Comprehensive Knowledge Archive Network (CKAN) Developer Documentation

Release 1.7.4

Open Knowledge Foundation

April 23, 2014

Note: This is the documentation for CKAN version '1.7.4'. If you are using a different version, use the links on the bottom right corner of the page to select the appropriate documentation.

This Administration Guide covers how to set up and manage CKAN software.

- The first two sections cover your two options for installing CKAN: package or source install.
- The rest of the first half of the Guide, up to *Set and Manage Permissions*, cover setup and basic admin.
- The second half of the Guide, from *Add Extensions* onwards, covers advanced tasks, including extensions and forms.

For high-level information on what CKAN is, see the CKAN website.

CHAPTER 1

Installation

1.1 Option 1: Package Installation

This section describes how to install CKAN from packages. This is the recommended and by far the easiest way to install CKAN.

The overall process is the following:

- *Prepare your system*
- *Run the Package Installer*
- *Upgrading a package install*

Note: We recommend you use package installation unless you are a core CKAN developer or have no access to Ubuntu 10.04 through any of the methods above, in which case, you should use *Option 2: Install from Source*.

For support during installation, please contact the ckan-dev mailing list.

1.1.1 Prepare your system

Package install requires you to use Ubuntu 10.04: either locally, through a virtual machine or Amazon EC2.

You can:

- Use Ubuntu 10.04 directly.
- `install-from-package-virtualbox`. This is suitable if you want to host your CKAN instance on a machine running any other OS.
- `install-from-package-amazon`. This is suitable if you want to host your CKAN instance in the cloud, on a ready-made Ubuntu OS.

1.1.2 Run the Package Installer

On your Ubuntu 10.04 system, open a terminal and run these commands to prepare your system (replace *MAJOR_VERSION* with a suitable value):

```
sudo apt-get update
sudo apt-get install -y wget
echo "deb http://apt.ckan.org/ckan-1.MAJOR_VERSION lucid universe" | sudo tee /etc/apt/sources.list.
```

```
wget -qO- "http://apt.ckan.org/packages_public.key" | sudo apt-key add -
sudo apt-get update
```

Now you are ready to install. If you already have a PostgreSQL and Solr instance that you want to use set up on a different server you don't need to install `postgresql-8.4` and `solr-jetty` locally. For most cases you'll need CKAN, PostgreSQL and Solr all running on the same server so run:

```
sudo apt-get install -y ckan postgresql-8.4 solr-jetty
```

The install will whirr away. With `ckan`, `postgresql-8.4` and `solr-jetty` chosen, over 180Mb of packages will be downloaded (on a clean install). This will take a few minutes, then towards the end you'll see this:

```
Setting up solr-jetty (1.4.0+ds1-1ubuntu1) ...
 * Not starting jetty - edit /etc/default/jetty and change NO_START to be 0 (or comment it out).
```

If you've installed `solr-jetty` locally you'll also need to configure your local Solr server for use with CKAN. You can do so like this:

```
sudo ckan-setup-solr
```

This changes the Solr schema to support CKAN, sets Solr to start automatically and then starts Solr. You shouldn't be using the Solr instance for anything apart from CKAN because the command above modifies its schema.

You can now create CKAN instances as you please using the `ckan-create-instance` command. It takes these arguments:

Instance name

> This should be a short letter only string representing the name of the CKAN instance. It is used (amongst other things) as the basis for:
>
> - The directory structure of the instance in `/var/lib/ckan`, `/var/log/ckan`, `/etc/ckan` and elsewhere
> - The name of the PostgreSQL database to use
> - The name of the Solr core to use

Instance Hostname/domain name

> The hostname that this CKAN instance will be hosted at. It is used in the Apache configuration virutal host in `/etc/apache2/sites-available/<INSTANCE_NAME>.common` so that Apache can resolve requests directly to CKAN.
>
> If you are using Amazon EC2, you will use the public DNS of your server as this argument. These look something like `ec2-79-125-86-107.eu-west-1.compute.amazonaws.com`. If you are using a VM, this will be the hostname of the VM you have configured in your `/etc/hosts` file.
>
> If you install more than one CKAN instance you'll need to set different hostnames for each. If you ever want to change the hostname CKAN responds on you can do so by editing `/etc/apache2/sites-available/<INSTANCE_NAME>.common` and restarting apache with `sudo /etc/init.d/apache2 restart`.

Local PostgreSQL support (`"yes"` or `"no"`)

> If you specify `"yes"`, CKAN will also set up a local database user and database and create its tables, populating them as necessary and saving the database password in the config file. You would normally say `"yes"` unless you plan to use CKAN with a PostgreSQL on a remote machine.
>
> If you choose `"no"` as the third parameter to tell the install command not to set up or configure the PostgreSQL database for CKANi you'll then need to perform any database creation and setup steps manually yourself.

For production use the second argument above is usually the domain name of the CKAN instance, but in our case we are testing, so we'll use the default hostname buildkit sets up to the server which is `default.vm.buildkit` (this is automatically added to your host machine's `/etc/hosts` when the VM is started so that it will resovle from your host machine - for more complex setups you'll have to set up DNS entries instead).

Create a new instance like this:

```
sudo ckan-create-instance std default.vm.buildkit yes
```

You'll need to specify a new instance name and different hostname for each CKAN instance you set up.

Don't worry about warnings you see like this during the creation process, they are harmless:

```
/usr/lib/pymodules/python2.6/ckan/sqlalchemy/engine/reflection.py:46: SAWarning: Did not recognize ty
```

You can now access your CKAN instance from your host machine as http://default.vm.buildkit/

Tip: If you get taken straight to a login screen it is a sign that the PostgreSQL database initialisation may not have run. Try running:

```
INSTANCE=std
sudo paster --plugin=ckan db init --config=/etc/ckan/${INSTANCE}/${INSTANCE}.ini
```

If you specified `"no"` as part of the `create-ckan-instance` you'll need to specify database and solr settings in `/etc/ckan/std/std.ini`. At the moment you'll see an "Internal Server Error" from Apache. You can always investigate such errors by looking in the Apache and CKAN logs for that instance.

Sometimes things don't go as planned so let's look at some of the log files.

This is the CKAN log information (leading data stripped for clarity):

```
$ sudo -u ckanstd tail -f /var/log/ckan/std/std.log
WARNI [vdm] Skipping adding property Package.all_revisions_unordered to revisioned object
WARNI [vdm] Skipping adding property PackageTag.all_revisions_unordered to revisioned object
WARNI [vdm] Skipping adding property Group.all_revisions_unordered to revisioned object
WARNI [vdm] Skipping adding property PackageGroup.all_revisions_unordered to revisioned object
WARNI [vdm] Skipping adding property GroupExtra.all_revisions_unordered to revisioned object
WARNI [vdm] Skipping adding property PackageExtra.all_revisions_unordered to revisioned object
WARNI [vdm] Skipping adding property Resource.all_revisions_unordered to revisioned object
WARNI [vdm] Skipping adding property ResourceGroup.resources_all to revisioned object
```

No error here, let's look in Apache (leading data stripped again) in the case where we chose `"no"` to PostgreSQL installation:

```
$ tail -f /var/log/apache2/std.error.log
    self.connection = self.__connect()
  File "/usr/lib/pymodules/python2.6/ckan/sqlalchemy/pool.py", line 319, in __connect
    connection = self.__pool._creator()
  File "/usr/lib/pymodules/python2.6/ckan/sqlalchemy/engine/strategies.py", line 82, in connect
    return dialect.connect(*cargs, **cparams)
  File "/usr/lib/pymodules/python2.6/ckan/sqlalchemy/engine/default.py", line 249, in connect
    return self.dbapi.connect(*cargs, **cparams)
OperationalError: (OperationalError) FATAL:  password authentication failed for user "ckanuser"
FATAL:  password authentication failed for user "ckanuser"
 None None
```

There's the problem. If you don't choose `"yes"` to install PostgreSQL, you need to set up the `sqlalchemy.url` option in the config file manually. Edit it to set the correct settings:

```
sudo -u ckanstd vi /etc/ckan/std/std.ini
```

Notice how you have to make changes to CKAN config files and view CKAN log files using the username set up for your CKAN user.

