of the user who created the attachment

class indico.modules.attachments.models.attachments.**AttachmentFile**(***kwargs*)
Bases: indico.core.storage.models.StoredFileMixin, flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

attachment_id
The ID of the associated attachment

content_type
The MIME type of the file

created_dt
The date/time when the file was uploaded

filename
The name of the file

id
The ID of the file

is_previewable

md5
An MD5 hash of the file.

Automatically assigned when *save()* is called.

size
The size of the file (in bytes).

Automatically assigned when *save()* is called.

storage_backend

storage_file_id

user
The user who uploaded the file

user_id
The user who uploaded the file

version_of = u'attachment'

class indico.modules.attachments.models.attachments.**AttachmentType**
Bases: indico.util.struct.enum.RichIntEnum

file = 1

link = 2

class indico.modules.attachments.models.folders.**AttachmentFolder**(***kwargs*)
Bases: indico.core.db.sqlalchemy.links.LinkMixin, indico.core.db.sqlalchemy.protection.ProtectionMixin, flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

access_key = None

acl
The ACL of the folder (used for ProtectionMode.protected)

acl_entries

attachments
The list of attachments that are not deleted, ordered by name

can_access(*user*, **args*, ***kwargs*)
Checks if the user is allowed to access the folder.

This is the case if the user has access the folder or if the user can manage attachments for the linked object.

can_view(*user*)
Checks if the user can see the folder.

This does not mean the user can actually access its contents. It just determines if it is visible to him or not.

category

category_id

contribution

contribution_id

description
The description of the folder

event

event_id

events_backref_name = u'all_attachment_folders'

classmethod get_for_linked_object(*linked_object*, *preload_event=False*)
Gets the attachments for the given object.

This only returns attachments that haven't been deleted.

Parameters

- **linked_object** – A category, event, session, contribution or subcontribution.
- **preload_event** – If all attachments for the same event should be pre-loaded and cached in the app context. This must not be used when `linked_object` is a category.

classmethod get_or_create(*linked_object*, *title=None*)
Gets a folder for the given object or creates it.

If no folder title is specified, the default folder will be used. It is the caller's responsibility to add the folder or an object (such as an attachment) associated with it to the SQLAlchemy session using `db.session.add(...)`.

classmethod get_or_create_default(*linked_object*)
Gets the default folder for the given object or creates it.

id
The ID of the folder

is_always_visible
If the folder is always visible (even if you cannot access it)

is_default
If the folder is the default folder (used for "folder-less" files)

is_deleted
If the folder has been deleted

link_backref_lazy = u'dynamic'

link_backref_name = u'attachment_folders'

link_type

linked_event

linked_event_id

locator
Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

own_no_access_contact = None

protection_mode

protection_parent

session

session_id

subcontribution

subcontribution_id

title
The name of the folder (None for the default folder)

unique_links = u'is_default'

class indico.modules.attachments.models.principals.**AttachmentFolderPrincipal**(***kwargs*)
Bases: indico.core.db.sqlalchemy.principals.PrincipalMixin, flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

email = None

folder_id
The ID of the associated folder

id
The ID of the acl entry

ip_network_group = None

ip_network_group_id = None

local_group

local_group_id

multipass_group_name

multipass_group_provider

principal_backref_name = u'in_attachment_folder_acls'

type

unique_columns = (u'folder_id',)

user

user_id

class indico.modules.attachments.models.principals.**AttachmentPrincipal**(***kwargs*)
Bases: indico.core.db.sqlalchemy.principals.PrincipalMixin, flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

attachment_id
The ID of the associated attachment

email = None

id
The ID of the acl entry

ip_network_group = None

ip_network_group_id = None

local_group

local_group_id

multipass_group_name

multipass_group_provider

principal_backref_name = u'in_attachment_acls'

type

unique_columns = (u'attachment_id',)

user

user_id

Operations

indico.modules.attachments.operations.**add_attachment_link**(*data*, *linked_object*)
Add a link attachment to linked_object

Utilities

indico.modules.attachments.util.**can_manage_attachments**(*obj*, *user*)
Checks if a user can manage attachments for the object

indico.modules.attachments.util.**get_attached_folders**(*linked_object*, *include_empty=True*, *include_hidden=True*, *preload_event=False*)
Return a list of all the folders linked to an object.

Parameters

- **linked_object** – The object whose attachments are to be returned
- **include_empty** – Whether to return empty folders as well.
- **include_hidden** – Include folders that the user can't see
- **preload_event** – in the process, preload all objects tied to the corresponding event and keep them in cache

indico.modules.attachments.util.**get_attached_items**(*linked_object*, *include_empty=True*, *include_hidden=True*, *preload_event=False*)
Return a structured representation of all the attachments linked to an object.

Parameters

- **linked_object** – The object whose attachments are to be returned
- **include_empty** – Whether to return empty folders as well.
- **include_hidden** – Include folders that the user can't see
- **preload_event** – in the process, preload all objects tied to the corresponding event and keep them in cache

indico.modules.attachments.util.**get_default_folder_names**()

indico.modules.attachments.util.**get_event**(*linked_object*)

indico.modules.attachments.util.**get_nested_attached_items**(*obj*)
Returns a structured representation of all attachments linked to an object and all its nested objects.

Parameters **obj** – A Event, Session, Contribution or SubContribution object.

class indico.modules.attachments.preview.**ImagePreviewer**
Bases: *indico.modules.attachments.preview.Previewer*

ALLOWED_CONTENT_TYPE = <_sre.SRE_Pattern object>

TEMPLATE = u'image_preview.html'

class indico.modules.attachments.preview.**MarkdownPreviewer**
Bases: *indico.modules.attachments.preview.Previewer*

ALLOWED_CONTENT_TYPE = <_sre.SRE_Pattern object>

classmethod generate_content(*attachment*)

class indico.modules.attachments.preview.**PDFPreviewer**
Bases: *indico.modules.attachments.preview.Previewer*

ALLOWED_CONTENT_TYPE = <_sre.SRE_Pattern object>

TEMPLATE = u'iframe_preview.html'

classmethod can_preview(*attachment_file*)

class indico.modules.attachments.preview.**Previewer**
Bases: object

Base class for file previewers

To create a new file prewiewer, subclass this class and register it using the *get_file_previewers* signal.

ALLOWED_CONTENT_TYPE = None

TEMPATE = None

TEMPLATES_DIR = u'attachments/previewers/'

classmethod can_preview(*attachment_file*)
Checks if the content type of the file matches the allowed content type of files that the previewer can be used for.

classmethod generate_content(*attachment*)
Generates the HTML output of the file preview

class indico.modules.attachments.preview.**TextPreviewer**
Bases: *indico.modules.attachments.preview.Previewer*

ALLOWED_CONTENT_TYPE = <_sre.SRE_Pattern object>

classmethod generate_content(*attachment*)

indico.modules.attachments.preview.**get_file_previewer**(*attachment_file*)
Returns a file previewer for the given attachment file based on the file's content type.

indico.modules.attachments.preview.**get_file_previewers**()

Room booking

Todo

Docstrings (module, models, utilities, services)

Models

class indico.modules.rb.models.rooms.**Room**(***kwargs*)

Bases: indico.core.db.sqlalchemy.util.cache._CacheVersionMixin, flask_sqlalchemy.Model, indico.util.serializer.Serializer

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

attributes

available_equipment

blocked_rooms

bookable_hours

booking_limit_days

booking_url

building

can_be_booked(*user*, *ignore_admin=False*)

Reservable rooms which does not require pre-booking can be booked by anyone. Other rooms - only by their responsibles.

can_be_deleted(*user*)

can_be_modified(*user*)

Only admin can modify rooms.

can_be_overridden(*user*)

can_be_prebooked(*user*, *ignore_admin=False*)

Reservable rooms can be pre-booked by anyone. Other rooms - only by their responsibles.

capacity

check_advance_days(*end_date*, *user=None*, *quiet=False*)

check_bookable_hours(*start_time*, *end_time*, *user=None*, *quiet=False*)

comments

details_url

division

static filter_available(*start_dt*, *end_dt*, *repetition*, *include_pre_bookings=True*, *include_pending_blockings=True*)

Returns a SQLAlchemy filter criterion ensuring that the room is available during the given time.

classmethod find_all(**args*, ***kwargs*)
Retrieves rooms, sorted by location and full name

find_available_vc_equipment()

classmethod find_with_attribute(*attribute*)
Search rooms which have a specific attribute

static find_with_filters(*filters*, *user=None*)

floor

full_name

generate_name()

get_attribute_by_name(*attribute_name*)

get_attribute_value(**args*, ***kwargs*)

get_blocked_rooms(**dates*, ***kwargs*)

classmethod get_owned_by(*user*)

static get_with_data(**args*, ***kwargs*)

has_attribute(*attribute_name*)

has_booking_groups

has_equipment(**args*, ***kwargs*)

has_live_reservations()

has_photo

has_projector

has_special_name

has_vc

has_webcast_recording

id

is_active

is_auto_confirm

is_owned_by(**args*, ***kwargs*)
Checks if the user is managing the room (owner or manager)

is_public

is_reservable

key_location

kind

large_photo_url

latitude

location_id

location_name

locator
Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

longitude

manager_emails

map_url

marker_description

max_advance_days

max_capacity
staticmethod(function) -> method

Convert a function to be a static method.

A static method does not receive an implicit first argument. To declare a static method, use this idiom:

class C: def f(arg1, arg2, ...): ... f = staticmethod(f)

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class.