Each instance you create has its own virtualenv that you can install extensions into at `/var/lib/ckan/std/pyenv` and its own system user, in this case `ckanstd`. Any time you make changes to the virtualenv, you should make sure you are running as the correct user otherwise Apache might not be able to load CKAN. For example, say you wanted to install a ckan extension, you might run:

```
sudo -u ckanstd /var/lib/ckan/std/pyenv/bin/pip install <name-of-extension>
```

You can now configure your instance by editing `/etc/ckan/std/std.ini`:

```
sudo -u ckanstd vi /etc/ckan/std/std.ini
```

After any change you can touch the `wsgi.py` to tell Apache's mod_wsgi that it needs to take notice of the change for future requests:

```
sudo touch /var/lib/ckan/std/wsgi.py
```

Or you can of course do a full restart if you prefer:

```
sudo /etc/init.d/apache2 restart
```

Caution: CKAN has etag caching enabled by default which encourages your browser to cache the homepage and all the dataset pages. This means that if you change CKAN's configuration you'll need to do a 'force refresh' by pressing `Shift + Ctrl + F5` together or `Shift + Ctrl + R` (depending on browser) before you'll see the change.

One of the key things it is good to set first is the `ckan.site_description` option. The text you set there appears in the banner at the top of your CKAN instance's pages.

You can enable and disable particular CKAN instances by running:

```
sudo a2ensite std
sudo /etc/init.d/apache2 reload
```

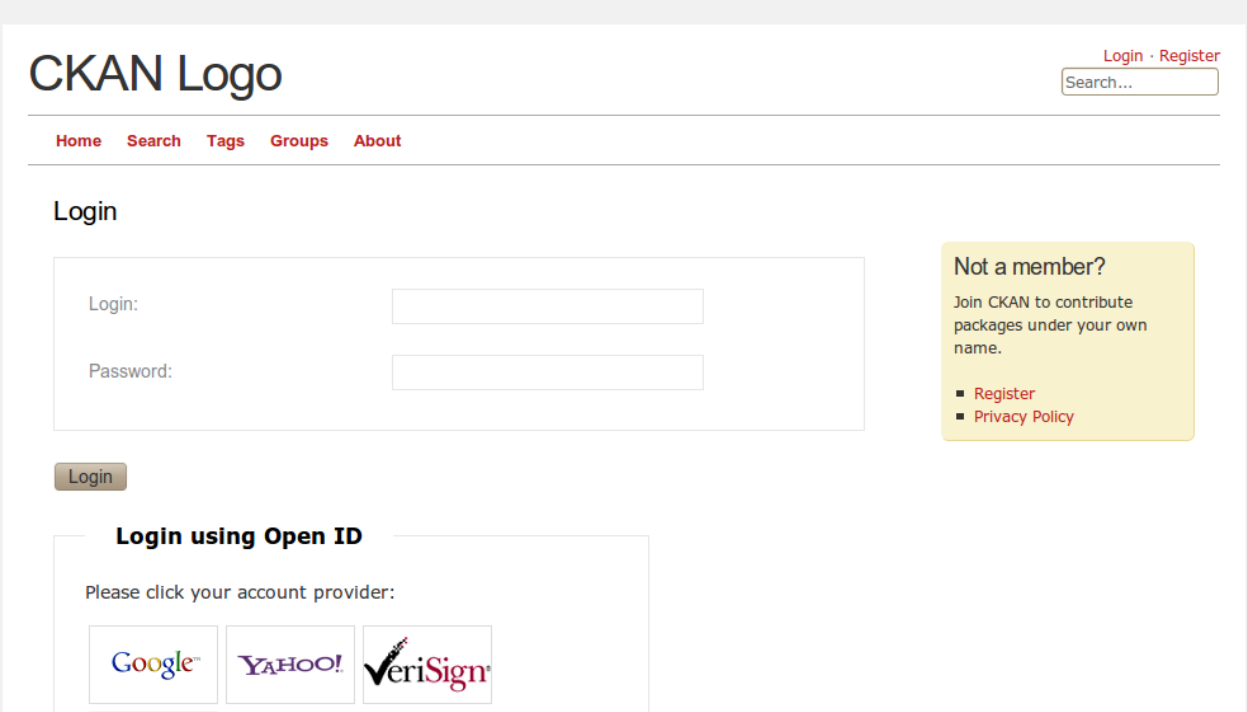
or:

```
sudo a2dissite std
sudo /etc/init.d/apache2 reload
```

respectively.

Now you should be up and running. Don't forget you there is the a help page for dealing with *Common error messages*.

Visit your CKAN instance - either at your Amazon EC2 hostname, or at on your host PC or virtual machine. You'll be redirected to the login screen because you won't have set up any permissions yet, so the welcome screen will look something like this.



You can now proceed to *Post-Installation Setup*.

Warning: If you use the `ckan-create-instance` command to create more than one instance there are a couple of things you need to be aware of. Firstly, you need to change the Apache configurations to put `mod_wsgi` into *daemon* mode and secondly you need to watch your Solr search index carefully to make sure that the different instances are not over-writing each other's data.
To change the Apache configuration uncomment the following lines for each instance in `/etc/apache2/sites-available/std.common` and make sure `${INSTANCE}` is replaced with your instance name:

```
# Deploy as a daemon (avoids conflicts between CKAN instances)
# WSGIDaemonProcess ${INSTANCE} display-name=${INSTANCE} processes=4 threads=15 maximum-requests=10
# WSGIProcessGroup ${INSTANCE}
```

If you don't do this and you install different versions of the same Python packages into the different pyenvs in `/var/lib/ckan` for each instance, there is a chance the CKAN instances might use the wrong package.
If you want to make sure that you CKAN instances are using different Solr indexes, you can configure Solr to run in multi-core mode. See *Multiple Solr cores* for more details.

CKAN packaging is well tested and reliable with single instance CKAN installs. Multi-instance support is newer, and whilst we believe will work well, hasn't had the same degree of testing. If you hit any problems with multi-instance installs, do let us know and we'll help you fix them.

1.1.3 Upgrading a package install

Starting on CKAN 1.7, the updating process is different depending on wether the new version is a major release (e.g. 1.7, 1.8, etc) or a minor release (e.g. 1.7.X, 1.7.Y). Major releases can introduce backwards incompatible changes, changes on the database and the Solr schema. Each major release and its subsequent minor versions has its own apt repository (Please note that this was not true for 1.5 and 1.5.1 versions).

Minor versions, on the other hand contain only bug fixes, non-breaking optimizations and new translations.

A fresh install or upgrade from another major version will install the latest minor version.

Upgrading from another major version

If you already have a major version installed via package install and wish to upgrade, you can try the approach documented below.

> **Caution:** Always make a backup first and be prepared to start again with a fresh install of the newer version of CKAN.

First remove the old CKAN code (it doesn't remove your data):

```
sudo apt-get autoremove ckan
```

Then update the repositories (replace *MAJOR_VERSION* with a suitable value):

```
echo "deb http://apt.ckan.org/ckan-1.MAJOR_VERSION lucid universe" | sudo tee /etc/apt/sources.list.
wget -qO- "http://apt.ckan.org/packages_public.key" | sudo apt-key add -
sudo apt-get update
```

Install the new CKAN and update all the dependencies:

```
sudo apt-get install ckan
```

Now you need to make some manual changes. In the following commands replace `std` with the name of your CKAN instance. Perform these steps for each instance you wish to upgrade.

1. Upgrade the Solr schema

   > **Note:** This only needs to be done if the Solr schema has been updated between major releases. The CHANGELOG or the announcement emails will specify if this is the case.

   Configure `ckan.site_url` or `ckan.site_id` in `/etc/ckan/std/std.ini` for SOLR search-index rebuild to work. eg:

   ```
   ckan.site_id = yoursite.ckan.org
   ```

   The site_id must be unique so the domain name of the CKAN instance is a good choice.

   Install the new schema:

   ```
   sudo rm /usr/share/solr/conf/schema.xml
   sudo ln -s /usr/lib/pymodules/python2.6/ckan/config/solr/schema-1.4.xml /usr/share/solr/conf/sch
   ```

2. Upgrade the database

   First install pastescript:

   ```
   sudo -u ckanstd /var/lib/ckan/std/pyenv/bin/pip install --ignore-installed pastescript
   ```

   Then upgrade the database:

   ```
   sudo -u ckanstd /var/lib/ckan/std/pyenv/bin/paster --plugin=ckan db upgrade --config=/etc/ckan/s
   ```

   When upgrading from CKAN 1.5 you may experience error `sqlalchemy.exc.IntegrityError: (IntegrityError) could not create unique index "user_name_key`. In this case then you need to rename users with duplicate names, before the database upgrade will run successfully. For example:

```
sudo -u ckanstd paster --plugin=pylons shell /etc/ckan/std/std.ini
model.meta.engine.execute('SELECT name, count(name) AS NumOccurrences FROM "user" GROUP BY name
users = model.Session.query(model.User).filter_by(name='https://www.google.com/accounts/o8/id?id
users[1].name = users[1].name[:-1]
model.repo.commit_and_remove()
```

3. Rebuild the search index (this can take some time - e.g. an hour for 5000 datasets):

```
sudo -u ckanstd /var/lib/ckan/std/pyenv/bin/paster --plugin=ckan search-index rebuild --config=/
```

4. Restart Apache

```
sudo /etc/init.d/apache2 restart
```

Upgrading from the same major version

If you want to update to a new minor version of a major release (e.g. upgrade to 1.7.1 to 1.7, or to 1.7.2 from 1.7.1), then you only need to update the *python-ckan* package to get the latest changes:

```
sudo apt-get install python-ckan
```

> **Caution:** This assumes that you already have installed CKAN via package install. If not, do not install this single package, follow the instructions on *Run the Package Installer*

After upgrading the package, you need to restart Apache for the effects to take place:

```
sudo /etc/init.d/apache2 restart
```

1.2 Option 2: Install from Source

This section describes how to install CKAN from source. Whereas *Option 1: Package Installation* requires Ubuntu 10.04, this way of installing CKAN is more flexible to work with other distributions and operating systems. Please share your experiences on our wiki: http://wiki.ckan.org/Install

This is also the option to use if you are going to develop the CKAN source.

> **Warning:** This option is more complex than *Option 1: Package Installation*.

There is a page of help for dealing with *Common error messages*.

For support during installation, please contact the ckan-dev mailing list.

1.2.1 1. Ensure the required packages are installed

If you have access to `apt-get`, you can install these packages as follows:

```
sudo apt-get install mercurial python-dev postgresql libpq-dev
sudo apt-get install libxml2-dev libxslt-dev python-virtualenv
sudo apt-get install wget build-essential git-core subversion
sudo apt-get install solr-jetty openjdk-6-jdk
```

Otherwise, you should install these packages from source.

Package	Description
mercurial	Source control
python	Python v2.5-2.7
postgresql	PostgreSQL database
libpq	PostgreSQL library
libxml2	XML library development files
libxslt	XSLT library development files
virtualenv	Python virtual environments
wget	Command line tool for downloading from the web
build-essential	Tools for building source code (or up-to-date Xcode on Mac)
git	Git source control (for getting MarkupSafe src)
subversion	Subversion source control (for pyutilib)
solr	Search engine
jetty	HTTP server (used for Solr)
openjdk-6-jdk	OpenJDK Java library

1.2.2 2. Create a Python virtual environment.

In your home directory run the command below. It is currently important to call your virtual environment `pyenv` so that the automated deployment tools work correctly.

```
cd ~
virtualenv pyenv
```

Tip: If you don't have a `python-virtualenv` package in your distribution you can get a `virtualenv.py` script from within the virtualenv source distribution and then run `python virtualenv.py pyenv` instead.

To help with automatically installing CKAN dependencies we use a tool called `pip`. Make sure you have activated your environment (see step 3) and then install it from an activated shell like this:

```
easy_install pip
```

1.2.3 3. Activate your virtual environment

To work with CKAN it is best to adjust your shell settings so that your shell uses the virtual environment you just created. You can do this like so:

```
. pyenv/bin/activate
```

When your shell is activated you will see the prompt change to something like this:

```
(pyenv)[ckan@host ~/]$
```

An activated shell looks in your virtual environment first when choosing which commands to run. If you enter `python` now it will actually run `~/pyenv/bin/python`, not the default `/usr/bin/python` which is what you want for CKAN. You can install python packages install this new environment and they won't affect the default `/usr/bin/python`. This is necessary so you can use particular versions of python packages, rather than the ones installed with default paython, and these installs do not affect other python software on your system that may not be compatible with these packages.

1.2.4 4. Install CKAN source code

Here is how to install the latest code (HEAD on the master branch):

```
pip install --ignore-installed -e git+https://github.com/okfn/ckan.git#egg=ckan
```

If you want to install a specific version, e.g. for v1.5.1:

```
pip install --ignore-installed -e git+https://github.com/okfn/ckan.git@ckan-1.7#egg=ckan
```

1.2.5 5. Install Additional Dependencies

CKAN has a set of dependencies it requires which you should install too. These are listed in three text files: requires/lucid_*.txt, followed by WebOb explicitly.

First we install two of the three lists of dependencies:

```
pip install --ignore-installed -r pyenv/src/ckan/requires/lucid_missing.txt -r pyenv/src/ckan/require
pip install webob==1.0.8
```

The `--ignore-installed` option ensures `pip` installs software into this virtual environment even if it is already present on the system.

WebOb has to be installed explicitly afterwards because by installing pylons with *–ignore-installed* you end up with a newer (incompatible) version than the one that Pylons and CKAN need.