Static methods in Python are similar to those found in Java or C++. For a more advanced concept, see the classmethod builtin.

name

nonbookable_periods

notification_before_days

notification_before_days_monthly

notification_before_days_weekly

notification_emails

notification_for_assistance

notifications_enabled

number

owner
The owner of the room. If the room has the *manager-group* attribute set, any users in that group are also considered owners when it comes to management privileges. Use *is_owned_by()* for ownership checks that should also check against the management group.

owner_id

photo

photo_id

reservations

reservations_need_confirmation

set_attribute_value(*name*, *value*)

site

small_photo_url

surface_area

telephone

update_name()

classmethod user_owns_rooms(*user*)

class indico.modules.rb.models.room_attributes.**RoomAttribute**(***kwargs*)
Bases: flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

children

id

is_hidden

is_required

location_id

name

parent_id

title

type

class indico.modules.rb.models.room_attributes.**RoomAttributeAssociation**(***kwargs*)
Bases: flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

attribute

attribute_id

room_id

value

class indico.modules.rb.models.room_bookable_hours.**BookableHours**(***kwargs*)
Bases: flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

end_time

fits_period(*st*, *et*)

room_id

start_time

class indico.modules.rb.models.room_nonbookable_periods.**NonBookablePeriod**(***kwargs*)
Bases: flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

end_dt

overlaps(*st*, *et*)

room_id

start_dt

class indico.modules.rb.models.aspects.**Aspect**(***kwargs*)
Bases: flask_sqlalchemy.Model, indico.util.serializer.Serializer

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

bottom_right_latitude

bottom_right_longitude

center_latitude

center_longitude

default_on_startup

id

location_id

name

top_left_latitude

top_left_longitude

zoom_level

class indico.modules.rb.models.blockings.**Blocking**(***kwargs*)
Bases: flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

allowed
A descriptor that presents a read/write view of an object attribute.

blocked_rooms

can_be_deleted(*user*)

can_be_modified(*user*)
The following persons are authorized to modify a blocking: - owner (the one who created the blocking) - admin (of course)

can_be_overridden(*user*, *room=None*, *explicit_only=False*)
Determines if a user can override the blocking

The following persons are authorized to override a blocking: - owner (the one who created the blocking) - any users on the blocking's ACL - unless explicitOnly is set: admins and room owners (if a room is given)

created_by_id

created_by_user
The user who created this blocking.

created_dt

end_date

id

is_active_at(*d*)

reason

start_date

class indico.modules.rb.models.blocked_rooms.**BlockedRoom**(***kwargs*)
Bases: flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

State
alias of *BlockedRoomState*

approve(*notify_blocker=True*)
Approve the room blocking, rejecting all colliding reservations/occurrences.

blocking_id

classmethod find_with_filters(*filters*)

id

reject(*user=None*, *reason=None*)
Reject the room blocking.

rejected_by

rejection_reason

room_id

state

state_name

class indico.modules.rb.models.blocked_rooms.**BlockedRoomState**
Bases: indico.util.struct.enum.RichIntEnum

accepted = 1

pending = 0

rejected = 2

class indico.modules.rb.models.blocking_principals.**BlockingPrincipal**(***kwargs*)
Bases: indico.core.db.sqlalchemy.principals.PrincipalMixin, flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

blocking_id

email = None

id

ip_network_group = None

ip_network_group_id = None

local_group

local_group_id

multipass_group_name

multipass_group_provider

principal_backref_name = u'in_blocking_acls'

type

unique_columns = (u'blocking_id',)

user

user_id

class indico.modules.rb.models.equipment.**EquipmentType**(***kwargs*)
Bases: flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

children

id

location_id

name

parent_id

class indico.modules.rb.models.holidays.**Holiday**(***kwargs*)
Bases: flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

date

id

location_id

name

class indico.modules.rb.models.locations.**Location**(***kwargs*)
Bases: flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

aspects

attributes

default_aspect

default_aspect_id

default_location
classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: def f(cls, arg1, arg2, ...): ... f = classmethod(f)

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

equipment_types

get_attribute_by_name(*name*)

get_buildings()

get_equipment_by_name(*name*)

holidays

id

is_default

is_map_available

locator

Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

map_url_template

name

rooms

set_default()

working_time_end = datetime.time(17, 30)

working_time_periods = ((datetime.time(8, 30), datetime.time(12, 30)), (datetime.time(13, 30), datetime.time(17, 30))

working_time_start = datetime.time(8, 30)

class indico.modules.rb.models.photos.**Photo**(***kwargs*)

Bases: flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

data

id

thumbnail

exception indico.modules.rb.models.reservations.**ConflictingOccurrences**
Bases: exceptions.Exception

class indico.modules.rb.models.reservations.**RepeatFrequency**
Bases: int, indico.util.struct.enum.IndicoEnum

DAY = 1

MONTH = 3

NEVER = 0

WEEK = 2

class indico.modules.rb.models.reservations.**RepeatMapping**
Bases: object

classmethod convert_legacy_repeatability(**args*, ***kw*)

classmethod get_message(**args*, ***kw*)

classmethod get_short_name(**args*, ***kw*)

mapping = {(<RepeatFrequency.NEVER: 0>, 0): ('Single reservation', None, 'none'), (<RepeatFrequency.MONTH: 3>, 1

class indico.modules.rb.models.reservations.**Reservation**(***kwargs*)
Bases: indico.util.serializer.Serializer, flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

accept(**args*, ***kwargs*)

add_edit_log(*edit_log*)

booked_for_id

booked_for_name

booked_for_user
The user this booking was made for. Assigning a user here also updates *booked_for_name*.

booking_reason

can_be_accepted(**args*, ***kwargs*)

can_be_cancelled(**args*, ***kwargs*)

can_be_deleted(**args*, ***kwargs*)

can_be_modified(**args*, ***kwargs*)

can_be_rejected(**args*, ***kwargs*)

cancel(**args*, ***kwargs*)

contact_email

contact_phone

classmethod create_from_data(*room*, *data*, *user*, *prebook=None*)
Creates a new reservation.

Parameters

- **room** – The Room that's being booked.
- **data** – A dict containing the booking data, usually from a `NewBookingConfirmForm` instance
- **user** – The *User* who creates the booking.
- **prebook** – Instead of determining the booking type from the user's permissions, always use the given mode.

create_occurrences(*skip_conflicts*, *user=None*)

created_by_id

created_by_user
The user who created this booking.

created_dt

details_url

edit_logs

end_dt

event
The Event this reservation was made for

event_id

find_excluded_days()

find_overlapping()

static **find_overlapping_with**(*room*, *occurrences*, *skip_reservation_id=None*)

get_conflicting_occurrences()

get_vc_equipment()

static **get_with_data**(**args*, ***kwargs*)

id

is_accepted

is_archived

is_booked_for(*user*)

is_cancelled

is_owned_by(**args*, ***kwargs*)

is_pending

is_rejected

is_repeating

is_valid

location_name

locator
Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

modify(*data*, *user*)
Modifies an existing reservation.

Parameters

- **data** – A dict containing the booking data, usually from a `ModifyBookingForm` instance
- **user** – The *User* who modifies the booking.

needs_assistance

needs_vc_assistance

occurrences

reject(**args*, ***kwargs*)

rejection_reason

repeat_frequency

repeat_interval

repetition

room_id

start_dt

status_string

used_equipment

uses_vc

class indico.modules.rb.models.reservation_edit_logs.**ReservationEditLog**(***kwargs*)
Bases: `flask_sqlalchemy.Model`

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

id

info

reservation_id

timestamp

user_name

class indico.modules.rb.models.reservation_occurrences.**ReservationOccurrence**(***kwargs*)
Bases: flask_sqlalchemy.Model, indico.util.serializer.Serializer

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

NO_RESERVATION_USER_STRATEGY = <sqlalchemy.orm.strategy_options._UnboundLoad object>
A relationship loading strategy that will avoid loading the users linked to a reservation. You want to use this in pretty much all cases where you eager-load the *reservation* relationship.

cancel(**args*, ***kwargs*)

classmethod create_series(*start*, *end*, *repetition*)

classmethod create_series_for_reservation(*reservation*)

date

end_dt

static filter_overlap(*occurrences*)

classmethod find_overlapping_with(*room*, *occurrences*, *skip_reservation_id=None*)

classmethod find_with_filters(*filters*, *user=None*)

get_overlap(*occurrence*, *skip_self=False*)

is_cancelled

is_rejected

is_valid

classmethod iter_create_occurrences(*start*, *end*, *repetition*)

static iter_start_time(*start*, *end*, *repetition*)

notification_sent

overlaps(*occurrence*, *skip_self=False*)

reject(**args*, ***kwargs*)

rejection_reason

reservation_id

start_dt

indico.modules.rb.models.util.**proxy_to_reservation_if_last_valid_occurrence**(*f*)
Forwards a method call to *self.reservation* if there is only one occurrence.

indico.modules.rb.models.util.**unimplemented**(*exceptions=(<type 'exceptions.Exception'>,)*, *message='Unimplemented'*)

Utilities

indico.modules.rb.util.**get_default_booking_interval**(*duration=90*, *precision=15*, *force_today=False*)

Get the default booking interval for a room.

Returns the default booking interval for a room as a tuple containing the start and end times as *datetime* objects.

The start time is the default working start time or the current time (if the working start time is in the past); rounded up to the given precision in minutes (15 by default).

The end time corresponds to the start time plus the given duration in minutes. If the booking ends after the end of work time, it is automatically moved to the next day.

Parameters

- **duration** – int – The duration of a booking in minutes (must be greater than 1)
- **precision** – int – The number of minutes by which to round up the current time for the start time of a booking. Negative values are allowed but will round the time down and create a booking starting in the past.
- **force_today** – Forces a booking to be for today, even if it past the end of work time. This is ignored if the current time is either after 23:50 or within the amount of minutes of the precision from midnight. For example with a precision of 30 minutes, if the current time is 23:42 then the meeting will be the following day.

Returns (datetime, datetime, bool) – A tuple with the start and end times of the booking and a boolean which is *True* if the date was changed from today and *False* otherwise.

Raises ValueError if the duration is less than 1 minute

indico.modules.rb.util.**rb_check_user_access**(**args*, ***kwargs*)

Checks if the user has access to the room booking system

indico.modules.rb.util.**rb_is_admin**(**args*, ***kwargs*)

Checks if the user is a room booking admin

indico.modules.rb.statistics.**calculate_rooms_bookable_time**(*rooms*, *start_date=None*, *end_date=None*)

indico.modules.rb.statistics.**calculate_rooms_booked_time**(*rooms*, *start_date=None*, *end_date=None*)

indico.modules.rb.statistics.**calculate_rooms_occupancy**(*rooms*, *start=None*, *end=None*)

indico.modules.rb.statistics.**compose_rooms_stats**(*rooms*)

Services

class indico.modules.rb.services.rooms.**BookingPermission**(*params*)

Bases: indico.legacy.services.implementation.base.LoggedOnlyService

UNICODE_PARAMS = True

class indico.modules.rb.services.rooms.**RoomBookingAvailabilitySearchRooms**(*params*)

Bases: indico.legacy.services.implementation.base.ServiceBase

UNICODE_PARAMS = True

class indico.modules.rb.services.rooms.**RoomBookingListLocationsAndRoomsWithGuids**(*params*)

Bases: indico.legacy.services.implementation.base.ServiceBase

UNICODE_PARAMS = True

class indico.modules.rb.services.aspects.**RoomBookingMapBase**(*params*)
Bases: indico.legacy.services.implementation.base.ServiceBase

UNICODE_PARAMS = True

class indico.modules.rb.services.aspects.**RoomBookingMapCreateAspect**(*params*)
Bases: *indico.modules.rb.services.aspects.RoomBookingMapBase*

class indico.modules.rb.services.aspects.**RoomBookingMapListAspects**(*params*)
Bases: *indico.modules.rb.services.aspects.RoomBookingMapBase*

class indico.modules.rb.services.aspects.**RoomBookingMapRemoveAspect**(*params*)
Bases: *indico.modules.rb.services.aspects.RoomBookingMapBase*

class indico.modules.rb.services.aspects.**RoomBookingMapUpdateAspect**(*params*)
Bases: *indico.modules.rb.services.aspects.RoomBookingMapBase*

class indico.modules.rb.services.blockings.**RoomBookingBlockingApprove**(*params*)
Bases: *indico.modules.rb.services.blockings.RoomBookingBlockingProcessBase*

class indico.modules.rb.services.blockings.**RoomBookingBlockingProcessBase**(*params*)
Bases: indico.legacy.services.implementation.base.ServiceBase

UNICODE_PARAMS = True

class indico.modules.rb.services.blockings.**RoomBookingBlockingReject**(*params*)
Bases: *indico.modules.rb.services.blockings.RoomBookingBlockingProcessBase*

Authentication

Todo

Docstrings (module, models, utilities)

Models

class indico.modules.auth.models.identities.**Identity**(***kwargs*)
Bases: flask_sqlalchemy.Model

Identities of Indico users

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

data

id
the unique id of the identity

identifier
the unique identifier of the user within its provider

last_login_dt
the timestamp of the latest login

last_login_ip
the ip address that was used for the latest login

locator

multipass_data
internal data used by the flask-multipass system

password
the password of the user in case of a local identity

password_hash
the hash of the password in case of a local identity

provider
the provider name of the identity

register_login(*ip*)
Updates the last login information

safe_last_login_dt
last_login_dt that is safe for sorting (no None values)

user_id
the id of the user this identity belongs to

class indico.modules.auth.models.registration_requests.**RegistrationRequest**(***kwargs*)
Bases: flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

comment

email

extra_emails

id

identity_data

locator
Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

settings

user_data

Utilities

indico.modules.auth.util.**impersonate_user**(*user*)
Impersonate another user as an admin

indico.modules.auth.util.**load_identity_info**()
Retrieves identity information from the session

indico.modules.auth.util.**redirect_to_login**(*next_url=None*, *reason=None*)
Redirects to the login page.

Parameters

- **next_url** – URL to be redirected upon successful login. If not specified, it will be set to `request.relative_url`.
- **reason** – Why the user is redirected to a login page.

indico.modules.auth.util.**register_user**(*email*, *extra_emails*, *user_data*, *identity_data*, *settings*, *from_moderation=False*)
Create a user based on the registration data provided during te user registration process (via *RHRegister* and *RegistrationHandler*).

This method is not meant to be used for generic user creation, the only reason why this is here is that approving a registration request is handled by the *users* module.

indico.modules.auth.util.**save_identity_info**(*identity_info*, *user*)
Saves information from IdentityInfo in the session

indico.modules.auth.util.**undo_impersonate_user**()
Undo an admin impersonation login and revert to the old user

indico.modules.auth.util.**url_for_login**(*next_url=None*)

indico.modules.auth.util.**url_for_logout**(*next_url=None*)

indico.modules.auth.util.**url_for_register**(*next_url=None*, *email=None*)
Returns the URL to register

Parameters

- **next_url** – The URL to redirect to afterwards.
- **email** – A pre-validated email address to use when creating a new local account. Use this argument ONLY when sending the link in an email or if the email address has already been validated using some other way.

OAuth

Todo

Docstrings (module, models, provider)

Models

class indico.modules.oauth.models.applications.**OAuthApplication**(***kwargs*)
Bases: flask_sqlalchemy.Model

OAuth applications registered in Indico

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

client_id
the OAuth client_id

client_secret
the OAuth client_secret

client_type

default_redirect_uri

default_scopes
the OAuth default scopes the application may request access to

description
human readable description

id
the unique id of the application

is_enabled
whether the application is enabled or disabled

is_trusted
whether the application can access user data without asking for permission

locator

name
human readable name

redirect_uris
the OAuth absolute URIs that a application may use to redirect to after authorization

reset_client_secret()

system_app_type
the type of system app (if any). system apps cannot be deleted

validate_redirect_uri(*redirect_uri*)
Called by flask-oauthlib to validate the redirect_uri.

Uses a logic similar to the one at GitHub, i.e. protocol and host/port must match exactly and if there is a path in the whitelisted URL, the path of the redirect_uri must start with that path.

class indico.modules.oauth.models.applications.**SystemAppType**
Bases: int, indico.util.struct.enum.IndicoEnum

checkin = 1

default_data

enforced_data

flower = 2

none = 0

class indico.modules.oauth.models.tokens.**OAuthGrant**(*client_id*, *code*, *redirect_uri*, *user*, *scopes*, *expires*)

Bases: object

OAuth grant token

delete()

classmethod get(*client_id*, *code*)

key

classmethod make_key(*client_id*, *code*)

save()

ttl

class indico.modules.oauth.models.tokens.**OAuthToken**(***kwargs*)

Bases: flask_sqlalchemy.Model

OAuth tokens

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

access_token
an unguessable unique string of characters

application
application authorized by this token

application_id
the identifier of the linked application

expires

id
the unique identifier of the token

last_used_dt
the last time the token was used by the application

locator

scopes
The set of scopes the linked application has access to.

type

user
the user who owns this token

user_id
the identifier of the linked user

Utilities

exception indico.modules.oauth.provider.**DisabledClientIdError**(*description=None*, *uri=None*, *state=None*, *status_code=None*, *request=None*)

Bases: oauthlib.oauth2.rfc6749.errors.FatalClientError

description: A human-readable ASCII [USASCII] text providing additional information, used to assist the client developer in understanding the error that occurred. Values for the "error_description" parameter MUST NOT include characters outside the set x20-21 / x23-5B / x5D-7E.

uri: A URI identifying a human-readable web page with information about the error, used to provide the client developer with additional information about the error. Values for the "error_uri" parameter MUST conform to the URI- Reference syntax, and thus MUST NOT include characters outside the set x21 / x23-5B / x5D-7E.

state: A CSRF protection value received from the client.

request: Oauthlib Request object

error = u'application_disabled_by_admin'

indico.modules.oauth.provider.**load_client**(*client_id*)

indico.modules.oauth.provider.**load_grant**(*client_id*, *code*)

indico.modules.oauth.provider.**load_token**(*access_token*, *refresh_token=None*)

indico.modules.oauth.provider.**save_grant**(*client_id*, *code*, *request*, **args*, ***kwargs*)

indico.modules.oauth.provider.**save_token**(*token_data*, *request*, **args*, ***kwargs*)

Group

Todo

Docstrings (module)

Models

class indico.modules.groups.models.groups.**LocalGroup**(***kwargs*)

Bases: flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

id
the unique id of the group

members
the local groups this user belongs to

name
the name of the group

proxy
Returns a GroupProxy wrapping this group

class indico.modules.groups.core.**GroupProxy**
Bases: object

Provides a generic interface for both local and multipass groups.

Creating an instance of this class actually creates either a LocalGroupProxy or a MultipassGroupProxy, but they expose the same API.

Parameters

- **name_or_id** – The name of a multipass group or ID of a local group
- **provider** – The provider of a multipass group

Creates the correct GroupProxy for the group type

as_legacy

as_legacy_group
The legacy-style group wrapper

as_principal
The serializable principal identifier of this group

get_members()
Gets the list of users who are members of the group

classmethod get_named_default_group(*name*)
Gets the group with the matching name from the default group provider.

If there is no default group provider, local groups will be used and *name* is the group's ID.

This method should only be used for legacy code or code that gets the group name from an external source which does not contain a provider identifier.

group
The underlying group object

has_member(*user*)
Checks if the user is a member of the group.