Now to install the remaining dependencies in requires/lucid_present.txt and you are using Ubuntu Lucid 10.04 you can install the system versions:

```
sudo apt-get install python-pybabel python-psycopg2 python-lxml
sudo apt-get install python-pylons python-repoze.who
sudo apt-get install python-repoze.who-plugins python-tempita python-zope.interface
```

Alternatively, if you are not using Ubuntu Lucid 10.04 you'll need to install them like this:

```
pip install --ignore-installed -r pyenv/src/ckan/requires/lucid_present.txt
```

This will take a **long** time. Particularly the install of the `lxml` package.

At this point you will need to deactivate and then re-activate your virtual environment to ensure that all the scripts point to the correct locations:

```
deactivate
. pyenv/bin/activate
```

1.2.6 6. Setup a PostgreSQL database

List existing databases:

```
sudo -u postgres psql -l
```

Check that the encoding of databases is 'UTF8', if not internationalisation may be a problem. Since changing the encoding of PostgreSQL may mean deleting existing databases, it is suggested that this is fixed before continuing with the CKAN install.

Next you'll need to create a database user if one doesn't already exist.

Tip: If you choose a database name, user or password which are different from the example values suggested below then you'll need to change the sqlalchemy.url value accordingly in the CKAN configuration file that you'll create in the next step.

Here we create a user called `ckanuser` and will enter `pass` for the password when prompted:

```
sudo -u postgres createuser -S -D -R -P ckanuser
```

Now create the database (owned by `ckanuser`), which we'll call `ckantest`:

```
sudo -u postgres createdb -O ckanuser ckantest
```

1.2.7 7. Create a CKAN config file

Make sure you are in an activated environment (see step 3) so that Python Paste and other modules are put on the python path (your command prompt will start with `(pyenv)` if you have) then change into the `ckan` directory which will have been created when you installed CKAN in step 4 and create the CKAN config file using Paste. These instructions call it `development.ini` since that is the required name for running the CKAN tests. But for a server deployment then you might want to call it say after the server hostname e.g. `test.ckan.net.ini`.

```
cd pyenv/src/ckan
paster make-config ckan development.ini
```

If you used a different database name or password when creating the database in step 6 you'll need to now edit `development.ini` and change the `sqlalchemy.url` line, filling in the database name, user and password you used.

```
sqlalchemy.url = postgresql://ckanuser:pass@localhost/ckantest
```

If you're using a remote host with password authentication rather than SSL authentication, use:

```
sqlalchemy.url = postgresql://<user>:<password>@<remotehost>/ckan?sslmode=disable
```

Caution: Legacy installs of CKAN may have the config file in the pyenv directory, e.g. `pyenv/ckan.net.ini`. This is fine but CKAN probably won't be able to find your `who.ini` file. To fix this edit `pyenv/ckan.net.ini`, search for the line `who.config_file = %(here)s/who.ini` and change it to `who.config_file = who.ini`.

1.2.8 8. Setup Solr

Set up Solr following the instructions on *Single Solr instance* or *Multiple Solr cores* depending on your needs.

Set appropriate values for the `ckan.site_id` and `solr_url` config variables in your CKAN config file:

```
ckan.site_id=my_ckan_instance
solr_url=http://127.0.0.1:8983/solr
```

1.2.9 9. Create database tables

Now that you have a configuration file that has the correct settings for your database, you'll need to create the tables. Make sure you are still in an activated environment with `(pyenv)` at the front of the command prompt and then from the `pyenv/src/ckan` directory run this command.

If your config file is called development.ini:

```
paster --plugin=ckan db init
```

or if your config file is something else, you need to specify it. e.g.:

```
paster --plugin=ckan db init --config=test.ckan.net.ini
```

You should see `Initialising DB: SUCCESS`.

If the command prompts for a password it is likely you haven't set up the database configuration correctly in step 6.

1.2.10 10. Create the cache and session directories

You need to create two directories for CKAN to put temporary files:

- Pylon's cache directory, specified by *cache_dir* in the config file.
- Repoze.who's OpenId session directory, specified by *store_file_path* in pyenv/ckan/who.ini

(from the `pyenv/src/ckan` directory or wherever your CKAN ini file you recently created is located):

```
mkdir data sstore
```

1.2.11 11. Link to who.ini

`who.ini` (the Repoze.who configuration) needs to be accessible in the same directory as your CKAN config file. So if your config file is not in `pyenv/src/ckan`, then cd to the directory with your config file and create a symbolic link to `who.ini`. e.g.:

```
ln -s pyenv/src/ckan/who.ini
```

1.2.12 12. Test the CKAN webserver

You can use Paste to serve CKAN from the command-line. This is a simple and lightweight way to serve CKAN and is especially useful for testing. However a production deployment will probably want to be served using Apache or nginx - see *Post-Installation Setup*

Note: If you've started a new shell, you'll have to activate the environment again first - see step 3.

(from the `pyenv/src/ckan` directory):

```
paster serve development.ini
```

1.2.13 13. Browse CKAN

Point your web browser at: http://127.0.0.1:5000/

The CKAN homepage should load.

Note: if you installed CKAN on a remote machine then you will need to run the web browser on that same machine. For example run the textual web browser *w3m* in a separate ssh session to the one running *paster serve*.

Finally, if doing development you should make sure that tests pass, as described in *Basic Tests*.

1.2.14 14. You are done

You can now proceed to *Post-Installation Setup* which covers getting an administrator account created and deploying using Apache.

1.3 Post-Installation Setup

After you have completed installation (from either package or source), follow this section for instructions on setting up an initial user, loading test data, and notes on deploying CKAN.

Note: If you installed CKAN from source, you will need to activate the virtualenv and switch to the ckan source directory. In this case, you don't need to specifiy the *–plugin* or *–config* parameters when executing the paster commands, e.g.:

```
(pyenv):~/pyenv/src/ckan$ paster user list
```

1.3.1 Create an Admin User

By default, CKAN has a set of locked-down permissions. To begin working with it you need to set up a user and some permissions.

First create an admin account from the command line (you must be root, `sudo -s`):

```
paster --plugin=ckan user add admin --config=/etc/ckan/std/std.ini
```

When prompted, enter a password - this is the password you will use to log in to CKAN. In the resulting output, note that you will also get assigned a CKAN API key.

Note: This command is your first introduction to some important CKAN concepts. **paster** is the script used to run CKAN commands. **std.ini** is the CKAN config file. You can change options in this file to configure CKAN.

For exploratory purposes, you might was well make the `admin` user a sysadmin. You obviously wouldn't give most users these rights as they would then be able to do anything. You can make the `admin` user a sysadmin like this:

```
paster --plugin=ckan sysadmin add admin --config=/etc/ckan/std/std.ini
```

You can now login to the CKAN frontend with the username `admin` and the password you set up.

1.3.2 Load Test Data

It can be handy to have some test data to start with. You can get test data like this:

```
paster --plugin=ckan create-test-data --config=/etc/ckan/std/std.ini
```

You now have a CKAN instance that you can log in to, with some test data to check everything works.

You can also create various specialised test data collections for testing specific features of CKAN. For example, `paster --plugin=ckan create-test-data translations` creates some test data with some translations for testing the ckanext-multilingual extension. For more information, see:

```
paster --plugin=ckan create-test-data --help
```

1.3.3 Deployment

You may want to deploy your CKAN instance at this point, to share with others.