This can also be accessed using the in operator.

is_group = True

is_network = False

is_single_person = False

principal_order = 2

classmethod search(*name*, *exact=False*, *providers=None*)
Searches for groups

Parameters

- **name** – The group name to search for.
- **exact** – If only exact matches should be found (much faster)

- **providers** – None to search in all providers and local groups. May be a set specifying providers to search in. For local groups, the 'indico' provider name may be used.

Utilities

indico.modules.groups.util.**serialize_group**(*group*)
Serialize group to JSON-like object

Video conference

Todo

Docstrings (module, models, utilities, plugins, exceptions)

Models

class indico.modules.vc.models.vc_rooms.**VCRoom**(***kwargs*)
Bases: flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

created_by_id
ID of the creator

created_by_user
The user who created the videoconference room

created_dt
Creation timestamp of the videoconference room

data
videoconference plugin-specific data

id
Videoconference room ID

locator

modified_dt
Modification timestamp of the videoconference room

name
Name of the videoconference room

plugin

status
Status of the videoconference room

type
Type of the videoconference room

class indico.modules.vc.models.vc_rooms.**VCRoomEventAssociation**(***kwargs*)
Bases: flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

contribution_id

data
videoconference plugin-specific data

delete(*user*, *delete_all=False*)
Deletes a VC room from an event

If the room is not used anywhere else, the room itself is also deleted.

Parameters

- **user** – the user performing the deletion
- **delete_all** – if True, the room is detached from all events and deleted.

event
The associated Event

event_id
ID of the event

classmethod find_for_event(**args*, ***kwargs*)
Returns a Query that retrieves the videoconference rooms for an event

Parameters

- **event** – an indico Event
- **only_linked_to_event** – only retrieve the vc rooms linked to the whole event
- **kwargs** – extra kwargs to pass to `find()`

classmethod get_linked_for_event(**args*, ***kwargs*)
Get a dict mapping link objects to event vc rooms

id
Association ID

link_object

link_type
Type of the object the vc_room is linked to

linked_block
The linked session block (if the VC room is attached to a block)

linked_contrib
The linked contribution (if the VC room is attached to a contribution)

linked_event
The linked event (if the VC room is attached to the event itself)

linked_event_id

locator

classmethod register_link_events()

session_block_id

show
If the vc room should be shown on the event page

vc_room
The associated :class:VCRoom

vc_room_id
ID of the videoconference room

class indico.modules.vc.models.vc_rooms.**VCRoomLinkType**
Bases: int, indico.util.struct.enum.IndicoEnum

block = 3

contribution = 2

event = 1

class indico.modules.vc.models.vc_rooms.**VCRoomStatus**
Bases: int, indico.util.struct.enum.IndicoEnum

created = 1

deleted = 2

Utilities

indico.modules.vc.util.**find_event_vc_rooms**(*from_dt=None*, *to_dt=None*, *distinct=False*)
Finds VC rooms matching certain criteria

Parameters

- **from_dt** – earliest event/contribution to include
- **to_dt** – latest event/contribution to include
- **distinct** – if True, never return the same (event, vcroom) more than once (even if it's linked more than once to that event)

indico.modules.vc.util.**get_linked_to_description**(*obj*)

indico.modules.vc.util.**get_managed_vc_plugins**(*user*)
Returns the plugins the user can manage

indico.modules.vc.util.**get_vc_plugins**()
Returns a dict containing the available videoconference plugins.

indico.modules.vc.util.**resolve_title**(*obj*)

Plugins

class indico.modules.vc.plugins.**VCPluginMixin**
Bases: object

acl_settings = set([u'managers', u'acl'])

can_manage_vc(*user*)
Checks if a user has management rights on this VC system

can_manage_vc_room(*user*, *room*)
Checks if a user can manage a vc room

can_manage_vc_rooms(*user*, *event*)
Checks if a user can manage vc rooms on an event

category = u'Videoconference'

create_form(*event*, *existing_vc_room=None*, *existing_event_vc_room=None*)
Creates the videoconference room form

Parameters

- **event** – the event the videoconference room is for
- **existing_vc_room** – a vc_room from which to retrieve data for the form
- ****kwargs** – extra data to pass to the form if an existing vc room is passed

Returns an instance of an `IndicoForm` subclass

create_room(*vc_room*, *event*)

default_settings = {u'notification_emails': []}

friendly_name = None
the readable name of the VC plugin

get_notification_bcc_list(*action*, *vc_room*, *event*)

get_notification_cc_list(*action*, *vc_room*, *event*)

get_vc_room_attach_form_defaults(*event*)

get_vc_room_form_defaults(*event*)

icon_url

init()

logo_url

render_buttons(*vc_room*, *event_vc_room*, ***kwargs*)
Renders a list of plugin specific buttons (eg: Join URL, etc) in the management area

Parameters

- **vc_room** – the VC room object
- **event_vc_room** – the association of an event and a VC room
- **kwargs** – arguments passed to the template

render_event_buttons(*vc_room*, *event_vc_room*, ***kwargs*)
Renders a list of plugin specific buttons (eg: Join URL, etc) in the event page

Parameters

- **vc_room** – the VC room object
- **event_vc_room** – the association of an event and a VC room
- **kwargs** – arguments passed to the template

render_form(***kwargs*)
Renders the videoconference room form :param kwargs: arguments passed to the template

render_info_box(*vc_room*, *event_vc_room*, *event*, ***kwargs*)
Renders the information shown in the expandable box of a VC room row :param vc_room: the VC room object :param event_vc_room: the association of an event and a VC room :param event: the event with the current VC room attached to it :param kwargs: arguments passed to the template

render_manage_event_info_box(*vc_room*, *event_vc_room*, *event*, ***kwargs*)
Renders the information shown in the expandable box on a VC room in the management area

Parameters

- **vc_room** – the VC room object
- **event_vc_room** – the association of an event and a VC room
- **event** – the event with the current VC room attached to it
- **kwargs** – arguments passed to the template

service_name

settings_form
alias of `VCPluginSettingsFormBase`

update_data_association(*event*, *vc_room*, *event_vc_room*, *data*)

update_data_vc_room(*vc_room*, *data*)

vc_room_attach_form = None
the `IndicoForm` to use for the videoconference room attach form

vc_room_form = None
the `IndicoForm` to use for the videoconference room form

Exceptions

exception indico.modules.vc.exceptions.**VCRoomError**(*message*, *field=None*)
Bases: exceptions.Exception

exception indico.modules.vc.exceptions.**VCRoomNotFoundError**(*message*)
Bases: *indico.modules.vc.exceptions.VCRoomError*

Designer

Todo

Docstrings (module, models, utilities)

Models

class indico.modules.designer.models.images.**DesignerImageFile**(***kwargs*)
Bases: indico.core.storage.models.StoredFileMixin, flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

content_type
The MIME type of the file

created_dt
The date/time when the file was uploaded

download_url

filename
The name of the file

id
The ID of the file

locator

md5
An MD5 hash of the file.

Automatically assigned when *save()* is called.

size
The size of the file (in bytes).

Automatically assigned when *save()* is called.

storage_backend

storage_file_id

template

template_id
The designer template the image belongs to

version_of = None

class indico.modules.designer.models.templates.**DesignerTemplate**(***kwargs*)
Bases: flask_sqlalchemy.Model

background_image

background_image_id

backside_template

backside_template_id

category

category_id

data

event

event_id

id

is_clonable

is_system_template

locator
Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

owner

title

type

Utilities

indico.modules.designer.util.**get_all_templates**(*obj*)
Get all templates usable by an event/category

indico.modules.designer.util.**get_default_template_on_category**(*category*, *only_inherited=False*)

indico.modules.designer.util.**get_inherited_templates**(*obj*)
Get all templates inherited by a given event/category

indico.modules.designer.util.**get_not_deletable_templates**(*obj*)
Get all non-deletable templates for an event/category

indico.modules.designer.util.**get_placeholder_options**()

class indico.modules.designer.pdf.**DesignerPDFBase**(*template*, *config*)
Bases: `object`

get_pdf()

class indico.modules.designer.pdf.**TplData**(*width*, *height*, *items*, *background_position*, *width_cm*, *height_cm*)
Bases: `tuple`

Create new instance of TplData(width, height, items, background_position, width_cm, height_cm)

background_position
Alias for field number 3

height
Alias for field number 1

height_cm
Alias for field number 5

items
Alias for field number 2

width
Alias for field number 0

width_cm
Alias for field number 4

Placeholders

class indico.modules.designer.placeholders.**EventDatesPlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'Event Dates'

group = u'event'

name = u'event_dates'

classmethod render(*event*)

class indico.modules.designer.placeholders.**EventDescriptionPlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'Event Description'

group = u'event'

name = u'event_description'

classmethod render(*event*)

class indico.modules.designer.placeholders.**RegistrationFullNamePlaceholder**
Bases: indico.modules.designer.placeholders.FullNamePlaceholderBase

description = lu'Full Name'

name = u'full_name'

name_options = {}

with_title = True

class indico.modules.designer.placeholders.**EventOrgTextPlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'Event Organizers'

group = u'event'

name = u'event_organizers'

classmethod render(*event*)

class indico.modules.designer.placeholders.**RegistrationFullNameNoTitlePlaceholder**
Bases: indico.modules.designer.placeholders.FullNamePlaceholderBase

description = lu'Full Name (no title)'

name = u'full_name_no_title'

name_options = {}

with_title = False

class indico.modules.designer.placeholders.**RegistrationFullNamePlaceholderB**
Bases: indico.modules.designer.placeholders.FullNamePlaceholderBase

description = lu'Full Name B'

name = u'full_name_b'

name_options = {u'last_name_first': False}

with_title = True

class indico.modules.designer.placeholders.**RegistrationFullNameNoTitlePlaceholderB**
Bases: indico.modules.designer.placeholders.FullNamePlaceholderBase

description = lu'Full Name B (no title)'

name = u'full_name_b_no_title'

name_options = {u'last_name_first': False}

with_title = False

class indico.modules.designer.placeholders.**RegistrationFullNamePlaceholderC**
Bases: indico.modules.designer.placeholders.FullNamePlaceholderBase

description = lu'Full Name C'

name = u'full_name_c'

name_options = {u'last_name_upper': True, u'last_name_first': False}

with_title = True

class indico.modules.designer.placeholders.**RegistrationFullNameNoTitlePlaceholderC**
Bases: indico.modules.designer.placeholders.FullNamePlaceholderBase

description = lu'Full Name C (no title)'

name = u'full_name_no_title_c'

name_options = {u'last_name_upper': True}

with_title = False

class indico.modules.designer.placeholders.**RegistrationFullNamePlaceholderD**
Bases: indico.modules.designer.placeholders.FullNamePlaceholderBase

description = lu'Full Name D (abbrev.)'