If you have installed CKAN from packages, then Apache and WSGI deployment scripts are already configured for you in standard locations.

If you have installed CKAN from source, then the standard production deployment of CKAN is Apache and WSGI, which you will need to configure yourself. For more information, see *CKAN Deployment*.

1.4 CKAN Deployment

Note: If you use the package installation method your site will already have been deployed using the Apache and modwsgi route described below.

This document covers how to deploy CKAN in a production setup where it is available on the Internet. This will usually involve connecting the CKAN web application to a web server such as Apache or NGinx.

As CKAN uses WSGI, a standard interface between web servers and Python web applications, CKAN can be used with a number of different web server and deployment configurations including:

- Apache with the modwsgi Apache module
- Apache with paster and reverse proxy
- Nginx with paster and reverse proxy
- Nginx with uwsgi

Note: below, we will only be able to give a few example of setups and many other ones are possible.

1.4.1 Deploying CKAN on an Ubuntu Server using Apache and modwsgi

These instructions have been tested on Ubuntu 10.04 with CKAN 1.7.

This is the standard way to deploy CKAN.

Install Apache and modwsgi

Install Apache (a web server) and modwsgi (an Apache module that adds WSGI support to Apache):

```
sudo aptitude install apache2 libapache2-mod-wsgi
```

Install CKAN

The following assumes you have installed to `/usr/local/demo.ckan.net` with your virtualenv at `/usr/local/demo.ckan.net/pyenv`.

Create the WSGI Script File

Create the WSGI script file for your CKAN instance, `/usr/local/demo.ckan.net/pyenv/bin/demo.ckan.net.py`:

```
import os
instance_dir = '/usr/local/demo.ckan.net'
config_file = '/usr/local/demo.ckan.net/pyenv/src/ckan/development.ini'
pyenv_bin_dir = os.path.join(instance_dir, 'pyenv', 'bin')
activate_this = os.path.join(pyenv_bin_dir, 'activate_this.py')
execfile(activate_this, dict(__file__=activate_this))
from paste.deploy import loadapp
config_filepath = os.path.join(instance_dir, config_file)
from paste.script.util.logging_config import fileConfig
fileConfig(config_filepath)
application = loadapp('config:%s' % config_filepath)
```

The modwsgi Apache module will redirect requests to your web server to this WSGI script file. The script file then handles those requests by directing them on to your CKAN instance (after first configuring the Python environment for CKAN to run in).

Create the Apache Config File

Create the Apache config file for your CKAN instance by copying the default Apache config file:

> cd /etc/apache2/sites-available sudo cp default demo.ckan.net

Edit `/etc/apache2/sites-available/demo.ckan.net`, before the last line (`</VirtualHost>`) add these lines:

```
ServerName demo.ckan.net
ServerAlias demo.ckan.net
WSGIScriptAlias / /usr/local/demo.ckan.net/pyenv/bin/demo.ckan.net.py

# pass authorization info on (needed for rest api)
WSGIPassAuthorization On
ErrorLog /var/log/apache2/demo.ckan.net.error.log
CustomLog /var/log/apache2/demo.ckan.net.custom.log combined
```

This tells the Apache modwsgi module to redirect any requests to the web server to the CKAN WSGI script that you created above (`demo.ckan.net.py`). Your WSGI script in turn directs the requests to your CKAN instance.

Create Directories for CKAN's Temporary Files

Make the data and sstore directories and give them the right permissions:

```
cd /usr/local/demo.ckan.net/pyenv/src/ckan/
mkdir data sstore
chmod g+w -R data sstore
sudo chgrp -R www-data data sstore
```

CKAN Log File

Edit your CKAN config file (e.g. `/usr/local/demo.ckan.net/pyenv/src/ckan/development.ini`), find this line:

```
args = ("ckan.log", "a", 20000000, 9)
```

and change it to set the ckan.log file location to somewhere that CKAN can write to, e.g.:

```
args = ("/var/log/ckan/demo.ckan.net/ckan.log", "a", 20000000, 9)
```

Then create that directory and give it the right permissions:

```
sudo mkdir -p /var/log/ckan/demo.ckan.net
sudo chown www-data /var/log/ckan/demo.ckan.net
```

Enable Your CKAN Site

Finally, enable your CKAN site in Apache:

```
sudo a2ensite demo.ckan.net
sudo /etc/init.d/apache2 restart
```

You should now be able to visit your server in a web browser and see your new CKAN instance.

Troubleshooting

Default Apache Welcome Page

If you see a default Apache welcome page where your CKAN front page should be, it may be because the default Apache config file is overriding your CKAN config file (both use port 80), so disable it and restart Apache:

```
$ sudo a2dissite default
$ sudo /etc/init.d/apache2 restart
```

403 Forbidden and 500 Internal Server Error

If you see a 403 Forbidden or 500 Internal Server Error page where your CKAN front page should be, you may have a problem with your unix file permissions. The Apache web server needs to have permission to access your WSGI script file (e.g. `/usr/local/demo.ckan.net/pyenv/bin/demo.ckan.net.py`) ''and all of its parent directories''. The permissions of the file should look like `-rw-r--r--` and the permissions of each of its parent directories should look like `drwxr-xr-x`.

IOError: sys.stdout access restricted by mod_wsgi

If you're getting 500 Internal Server Error pages and you see `IOError: sys.stdout access restricted by mod_wsgi` in your log files, it means that something in your WSGI application (e.g. your WSGI script file, your CKAN instance, or one of your CKAN extensions) is trying to print to stdout, for example by using standard Python `print` statements. WSGI applications are not allowed to write to stdout. Possible solutions include:

1. Remove the offending print statements. One option is to replace print statements with statements like `print >> sys.stderr, "..."`
2. Redirect all print statements to stderr:

   ```
   import sys
   sys.stdout = sys.stderr
   ```

3. Allow your application to print to stdout by putting `WSGIRestrictStdout Off` in your Apache config file (not recommended).

Also see https://code.google.com/p/modwsgi/wiki/ApplicationIssues

Log Files

In general, if it's not working look in the log files in `/var/log/apache2` for error messages. `demo.ckan.net.error.log` should be particularly interesting.

modwsgi wiki

Some pages on the modwsgi wiki have some useful information for troubleshooting modwsgi problems:

- https://code.google.com/p/modwsgi/wiki/ApplicationIssues
- http://code.google.com/p/modwsgi/wiki/DebuggingTechniques
- http://code.google.com/p/modwsgi/wiki/QuickConfigurationGuide
- http://code.google.com/p/modwsgi/wiki/ConfigurationGuidelines
- http://code.google.com/p/modwsgi/wiki/FrequentlyAskedQuestions
- http://code.google.com/p/modwsgi/wiki/ConfigurationIssues

1.4.2 Mounting CKAN at a non-root URL

CKAN (since version 1.6) can run mounted at a 'sub-directory' URL, such as http://mysite.com/data/. This is achieved by changing the WSGIScriptAlias first parameter (in the Apache site config). e.g.:

```
WSGIScriptAlias /data /home/dread/etc/ckan-pylons.py
```

1.4.3 CORS

As of CKAN v1.5 CORS is built in to CKAN so for CKAN >= 1.5 no modifications to your webserver config are needed.

CORS = Cross Origin Resource Sharing. It is away to allow browsers (and hence javascript in browsers) make requests to domains other than the one the browser is currently on.

In Apache you can enable CORS for you CKAN site by setting the following in your config:

```
Header always set Access-Control-Allow-Origin "*"
Header always set Access-Control-Allow-Methods "GET, POST, PUT, DELETE, OPTIONS"
Header always set Access-Control-Allow-Headers "X-CKAN-API-KEY, Content-Type"

# Respond to all OPTIONS requests with 200 OK
# This could be done in the webapp
# This is need for pre-flighted requests (POSTs/PUTs)
RewriteEngine On
RewriteCond %{REQUEST_METHOD} OPTIONS
RewriteRule ^(.*)$ $1 [R=200,L]
```

1.5 Setting up Solr

CKAN uses Solr as search platform. This document describes different topics related with the deployment and management of Solr from a CKAN point of view.

CKAN uses customized schema files that take into account its specific search needs. Different versions of the schema file are found in `ckan/ckan/config/solr`

The following instructions apply to Ubuntu 10.04 (Lucid), the supported platform by the CKAN team. Other versions or distributions may need slightly different instructions.

Note: The following instructions deploy Solr on the Jetty server, but CKAN does not require it, you can use Tomcat if that is more convenient on your distribution.

1.5.1 Single Solr instance

In this case, there will be only one Solr endpoint that uses a single schema file. This can be useful for a Solr server used by only a single CKAN instance, or different instances that share the same schema version.

To install Solr (if you are following the *Option 2: Install from Source* or *Option 1: Package Installation* instructions, you already did this):

```
sudo apt-get install solr-jetty openjdk-6-jdk
```

You'll need to edit the Jetty configuration file (*/etc/default/jetty*) with the suitable values:

```
NO_START=0            # (line 4)
JETTY_HOST=127.0.0.1  # (line 15)
JETTY_PORT=8983       # (line 18)
```

Start the Jetty server:

```
sudo service jetty start
```

You should see welcome page from Solr when visiting (replace localhost with your server address if needed):

```
http://localhost:8983/solr/
```

and the admin site:

```
http://localhost:8983/solr/admin
```

Note: If you get the message `Could not start Jetty servlet engine because no Java Development Kit (JDK) was found.` then you will have to edit the `JAVA_HOME` setting in `/etc/default/jetty` (adjusting the path for your machine's JDK install):

```
JAVA_HOME=/usr/lib/jvm/java-6-openjdk-amd64/
```

Now run:

```
sudo service jetty start
```

This default setup will use the following locations in your file system:

- */usr/share/solr*: Solr home, with a symlink pointing to the configuration dir in */etc*.
- */etc/solr/conf*: Solr configuration files. The more important ones are *schema.xml* and *solrconfig.xml*.
- */var/lib/solr/data/*: This is where the index files are physically stored.

You will obviously need to replace the default *schema.xml* file with the CKAN one. To do so, create a symbolic link to the schema file in the config folder. Use the latest schema version supported by the CKAN version you are installing (it will generally be the highest one):

```
sudo mv /etc/solr/conf/schema.xml /etc/solr/conf/schema.xml.bak
sudo ln -s ~/ckan/ckan/config/solr/schema-1.4.xml /etc/solr/conf/schema.xml
```

Now restart jetty:

```
sudo /etc/init.d/jetty stop
sudo /etc/init.d/jetty start
```

And check that Solr is running by browsing http://localhost:8983/solr/ which should offer the Administration link.

1.5.2 Multiple Solr cores

Solr can also be set up to have multiple configurations and indexes on the same instance. This is specially useful when you want other applications than CKAN or different CKAN versions to use the same Solr instance. The different cores will have different paths in the Solr server URL:

```
http://localhost:8983/solr/ckan-schema-1.2       # Used by CKAN up to 1.5
http://localhost:8983/solr/ckan-schema-1.3       # Used by CKAN 1.5.1
http://localhost:8983/solr/ckan-schema-1.4       # Used by CKAN 1.7
http://localhost:8983/solr/some-other-site  # Used by another site
```

To set up a multicore Solr instance, repeat the steps on the previous section to configure a single Solr instance.

Create a *solr.xml* file in */usr/share/solr*. This file will list the different cores, and allows also to define some configuration options. This is how cores are defined:

```
<solr persistent="true" sharedLib="lib">
    <cores adminPath="/admin/cores">
        <core name="ckan-schema-1.2" instanceDir="core0">
            <property name="dataDir" value="/var/lib/solr/data/core0" />
        </core>
        <core name="ckan-schema-1.3" instanceDir="core1">
            <property name="dataDir" value="/var/lib/solr/data/core1" />
        </core>
    </cores>
</solr>
```

Adjust the names to match the CKAN schema versions you want to run.

Note that each core is configured with its own data directory. This is really important to prevent conflicts between cores. Now create them like this:

```
sudo -u jetty mkdir /var/lib/solr/data/core0
sudo -u jetty mkdir /var/lib/solr/data/core1
```

For each core, we will create a folder in */usr/share/solr*, with a symbolic link to a specific configuration folder in */etc/solr/*. Copy the existing conf directory to the core directory and link it from the home dir like this:

```
sudo mkdir /etc/solr/core0
sudo mv /etc/solr/conf /etc/solr/core0/

sudo mkdir /usr/share/solr/core0
sudo ln -s /etc/solr/core0/conf /usr/share/solr/core0/conf
```

Now configure the core to use the data directory you have created. Edit */etc/solr/core0/conf/solrconfig.xml* and change the <*dataDir*> to this variable:

```
<dataDir>${dataDir}</dataDir>
```

This will ensure the core uses the data directory specified earlier in *solr.xml*.

Once you have your first core configured, to create new ones, you just need to add them to the *solr.xml* file and copy the existing configuration dir:

```
sudo mkdir /etc/solr/core1
sudo cp -R /etc/solr/core0/conf /etc/solr/core1

sudo mkdir /usr/share/solr/core1
sudo ln -s /etc/solr/core1/conf /usr/share/solr/core1/conf
```

Remember to ensure each core points to the correct CKAN schema. To change core1 to be ckan-schema-1.3:

```
sudo rm /etc/solr/core1/conf/schema.xml
sudo ln -s <full-path>/schema-1.3.xml /etc/solr/core1/conf/schema.xml
```

(where `<full-path>` is the full path to the schema file on your machine)

Now restart jetty:

```
sudo /etc/init.d/jetty stop
sudo /etc/init.d/jetty start
```

And check that Solr is listing all the cores when browsing http://localhost:8983/solr/

1.5.3 Troubleshooting

Solr requests and errors are logged in the web server log.

- For jetty servers, they are located in:

  ```
  /var/log/jetty/<date>.stderrout.log
  ```

- For Tomcat servers, they are located in:

  ```
  /var/log/tomcat6/catalina.<date>.log
  ```

Some problems that can be found during the install:

- When setting up a multi-core Solr instance, no cores are shown when visiting the Solr index page, and the admin interface returns a 404 error.