name = u'full_name_d'

name_options = {u'last_name_upper': True, u'last_name_first': False, u'abbrev_first_name': True}

with_title = True

class indico.modules.designer.placeholders.**RegistrationFullNameNoTitlePlaceholderD**
Bases: indico.modules.designer.placeholders.FullNamePlaceholderBase

description = lu'Full Name D (abbrev., no title)'

name = u'full_name_no_title_d'

name_options = {u'last_name_upper': True, u'abbrev_first_name': True}

with_title = False

class indico.modules.designer.placeholders.**RegistrationTitlePlaceholder**
Bases: indico.modules.designer.placeholders.RegistrationPDPlaceholder

description = lu'Title'

field = u'title'

name = u'title'

class indico.modules.designer.placeholders.**RegistrationFirstNamePlaceholder**
Bases: indico.modules.designer.placeholders.RegistrationPlaceholder

description = lu'First Name'

field = u'first_name'

name = u'first_name'

class indico.modules.designer.placeholders.**RegistrationLastNamePlaceholder**
Bases: indico.modules.designer.placeholders.RegistrationPlaceholder

description = lu'Last Name'

field = u'last_name'

name = u'last_name'

class indico.modules.designer.placeholders.**RegistrationTicketQRPlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'Ticket QR Code'

group = u'registrant'

name = u'ticket_qr_code'

classmethod render(*registration*)

class indico.modules.designer.placeholders.**RegistrationEmailPlaceholder**
Bases: indico.modules.designer.placeholders.RegistrationPlaceholder

description = lu'E-mail'

field = u'email'

name = u'email'

class indico.modules.designer.placeholders.**RegistrationAmountPlaceholder**
Bases: indico.modules.designer.placeholders.RegistrationPlaceholder

description = lu'Price (no currency)'

name = u'amount'

classmethod render(*registration*)

class indico.modules.designer.placeholders.**RegistrationPricePlaceholder**
Bases: indico.modules.designer.placeholders.RegistrationPlaceholder

description = lu'Price (with currency)'

name = u'price'

classmethod render(*registration*)

class indico.modules.designer.placeholders.**RegistrationAffiliationPlaceholder**
Bases: indico.modules.designer.placeholders.RegistrationPDPlaceholder

description = lu'Institution'

field = u'affiliation'

name = u'affiliation'

class indico.modules.designer.placeholders.**RegistrationPositionPlaceholder**
Bases: indico.modules.designer.placeholders.RegistrationPDPlaceholder

description = lu'Position'

field = u'position'

name = u'position'

class indico.modules.designer.placeholders.**RegistrationAddressPlaceholder**
Bases: indico.modules.designer.placeholders.RegistrationPDPlaceholder

description = lu'Address'

field = u'address'

name = u'address'

class indico.modules.designer.placeholders.**RegistrationCountryPlaceholder**
Bases: indico.modules.designer.placeholders.RegistrationPDPlaceholder

description = lu'Country'

field = u'country'

name = u'country'

class indico.modules.designer.placeholders.**RegistrationPhonePlaceholder**
Bases: indico.modules.designer.placeholders.RegistrationPDPlaceholder

description = lu'Phone'

field = u'phone'

name = u'phone'

class indico.modules.designer.placeholders.**EventTitlePlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'Event Title'

group = u'event'

name = u'event_title'

classmethod render(*event*)

class indico.modules.designer.placeholders.**CategoryTitlePlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'Category Title'

group = u'event'

name = u'category_title'

classmethod render(*event*)

class indico.modules.designer.placeholders.**EventRoomPlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'Event Room'

group = u'event'

name = u'event_room'

classmethod render(*event*)

class indico.modules.designer.placeholders.**EventVenuePlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'Event Venue'

group = u'event'

name = u'event_venue'

classmethod render(*event*)

class indico.modules.designer.placeholders.**EventSpeakersPlaceholder**
Bases: indico.util.placeholders.Placeholder

description = lu'Event Speakers/Chairs'

group = u'event'

name = u'event_speakers'

classmethod render(*event*)

Network

Todo

Docstrings (module, models)

Models

class indico.modules.networks.models.networks.**IPNetwork**(***kwargs*)
Bases: flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

group_id

network

class indico.modules.networks.models.networks.**IPNetworkGroup**(***kwargs*)
Bases: flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

attachment_access_override
Grants all IPs in the network group read access to all attachments

contains_ip(*ip*)

description

hidden
Whether the network group is hidden in ACL forms

id

is_group = False

is_network = True

is_single_person = False

locator

name

networks
A descriptor that presents a read/write view of an object attribute.

principal_order = 1

principal_type = 5

Utilities

indico.modules.networks.util.**serialize_ip_network_group**(*group*)
Serialize group to JSON-like object

News

Todo

Docstrings (module, models)

Models

class indico.modules.news.models.news.**NewsItem**(**kwargs*)
Bases: flask_sqlalchemy.Model

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

anchor

content

created_dt

id

locator
Defines a smart locator property.

This behaves pretty much like a normal read-only property and the decorated function should return a dict containing the necessary data to build a URL for the object.

This decorator should usually be applied to a method named `locator` as this name is required for *get_locator* to find it automatically when just passing the object.

If you need more than one locator, you can define it like this:

```
@locator_property
def locator(self):
    return {...}

@locator.other
def locator(self):
    return {...}
```

The `other` locator can then be accessed by passing `obj.locator.other` to the code expecting an object with a locator.

title

Utilities

indico.modules.news.util.**get_recent_news**(**args*, ***kwargs*)
Get a list of recent news for the home page

Indico fields

Todo

Docstrings to all fields

Indico fields extend from WTForm fields and are used for the special cases where the simple form fields are not enough to cover all needs.

class indico.modules.events.fields.**EventPersonLinkListField**(**args*, ***kwargs*)
Bases: *indico.modules.events.fields.PersonLinkListFieldBase*

A field to manage event's chairpersons

linked_object_attr = u'event'

person_link_cls
alias of EventPersonLink

pre_validate(*form*)

widget = <indico.web.forms.widgets.JinjaWidget object>

class indico.modules.events.fields.**EventPersonListField**(**args*, ***kwargs*)
Bases: indico.web.forms.fields.principals.PrincipalListField

A field that lets you select a list Indico user and EventPersons

Requires its form to have an event set.

create_untrusted_persons = False
Whether new event persons created by the field should be marked as untrusted

event

pre_validate(*form*)

process_formdata(*valuelist*)

class indico.modules.events.fields.**IndicoThemeSelectField**(**args*, ***kwargs*)
Bases: wtforms.fields.core.SelectField

class indico.modules.events.fields.**PersonLinkListFieldBase**(**args*, ***kwargs*)
Bases: *indico.modules.events.fields.EventPersonListField*

default_sort_alpha = True
If set to *True*, will be sorted alphabetically by default

linked_object_attr = None
name of the attribute on the form containing the linked object

person_link_cls = None
class that inherits from *PersonLinkBase*

widget = None

class indico.modules.events.fields.**ReferencesField**(**args*, ***kwargs*)
Bases: indico.web.forms.fields.itemlists.MultipleItemsField

A field to manage external references.

pre_validate(*form*)

process_formdata(*valuelist*)

class indico.modules.events.fields.**ReviewQuestionsField**(**args*, ***kwargs*)
Bases: indico.web.forms.fields.itemlists.MultipleItemsField

process_formdata(*valuelist*)

class indico.modules.events.abstracts.fields.**AbstractField**(**args*, ***kwargs*)
Bases: wtforms.ext.sqlalchemy.fields.QuerySelectField

A selectize-based field to select an abstract from an event.

event

pre_validate(*form*)

search_payload

search_url

widget = <indico.web.forms.widgets.SelectizeWidget object>

class indico.modules.events.abstracts.fields.**AbstractPersonLinkListField**(**args*, ***kwargs*)
Bases: *indico.modules.events.fields.PersonLinkListFieldBase*

A field to configure a list of abstract persons

create_untrusted_persons = True

default_sort_alpha = False

linked_object_attr = u'abstract'

person_link_cls
alias of AbstractPersonLink

pre_validate(*form*)

widget = <indico.web.forms.widgets.JinjaWidget object>

class indico.modules.events.abstracts.fields.**EmailRuleListField**(*label=None*, *validators=None*, *filters=()*, *description=u''*, *id=None*, *default=None*, *widget=None*, *render_kw=None*, *_form=None*, *_name=None*, *_prefix=u''*, *_translations=None*, *_meta=None*)

Bases: indico.web.forms.fields.simple.JSONField

A field that stores a list of e-mail template rules.

Construct a new field.