  Check the web server error log if you can find an error similar to this one:

  ```
  WARNING: [iatiregistry.org] Solr index directory '/usr/share/solr/iatiregistry.org/data/index' d
  07-Dec-2011 18:06:33 org.apache.solr.common.SolrException log
  SEVERE: java.lang.RuntimeException: Cannot create directory: /usr/share/solr/iatiregistry.org/da
        [...]
  ```

  The dataDir is not properly configured. With our setup the data directory should be under */var/lib/solr/data*. Make sure that you defined the correct *dataDir* in the *solr.xml* file and that in the *solrconfig.xml* file you have the following configuration option:

  ```
  <dataDir>${dataDir}</dataDir>
  ```

1.5.4 Handling changes in the CKAN schema

At some point, changes in new CKAN versions will mean modifications in the schema to support new features or fix defects. These changes won't be always backwards compatible, so some changes in the Solr servers will need to be performed.

If a CKAN instance is using a Solr server for itself, the schema can just be updated on the Solr server and the index rebuilt. But if a Solr server is shared between different CKAN instances, there may be conflicts if the schema is updated.

CKAN uses the following conventions for supporting different schemas:

- If needed, create a new schema file when releasing a new version of CKAN (i.e if there are two or more different modifications in the schema file between CKAN releases, only one new schema file is created).
- Keep different versions of the Solr schema in the CKAN source, with a naming convention, *schema-<version>.xml*:

  ```
  ckan/config/solr/schema-1.2.xml
  ckan/config/solr/schema-1.3.xml
  ```

- Each new version of the schema file must include its version in the main <*schema*> tag:

  ```
  <schema name="ckan" version="1.3">
  ```

- Solr servers used by more than one CKAN instance should be configured as multiple cores, and provide a core for each schema version needed. The cores should be named following the convention *schema-<version>*, e.g.:

  ```
  http://<solr-server>/solr/ckan-schema-1.2/
  http://<solr-server>/solr/ckan-schema-1.3/
  ```

When a new version of the schema becomes available, a new core is created, with a link to the latest schema.xml file in the CKAN source. That way, CKAN instances that use an older version of the schema can still point to the core that uses it, while more recent versions can point to the latest one. When old versions of CKAN are updated, they only need to change their *solr_url* setting to point to the suitable Solr core.

CHAPTER 2

Customizing and Extending

2.1 Theming and Customizing Appearance

After installing CKAN, the next step is probably to re-theme the site with your own logo, site name, and CSS.

2.1.1 Site Name and Description

You can change the name and logo of the site by setting options in the CKAN config file.

This is the file called `std.ini` that you first encountered in *Create an Admin User*. It is usually located at `/etc/ckan/std/std.ini`.

Open this file, and change the following options:

```
ckan.site_title = My CKAN Site
ckan.site_description = The easy way to get, use and share data
```

After you've edited these options, restart Apache:

```
sudo /etc/init.d/apache2 restart
```

Refresh your home page (clearing the cache if necessary) and you should see your new title and description.

2.1.2 Adding CSS, Javascript and other HTML using Config Options

CKAN provides two config options that allow you to add material directly into templates:

```
ckan.template_head_end = ...
ckan.template_footer_end = ...
```

The first of these allows you to specify content that will be inserted just before `</head>` tag while the second allows you to insert content just before the `</body>`. You can use html in both of these as well as provide multiline values by indenting lines.

Adding CSS

For example, by referencing an external CSS stylesheet:

```
ckan.template_head_end = <link rel="stylesheet" href="http://some-other-site.org/custom.css" type="te
```

Note: This requires you have a css file uploaded elsewhere. You may want your css to be part of your CKAN site. CKAN provides an easy way to do this – see `extra_public_paths` below.

Alternatively you can provide CSS directly in a style tag (note how we indent lines to provide a multiline value to a config option):

```
ckan.template_head_end = <style type="text/css">
    body {
      font-size: xlarge;
    }
  </style>
```

Adding Javascript

You could also use this config option to add script tags (or any other material to the `<head>` of all site pages).

However, for javascript it is probably better to use the `ckan.template_footer_end` option as it adds material just before the closing `</body>` tag – in CKAN v1.5 scripts are included at the foot of the page rather than in the <head> section (thus if your scripts requires jquery it needs to come after jqurey is included at the bottom of the page and hence you should use the footer end option).

2.1.3 More Advanced Customization

If you want to make broader changes to the look and feel of your CKAN site, we offer ways to add custom CSS and over-ride the default CKAN templates.

Adding (and Overriding) Files and Templates

You can add (and override) files (e.g. CSS, scripts and images) as well as templates to your site using the `extra_template_paths` and `extra_public_paths` options in the CKAN config file:

```
extra_template_paths = %(here)s/my-templates
extra_public_paths = %(here)s/my-public
```

All contents of the public directory is mounted directly into the URL space of the site (taking precedence over existing files of the same name).

Furthermore, you can supply multiple public directories, which will be searched in order.

For example, if you set the following option in the CKAN config file:

```
extra_public_paths = /path/to/mypublicdir
```

And then add a file called `myhtmlfile.html` in `/path/to/mypublicdir`, the file would appear on http://yourckan.org/ at `http://yourckan.org/myhtmlfile.html`.

If you create a file with the same path as one in the main CKAN public directory, your file will override the default CKAN file. For example, if you add `mypublicdir/css/ckan.css`, then `http://yourckan.org/css/ckan.css` will be your file.

Adding a New Logo

One example is introducing your own logo, which you can do with a new file and a CKAN config option.

Add a logo file at `mypublicdir/images/mylogo.png`, then set options in the CKAN config file (`/etc/ckan/std/std.ini`) as follows:

```
extra_public_paths = /path/to/mypublicdir
ckan.site_logo = /images/mylogo.png
```

Adding a New Stylesheet

Lots of visual changes can be made simply by changing the stylesheet. We've already

The easiest way to override the default CKAN style is to create one or more custom CSS files and load them in the `layout.html` template.

Use the 'public' directory as described in the previous section, then add a new file at `mypublicdir/css/mycss.css`.

Your next step is to have that css file including by the templates.

Next, copy the `layout.html` template and add a reference to the new CSS file. Here is an example of the edited `layout.html` template:

```
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:i18n="http://genshi.edgewall.org/i18n"
  xmlns:py="http://genshi.edgewall.org/"
  xmlns:xi="http://www.w3.org/2001/XInclude"
  py:strip="">
  <head py:match="head">
    ${select('*')}
    <link rel="stylesheet" href="${h.url_for('/css/mycss.css')}" />
  </head>
  <xi:include href="layout_base.html" />
</html>
```

Retheming the Site with Templates

Template files are used as source templates for rendered pages on the site. These templates are just an HTML page but with variables, such as the page title set by each page: `${page_title}`.

To over-ride a template, set the `extra_template_paths` directory as described above, then copy and rewrite the template file you wish to over-ride.

Commonly modified templates are:

- `layout.html` - empty by default
- `home/index.html` - the home page of the site
- `home/about.html` - the about page

If you are re-theming the site, we recommend you over-ride `layout.html`, which is empty but inherits from `layout_base.html`. This will mean you can upgrade the site more easily in the future.

Note: For more information on the syntax of the CKAN templates, refer to the Genshi documentation.

2.2 Comments and Commenting

In a default CKAN install commenting is disabled. To enable it you have to install and enable the disqus (commenting) extension:

https://github.com/okfn/ckanext-disqus/

Please read and follow the instructions there (as well as the standard *Add Extensions* documentation).