Parameters

- **label** – The label of the field.
- **validators** – A sequence of validators to call when *validate* is called.
- **filters** – A sequence of filters which are run on input data by *process*.
- **description** – A description for the field, typically used for help text.
- **id** – An id to use for the field. A reasonable default is set by the form, and you shouldn't need to set this manually.
- **default** – The default value to assign to the field, if no form or object input is provided. May be a callable.
- **widget** – If provided, overrides the widget used to render the field.
- **render_kw** (*dict*) – If provided, a dictionary which provides default keywords that will be given to the widget at render time.
- **_form** – The form holding this field. It is passed by the form itself during construction. You should never pass this value yourself.
- **_name** – The name of this field, passed by the enclosing form during its construction. You should never pass this value yourself.
- **_prefix** – The prefix to prepend to the form name of this field, passed by the enclosing form during construction.
- **_translations** – A translations object providing message translations. Usually passed by the enclosing form during construction. See I18n docs for information on message translations.
- **_meta** – If provided, this is the 'meta' instance from the form. You usually don't pass this yourself.

If *_form* and *_name* isn't provided, an UnboundField will be returned instead. Call its bind() method with a form instance and a name to construct the field.

CAN_POPULATE = True

accepted_condition_types = (<class 'indico.modules.events.abstracts.notifications.StateCondition'>, <class 'indico

condition_choices

condition_class_map = {u'track': <class 'indico.modules.events.abstracts.notifications.TrackCondition'>, u'state': <

pre_validate(*form*)

widget = <indico.web.forms.widgets.JinjaWidget object>

class indico.modules.events.abstracts.fields.**TrackRoleField**(*label=None*, *validators=None*, *filters=()*, *description=u''*, *id=None*, *default=None*, *widget=None*, *render_kw=None*, *_form=None*, *_name=None*, *_prefix=u''*, *_translations=None*, *_meta=None*)

Bases: indico.web.forms.fields.simple.JSONField

A field that stores a list of e-mail template rules.

Construct a new field.

Parameters

- **label** – The label of the field.
- **validators** – A sequence of validators to call when *validate* is called.
- **filters** – A sequence of filters which are run on input data by *process*.
- **description** – A description for the field, typically used for help text.
- **id** – An id to use for the field. A reasonable default is set by the form, and you shouldn't need to set this manually.
- **default** – The default value to assign to the field, if no form or object input is provided. May be a callable.
- **widget** – If provided, overrides the widget used to render the field.
- **render_kw** (*dict*) – If provided, a dictionary which provides default keywords that will be given to the widget at render time.
- **_form** – The form holding this field. It is passed by the form itself during construction. You should never pass this value yourself.
- **_name** – The name of this field, passed by the enclosing form during its construction. You should never pass this value yourself.
- **_prefix** – The prefix to prepend to the form name of this field, passed by the enclosing form during construction.
- **_translations** – A translations object providing message translations. Usually passed by the enclosing form during construction. See I18n docs for information on message translations.
- **_meta** – If provided, this is the 'meta' instance from the form. You usually don't pass this yourself.

If *_form* and *_name* isn't provided, an UnboundField will be returned instead. Call its bind() method with a form instance and a name to construct the field.

CAN_POPULATE = True

role_data

users

widget = <indico.web.forms.widgets.JinjaWidget object>

class indico.modules.events.contributions.fields.**ContributionPersonLinkListField**(**args*, ***kwargs*)

Bases: *indico.modules.events.fields.PersonLinkListFieldBase*

A field to configure a list of contribution persons

linked_object_attr = u'contrib'

person_link_cls
alias of ContributionPersonLink

pre_validate(*form*)

widget = <indico.web.forms.widgets.JinjaWidget object>

class indico.modules.events.contributions.fields.**SubContributionPersonLinkListField**(**args*, ***kwargs*)

Bases: *indico.modules.events.contributions.fields.ContributionPersonLinkListField*

A field to configure a list of subcontribution persons

linked_object_attr = u'subcontrib'

person_link_cls
alias of SubContributionPersonLink

widget = <indico.web.forms.widgets.JinjaWidget object>

class indico.modules.events.papers.fields.**PaperEmailSettingsField**(*label=None*, *validators=None*, *filters=()*, *description=u''*, *id=None*, *default=None*, *widget=None*, *render_kw=None*, *_form=None*, *_name=None*, *_prefix=u''*, *_translations=None*, *_meta=None*)

Bases: indico.web.forms.fields.simple.JSONField

Construct a new field.

Parameters

- **label** – The label of the field.
- **validators** – A sequence of validators to call when *validate* is called.
- **filters** – A sequence of filters which are run on input data by *process*.
- **description** – A description for the field, typically used for help text.

- **id** – An id to use for the field. A reasonable default is set by the form, and you shouldn't need to set this manually.
- **default** – The default value to assign to the field, if no form or object input is provided. May be a callable.
- **widget** – If provided, overrides the widget used to render the field.
- **render_kw** (*dict*) – If provided, a dictionary which provides default keywords that will be given to the widget at render time.
- **_form** – The form holding this field. It is passed by the form itself during construction. You should never pass this value yourself.
- **_name** – The name of this field, passed by the enclosing form during its construction. You should never pass this value yourself.
- **_prefix** – The prefix to prepend to the form name of this field, passed by the enclosing form during construction.
- **_translations** – A translations object providing message translations. Usually passed by the enclosing form during construction. See I18n docs for information on message translations.
- **_meta** – If provided, this is the 'meta' instance from the form. You usually don't pass this yourself.

If *_form* and *_name* isn't provided, an `UnboundField` will be returned instead. Call its `bind()` method with a form instance and a name to construct the field.

CAN_POPULATE = True

event

process_formdata(*valuelist*)

widget = <indico.web.forms.widgets.JinjaWidget object>

class indico.modules.events.sessions.fields.**SessionBlockPersonLinkListField**(**args*, ***kwargs*)

Bases: *indico.modules.events.fields.PersonLinkListFieldBase*

linked_object_attr = u'session_block'

person_link_cls

alias of `SessionBlockPersonLink`

widget = <indico.web.forms.widgets.JinjaWidget object>

class indico.modules.categories.fields.**CategoryField**(**args*, ***kwargs*)

Bases: `wtforms.fields.simple.HiddenField`

WTForms field that lets you select a category.

Parameters

- **allow_events** – Whether to allow selecting a category that contains events.
- **allow_subcats** – Whether to allow selecting a category that contains subcategories.
- **require_event_creation_rights** – Whether to allow selecting only categories where the user can create events.

pre_validate(*form*)

process_data(*value*)

process_formdata(*valuelist*)

widget = <indico.web.forms.widgets.JinjaWidget object>

class indico.modules.networks.fields.**MultiIPNetworkField**(**args*, ***kwargs*)
Bases: indico.web.forms.fields.itemlists.MultiStringField

A field to enter multiple IPv4 or IPv6 networks.

The field data is a set of IPNetwork``s not bound to a DB session. The ``unique and sortable parameters of the parent class cannot be used with this class.

pre_validate(*form*)

process_data(*value*)

process_formdata(*valuelist*)

class indico.web.forms.fields.**IndicoSelectMultipleCheckboxField**(*label=None*, *validators=None*, *coerce=<type 'unicode'>*, *choices=None*, ***kwargs*)
Bases: wtforms.fields.core.SelectMultipleField

option_widget = <wtforms.widgets.core.CheckboxInput object>

widget = <indico.web.forms.widgets.JinjaWidget object>

class indico.web.forms.fields.**IndicoRadioField**(**args*, ***kwargs*)
Bases: wtforms.fields.core.RadioField

widget = <indico.web.forms.widgets.JinjaWidget object>

class indico.web.forms.fields.**JSONField**(*label=None*, *validators=None*, *filters=()*, *description=u''*, *id=None*, *default=None*, *widget=None*, *render_kw=None*, *_form=None*, *_name=None*, *_prefix=u''*, *_translations=None*, *_meta=None*)
Bases: wtforms.fields.simple.HiddenField

Construct a new field.

Parameters

- **label** – The label of the field.
- **validators** – A sequence of validators to call when *validate* is called.
- **filters** – A sequence of filters which are run on input data by *process*.
- **description** – A description for the field, typically used for help text.
- **id** – An id to use for the field. A reasonable default is set by the form, and you shouldn't need to set this manually.
- **default** – The default value to assign to the field, if no form or object input is provided. May be a callable.
- **widget** – If provided, overrides the widget used to render the field.
- **render_kw** (*dict*) – If provided, a dictionary which provides default keywords that will be given to the widget at render time.
- **_form** – The form holding this field. It is passed by the form itself during construction. You should never pass this value yourself.

- **_name** – The name of this field, passed by the enclosing form during its construction. You should never pass this value yourself.
- **_prefix** – The prefix to prepend to the form name of this field, passed by the enclosing form during construction.
- **_translations** – A translations object providing message translations. Usually passed by the enclosing form during construction. See I18n docs for information on message translations.
- **_meta** – If provided, this is the 'meta' instance from the form. You usually don't pass this yourself.

If *_form* and *_name* isn't provided, an `UnboundField` will be returned instead. Call its `bind()` method with a form instance and a name to construct the field.

CAN_POPULATE = False

populate_obj(*obj*, *name*)

process_formdata(*valuelist*)

class indico.web.forms.fields.**HiddenFieldList**(*label=None*, *validators=None*, *filters=()*, *description=u''*, *id=None*, *default=None*, *widget=None*, *render_kw=None*, *_form=None*, *_name=None*, *_prefix=u''*, *_translations=None*, *_meta=None*)

Bases: `wtforms.fields.simple.HiddenField`

A hidden field that handles lists of strings.

This is done *getlist*-style, i.e. by repeating the input element with the same name for each list item.

The only case where this field is useful is when you display a form via POST and provide a list of items (e.g. ids) related to the form which needs to be kept when the form is submitted and also need to access it via `request.form.getlist(...)` before submitting the form.

Construct a new field.

Parameters

- **label** – The label of the field.
- **validators** – A sequence of validators to call when *validate* is called.
- **filters** – A sequence of filters which are run on input data by *process*.
- **description** – A description for the field, typically used for help text.
- **id** – An id to use for the field. A reasonable default is set by the form, and you shouldn't need to set this manually.
- **default** – The default value to assign to the field, if no form or object input is provided. May be a callable.
- **widget** – If provided, overrides the widget used to render the field.
- **render_kw** (*dict*) – If provided, a dictionary which provides default keywords that will be given to the widget at render time.
- **_form** – The form holding this field. It is passed by the form itself during construction. You should never pass this value yourself.
- **_name** – The name of this field, passed by the enclosing form during its construction. You should never pass this value yourself.

- **_prefix** – The prefix to prepend to the form name of this field, passed by the enclosing form during construction.
- **_translations** – A translations object providing message translations. Usually passed by the enclosing form during construction. See I18n docs for information on message translations.
- **_meta** – If provided, this is the 'meta' instance from the form. You usually don't pass this yourself.

If *_form* and *_name* isn't provided, an `UnboundField` will be returned instead. Call its `bind()` method with a form instance and a name to construct the field.

process_formdata(*valuelist*)

widget = <indico.web.forms.widgets.HiddenInputs object>

class indico.web.forms.fields.**TextListField**(*label=None*, *validators=None*, *filters=()*, *description=u''*, *id=None*, *default=None*, *widget=None*, *render_kw=None*, *_form=None*, *_name=None*, *_prefix=u''*, *_translations=None*, *_meta=None*)

Bases: `wtforms.fields.simple.TextAreaField`

Construct a new field.

Parameters

- **label** – The label of the field.
- **validators** – A sequence of validators to call when *validate* is called.
- **filters** – A sequence of filters which are run on input data by *process*.
- **description** – A description for the field, typically used for help text.
- **id** – An id to use for the field. A reasonable default is set by the form, and you shouldn't need to set this manually.
- **default** – The default value to assign to the field, if no form or object input is provided. May be a callable.
- **widget** – If provided, overrides the widget used to render the field.
- **render_kw** (*dict*) – If provided, a dictionary which provides default keywords that will be given to the widget at render time.
- **_form** – The form holding this field. It is passed by the form itself during construction. You should never pass this value yourself.
- **_name** – The name of this field, passed by the enclosing form during its construction. You should never pass this value yourself.
- **_prefix** – The prefix to prepend to the form name of this field, passed by the enclosing form during construction.
- **_translations** – A translations object providing message translations. Usually passed by the enclosing form during construction. See I18n docs for information on message translations.
- **_meta** – If provided, this is the 'meta' instance from the form. You usually don't pass this yourself.

If *_form* and *_name* isn't provided, an `UnboundField` will be returned instead. Call its `bind()` method with a form instance and a name to construct the field.

pre_validate(*form*)

process_formdata(*valuelist*)

class indico.web.forms.fields.**EmailListField**(*label=None*, *validators=None*, *filters=()*, *description=u''*, *id=None*, *default=None*, *widget=None*, *render_kw=None*, *_form=None*, *_name=None*, *_prefix=u''*, *_translations=None*, *_meta=None*)

Bases: indico.web.forms.fields.simple.TextListField

Construct a new field.

Parameters

- **label** – The label of the field.
- **validators** – A sequence of validators to call when *validate* is called.
- **filters** – A sequence of filters which are run on input data by *process*.
- **description** – A description for the field, typically used for help text.
- **id** – An id to use for the field. A reasonable default is set by the form, and you shouldn't need to set this manually.
- **default** – The default value to assign to the field, if no form or object input is provided. May be a callable.
- **widget** – If provided, overrides the widget used to render the field.
- **render_kw** (*dict*) – If provided, a dictionary which provides default keywords that will be given to the widget at render time.
- **_form** – The form holding this field. It is passed by the form itself during construction. You should never pass this value yourself.
- **_name** – The name of this field, passed by the enclosing form during its construction. You should never pass this value yourself.
- **_prefix** – The prefix to prepend to the form name of this field, passed by the enclosing form during construction.
- **_translations** – A translations object providing message translations. Usually passed by the enclosing form during construction. See I18n docs for information on message translations.
- **_meta** – If provided, this is the 'meta' instance from the form. You usually don't pass this yourself.

If *_form* and *_name* isn't provided, an UnboundField will be returned instead. Call its bind() method with a form instance and a name to construct the field.

process_formdata(*valuelist*)

class indico.web.forms.fields.**IndicoPasswordField**(**args*, ***kwargs*)

Bases: wtforms.fields.simple.PasswordField

Password field which can show or hide the password.

widget = <indico.web.forms.widgets.PasswordWidget object>

class indico.web.forms.fields.**IndicoStaticTextField**(**args*, ***kwargs*)

Bases: wtforms.fields.core.Field

Return an html element with text taken from this field's value

process_data(*data*)

widget = <indico.web.forms.widgets.JinjaWidget object>

class indico.web.forms.fields.**IndicoTagListField**(*label=None*, *validators=None*, *filters=()*, *description=u''*, *id=None*, *default=None*, *widget=None*, *render_kw=None*, *_form=None*, *_name=None*, *_prefix=u''*, *_translations=None*, *_meta=None*)

Bases: indico.web.forms.fields.simple.HiddenFieldList

Construct a new field.

Parameters

- **label** – The label of the field.
- **validators** – A sequence of validators to call when *validate* is called.
- **filters** – A sequence of filters which are run on input data by *process*.
- **description** – A description for the field, typically used for help text.
- **id** – An id to use for the field. A reasonable default is set by the form, and you shouldn't need to set this manually.
- **default** – The default value to assign to the field, if no form or object input is provided. May be a callable.
- **widget** – If provided, overrides the widget used to render the field.
- **render_kw** (*dict*) – If provided, a dictionary which provides default keywords that will be given to the widget at render time.
- **_form** – The form holding this field. It is passed by the form itself during construction. You should never pass this value yourself.
- **_name** – The name of this field, passed by the enclosing form during its construction. You should never pass this value yourself.
- **_prefix** – The prefix to prepend to the form name of this field, passed by the enclosing form during construction.
- **_translations** – A translations object providing message translations. Usually passed by the enclosing form during construction. See I18n docs for information on message translations.
- **_meta** – If provided, this is the 'meta' instance from the form. You usually don't pass this yourself.

If *_form* and *_name* isn't provided, an UnboundField will be returned instead. Call its bind() method with a form instance and a name to construct the field.

widget = <indico.web.forms.widgets.JinjaWidget object>

class indico.web.forms.fields.**IndicoPalettePickerField**(**args*, ***kwargs*)

Bases: indico.web.forms.fields.simple.JSONField

Field allowing user to pick a color from a set of predefined values

CAN_POPULATE = True

pre_validate(*form*)

process_data(*value*)

process_formdata(*valuelist*)

widget = <indico.web.forms.widgets.JinjaWidget object>

class indico.web.forms.fields.**TimeDeltaField**(**args*, ***kwargs*)
Bases: wtforms.fields.core.Field

A field that lets the user select a simple timedelta.

It does not support mixing multiple units, but it is smart enough to switch to a different unit to represent a timedelta that could not be represented otherwise.

Parameters **units** – The available units. Must be a tuple containing any any of 'seconds', 'minutes', 'hours' and 'days'. If not specified, `('hours', 'days')` is assumed.

best_unit
Return the largest unit that covers the current timedelta

choices

magnitudes = OrderedDict([(u'days', 86400), (u'hours', 3600), (u'minutes', 60), (u'seconds', 1)])

pre_validate(*form*)

process_formdata(*valuelist*)

unit_names = {u'seconds': u'Seconds', u'hours': u'Hours', u'minutes': u'Minutes', u'days': u'Days'}

widget = <indico.web.forms.widgets.JinjaWidget object>

class indico.web.forms.fields.**IndicoDateTimeField**(**args*, ***kwargs*)
Bases: wtforms.ext.dateutil.fields.DateTimeField

Friendly datetime field that handles timezones and validations.

Important: When the form has a *timezone* field it must be declared before any *IndicoDateTimeField*. Otherwise its value is not available in this field resulting in an error during form submission.

earliest_dt

latest_dt

linked_datetime_validator

linked_field

pre_validate(*form*)

process_formdata(*valuelist*)

timezone

timezone_field

widget = <indico.web.forms.widgets.JinjaWidget object>

class indico.web.forms.fields.**OccurrencesField**(**args*, ***kwargs*)
Bases: indico.web.forms.fields.simple.JSONField

A field that lets you select multiple occurrences consisting of a start date/time and a duration.

CAN_POPULATE = True

process_formdata(*valuelist*)

timezone

timezone_field

widget = <indico.web.forms.widgets.JinjaWidget object>

class indico.web.forms.fields.**IndicoTimezoneSelectField**(**args*, ***kwargs*)
Bases: wtforms.fields.core.SelectField

process_data(*value*)

class indico.web.forms.fields.**IndicoEnumSelectField**(*label=None*, *validators=None*, *enum=None*, *sorted=False*, *only=None*, *skip=None*, *none=None*, *titles=None*, ***kwargs*)
Bases: indico.web.forms.fields.enums._EnumFieldMixin, wtforms.fields.core.SelectFieldBase

Select field backed by a RichEnum

iter_choices()

widget = <wtforms.widgets.core.Select object>

class indico.web.forms.fields.**IndicoEnumRadioField**(*label=None*, *validators=None*, *enum=None*, *sorted=False*, *only=None*, *skip=None*, *none=None*, *titles=None*, ***kwargs*)
Bases: indico.web.forms.fields.enums.IndicoEnumSelectField

option_widget = <wtforms.widgets.core.RadioInput object>

widget = <indico.web.forms.widgets.JinjaWidget object>

class indico.web.forms.fields.**HiddenEnumField**(*label=None*, *validators=None*, *enum=None*, *only=None*, *skip=None*, *none=None*, ***kwargs*)
Bases: indico.web.forms.fields.enums._EnumFieldMixin, wtforms.fields.simple.HiddenField

Hidden field that only accepts values from an Enum

process_formdata(*valuelist*)

class indico.web.forms.fields.**FileField**(**args*, ***kwargs*)
Bases: wtforms.fields.core.Field

A dropzone field

default_options = {u'multiple_files': False, u'handle_flashes': False, u'lightweight': False, u'add_remove_links': Tru

process_formdata(*valuelist*)

widget = <indico.web.forms.widgets.JinjaWidget object>

class indico.web.forms.fields.**MultiStringField**(**args*, ***kwargs*)
Bases: wtforms.fields.simple.HiddenField

A field with multiple input text fields.

Parameters

- **field** – A tuple (fieldname, title) where the title is used in the placeholder.
- **uuid_field** – If set, each item will have a UUID assigned and stored in the field specified here.
- **flat** – If True, the field returns a list of string values instead of dicts. Cannot be combined with *uuid_field*.

- **unique** – Whether the values should be unique.
- **sortable** – Whether items should be sortable.

pre_validate(*form*)

process_formdata(*valuelist*)

widget = <indico.web.forms.widgets.JinjaWidget object>

class indico.web.forms.fields.**MultipleItemsField**(**args*, ***kwargs*)
Bases: wtforms.fields.simple.HiddenField

A field with multiple items consisting of multiple string values.

Parameters

- **fields** – A list of dicts with the following arguments: 'id': the unique ID of the field 'caption': the title of the column and the placeholder 'type': 'text|number|select', the type of the field 'coerce': callable to convert the value to a python type.

  the type must be comvertible back to a string, so usually you just want something like *int* or *float* here.

  In case the type is 'select', the property 'choices' of the *MultipleItemsField* or the 'choices' kwarg needs to be a dict where the key is the 'id' of the select field and the value is another dict mapping the option's id to it caption.
- **uuid_field** – If set, each item will have a UUID assigned and stored in the field specified here. The name specified here may not be in *fields*.
- **uuid_field_opaque** – If set, the *uuid_field* is considered opaque, i.e. it is never touched by this field. This is useful when you subclass the field and use e.g. actual database IDs instead of UUIDs.
- **unique_field** – The name of a field in *fields* that needs to be unique.
- **sortable** – Whether items should be sortable.

pre_validate(*form*)

process_formdata(*valuelist*)

widget = <indico.web.forms.widgets.JinjaWidget object>

class indico.web.forms.fields.**OverrideMultipleItemsField**(**args*, ***kwargs*)
Bases: wtforms.fields.simple.HiddenField

A field similar to *MultipleItemsField* which allows the user to override some values.

Parameters

- **fields** – a list of `(fieldname, title)` tuples. Should match the fields of the corresponding *MultipleItemsField*.
- **field_data** – the data from the corresponding *MultipleItemsField*.
- **unique_field** – the name of the field which is unique among all rows
- **edit_fields** – a set containing the field names which can be edited

If you decide to use this field, please consider adding support for *uuid_field* here!

get_overridden_value(*row*, *name*)
Utility for the widget to get the entered value for an editable field

get_row_key(*row*)
Utility for the widget to get the unique value for a row

pre_validate(*form*)

process_formdata(*valuelist*)

widget = <indico.web.forms.widgets.JinjaWidget object>

class indico.web.forms.fields.**PrincipalListField**(**args*, ***kwargs*)
Bases: wtforms.fields.simple.HiddenField

A field that lets you select a list Indico user/group ("principal")

Parameters

- **groups** – If groups should be selectable.
- **allow_networks** – If ip networks should be selectable.
- **allow_emails** – If emails should be allowed.
- **allow_external** – If "search users with no indico account" should be available. Selecting such a user will automatically create a pending user once the form is submitted, even if other fields in the form fail to validate!

pre_validate(*form*)

process_formdata(*valuelist*)

widget = <indico.web.forms.widgets.JinjaWidget object>

class indico.web.forms.fields.**PrincipalField**(**args*, ***kwargs*)
Bases: indico.web.forms.fields.principals.PrincipalListField

A field that lets you select an Indico user/group ("principal")

process_formdata(*valuelist*)

widget = <indico.web.forms.widgets.JinjaWidget object>

class indico.web.forms.fields.**AccessControlListField**(**args*, ***kwargs*)
Bases: indico.web.forms.fields.principals.PrincipalListField

widget = <indico.web.forms.widgets.JinjaWidget object>

class indico.web.forms.fields.**IndicoQuerySelectMultipleField**(**args*, ***kwargs*)
Bases: wtforms.ext.sqlalchemy.fields.QuerySelectMultipleField

Like the parent, but with a callback that allows you to modify the list

The callback can return a new list or yield items, and you can use it e.g. to sort the list.

data

class indico.web.forms.fields.**IndicoLocationField**(**args*, ***kwargs*)
Bases: indico.web.forms.fields.simple.JSONField

CAN_POPULATE = True

process_formdata(*valuelist*)

widget = <indico.web.forms.widgets.LocationWidget object>

class indico.web.forms.fields.**IndicoMarkdownField**(**args*, ***kwargs*)
Bases: wtforms.fields.simple.TextAreaField

A Markdown-enhanced textarea.

When using the editor you need to include the markdown JS/CSS bundles and also the MathJax JS bundle (even when using only the editor without Mathjax).

Parameters

- **editor** – Whether to use the WMD widget with its live preview
- **mathjax** – Whether to use MathJax in the WMD live preview

widget = <indico.web.forms.widgets.JinjaWidget object>

class indico.web.forms.fields.**IndicoDateField**(**args*, ***kwargs*)
Bases: wtforms.ext.dateutil.fields.DateField

widget = <indico.web.forms.widgets.JinjaWidget object>

class indico.web.forms.fields.**IndicoProtectionField**(**args*, ***kwargs*)
Bases: indico.web.forms.fields.enums.IndicoEnumRadioField

radio_widget = <indico.web.forms.widgets.JinjaWidget object>

render_protection_message()

widget = <indico.web.forms.widgets.JinjaWidget object>

class indico.web.forms.fields.**IndicoSelectMultipleCheckboxBooleanField**(*label=None*, *validators=None*, *coerce=<type 'unicode'>*, *choices=None*, ***kwargs*)
Bases: indico.web.forms.fields.simple.IndicoSelectMultipleCheckboxField

iter_choices()

process_formdata(*valuelist*)

class indico.web.forms.fields.**RelativeDeltaField**(**args*, ***kwargs*)
Bases: wtforms.fields.core.Field

A field that lets the user select a simple timedelta.

It does not support mixing multiple units, but it is smart enough to switch to a different unit to represent a timedelta that could not be represented otherwise.

Parameters **units** – The available units. Must be a tuple containing any any of 'seconds', 'minutes', 'hours' and 'days'. If not specified, `('hours', 'days')` is assumed.

choices

magnitudes = OrderedDict([(u'years', relativedelta(years=+1)), (u'months', relativedelta(months=+1)), (u'weeks', relati

pre_validate(*form*)

process_formdata(*valuelist*)

split_data

unit_names = {u'seconds': u'Seconds', u'months': u'Months', u'days': u'Days', u'years': u'Years', u'hours': u'Hours'

widget = <indico.web.forms.widgets.JinjaWidget object>

class indico.web.forms.fields.**IndicoWeekDayRepetitionField**(**args*, ***kwargs*)
Bases: wtforms.fields.core.Field

Field that lets you select an ordinal day of the week.

WEEK_DAY_NUMBER_CHOICES = ((1, lu'first'), (2, lu'second'), (3, lu'third'), (4, lu'fourth'), (-1, lu'last'))

day_number_data

process_formdata(*valuelist*)

week_day_data

widget = <indico.web.forms.widgets.JinjaWidget object>

class indico.web.forms.fields.**IndicoEmailRecipientsField**(*label=None*, *validators=None*, *filters=()*, *description=u''*, *id=None*, *default=None*, *widget=None*, *render_kw=None*, *_form=None*, *_name=None*, *_prefix=u''*, *_translations=None*, *_meta=None*)
Bases: wtforms.fields.core.Field

Construct a new field.

Parameters

- **label** – The label of the field.
- **validators** – A sequence of validators to call when *validate* is called.
- **filters** – A sequence of filters which are run on input data by *process*.
- **description** – A description for the field, typically used for help text.
- **id** – An id to use for the field. A reasonable default is set by the form, and you shouldn't need to set this manually.
- **default** – The default value to assign to the field, if no form or object input is provided. May be a callable.
- **widget** – If provided, overrides the widget used to render the field.
- **render_kw** (*dict*) – If provided, a dictionary which provides default keywords that will be given to the widget at render time.
- **_form** – The form holding this field. It is passed by the form itself during construction. You should never pass this value yourself.
- **_name** – The name of this field, passed by the enclosing form during its construction. You should never pass this value yourself.
- **_prefix** – The prefix to prepend to the form name of this field, passed by the enclosing form during construction.
- **_translations** – A translations object providing message translations. Usually passed by the enclosing form during construction. See I18n docs for information on message translations.
- **_meta** – If provided, this is the 'meta' instance from the form. You usually don't pass this yourself.

If *_form* and *_name* isn't provided, an UnboundField will be returned instead. Call its bind() method with a form instance and a name to construct the field.

process_data(*data*)

widget = <indico.web.forms.widgets.JinjaWidget object>

CHAPTER 6

Indices and tables

- genindex
- modindex

Python Module Index

i

Index

A

B

C

D

E

F

G

H

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z