2.2.1 Important Note

Once installed and enabled, the presence of comments on a given page is configured by your theme (see the documentation in the disqus extension for details).

In the default CKAN theme (as of v1.7), comments will be shown only on dataset and resource pages (and recent comments only on the *CKAN Administrative Dashboard*).

2.3 Add Extensions

This is where it gets interesting! The CKAN software can be customised with 'extensions'. These are a simple way to extend core CKAN functions.

Extensions allow you to customise CKAN for your own requirements, without interfering with the basic CKAN system.

Warning: This is an advanced topic.

2.3.1 Choosing Extensions

All CKAN extensions are listed on the official Extension listing on the CKAN wiki.

Some popular extensions include:

Note: Those marked with (x) are 'core' extensions and are shipped as part of the core CKAN distribution

- ckanext-stats (x): Statistics (and visuals) about the datasets in a CKAN instance.
- ckanext-apps: Apps and ideas catalogue extension for CKAN.
- ckanext-disqus: Allows users to comment on dataset pages with Disqus.
- ckanext-follower: Allow users to follow datasets.
- ckanext-googleanalytics: Integrates Google Analytics data into CKAN. Gives download stats on dataset pages, list * of most popular datasets, etc.
- ckanext-qa: Provides link checker, 5 stars of openness and other Quality Assurance features.
- ckanext-rdf: Consolidated handling of RDF export and import for CKAN.
- ckanext-wordpresser: CKAN plugin / WSGI middleware for combining CKAN with a Wordpress site.
- ckanext-spatial: Adds geospatial capabilities to CKAN datasets, including a spatial search API.

2.3.2 Installing an Extension

You can install an extension on a CKAN instance as follows.

Note: 'Core' extensions do not need to be installed – just enabled (see below).

1. Locate your CKAN virtual environment (pyenv) in your filesystem. It is usually in a directory similar to this: `/var/lib/ckan/INSTANCE_NAME/pyenv`

If it is not here, to get the definitive answer, check your CKAN Apache configuration (`/etc/apache2/sites-enabled`) for your WSGIScriptAlias (e.g. `/var/lib/ckan/colorado/wsgi.py`) which has an `execfile` instruction. The first parameter is the pyenv directory plus `/bin/activate_this.py`. e.g. `/var/lib/ckan/colorado/pyenv/bin/activate_this.py` means the pyenv dir is: `/var/lib/ckan/colorado/pyenv`.

1. Install the extension package code into your pyenv using `pip`.

   For example, to install the Disqus extension, which allows users to comment on datasets (replacing "INSTANCE_NAME" with the name of your CKAN instance):

   ```
   sudo -u ckanINSTANCE_NAME /var/lib/ckan/INSTANCE_NAME/pyenv/bin/pip install -E /var/lib/ckan/INS
   ```

   Prefix the source URL with the repo type (`hg+` for Mercurial, `git+` for Git).

   The dependency you've installed will appear in the `src/` directory under your Python environment.

Now the extension is installed, so now you can enable it.

2.3.3 Enabling an Extension

1. Add the names of the extension's plugins to the CKAN config file in the '[app:main]' section under 'ckan.plugins'. e.g.:

   ```
   [app:main]
   ckan.plugins = disqus
   ```

   If your extension implements multiple different plugin interfaces, separate them with spaces:

   ```
   ckan.plugins = disqus amqp myplugin
   ```

   Note: Finding out the name of an extension's plugins: this information should usually be provided in the extension's documentation, but you can also find this information in the plugin's `setup.py` file under `[ckan.plugins]`.

2. To have this configuration change take effect it may be necessary to restart WSGI, which usually means restarting Apache:

   ```
   sudo /etc/init.d/apache2 restart
   ```

Your extension should now be enabled. You can disable it at any time by removing it from the list of ckan.plugins in the config file.

Enabling an Extension with Background Tasks

Some extensions need to run tasks in the background. See *Background Tasks* for how to enable background tasks.

2.4 Understand and Write Extensions

If you want to extend CKAN core functionality, the best way to do so is by writing extensions.

Extensions allow you to customise CKAN for your own requirements, without interfering with the basic CKAN system.

To meet the need to customize CKAN efficiently, we have introduced the concepts of CKAN extensions and plugin interfaces. These work together to provide a simple mechanism to extend core CKAN functionality.

Warning: This is an advanced topic. We are working to make the most popular extensions more easily available as Debian packages.

Note: The terms **extension** and **plugin interface** have very precise meanings: the use of the generic word **plugin** to describe any way in which CKAN might be extended is deprecated.

Contents

2.4.1 CKAN Extensions

Extensions are implemented as *namespace packages* under the `ckanext` package which means that they can be imported like this:

```
$ python
>>> import ckanext.example
```

Individual CKAN *extensions* may implement one or more *plugin interfaces* to provide their functionality.

Creating CKAN Extensions

All CKAN extensions must start with the name `ckanext-`. You can create your own CKAN extension like this:

```
(pyenv)$ paster create -t ckanext ckanext-myextension
```

You'll get prompted to complete a number of variables which will be used in your dataset. You change these later by editing the generated `setup.py` file. Here's some example output:

```
Selected and implied templates:
  ckan#ckanext  CKAN extension project template

Variables:
  egg:      ckanext_myextension
  package:  ckanextmyextension
  project:  ckanext-myextension
Enter version (Version (like 0.1)) ['']: 0.4
Enter description (One-line description of the package) ['']: Great extension package
Enter author (Author name) ['']: James Gardner
Enter author_email (Author email) ['']: james.gardner@okfn.org
Enter url (URL of homepage) ['']: http://jimmyg.org
Enter license_name (License name) ['']: GPL
Creating template ckanext
Creating directory ./ckanext-myextension
  Directory ./ckanext-myextension exists
  Skipping hidden file pyenv/src/ckan/ckan/pastertemplates/template/.setup.py_tmpl.swp
  Recursing into ckanext
    Creating ./ckanext-myextension/ckanext/
    .svn/ does not exist; cannot add directory
    Recursing into +project+
      Creating ./ckanext-myextension/ckanext/myextension/
      .svn/ does not exist; cannot add directory
      Copying __init__.py to ./ckanext-myextension/ckanext/myextension/__init__.py
      .svn/ does not exist; cannot add file
    Copying __init__.py to ./ckanext-myextension/ckanext/__init__.py
    .svn/ does not exist; cannot add file
  Copying setup.py_tmpl to ./ckanext-myextension/setup.py
  .svn/ does not exist; cannot add file
Running pyenv/bin/python setup.py egg_info
```

Once you've run this, you should now install the extension in your virtual environment:

```
(pyenv)$ cd ckanext-myextension
(pyenv)$ python setup.py develop
(pyenv)$ python
Python 2.6.6 (r266:84292, Oct  6 2010, 16:19:55)
[GCC 4.1.2 20080704 (Red Hat 4.1.2-48)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import ckanext.myextension
>>>
```

Note: Running `python setup.py develop` will add a `.egg-link` file to your python site-packages directory (which is on your python path). This allows your extension to be imported and used, with any changes made to the extension source code showing up immediately without needing to be reinstalled, which is very useful during development.

To instead install a python package by copying all of the files to the site-packages directory run `python setup.py install`.

To build useful extensions you need to be able to "hook into" different parts of CKAN in order to extend its functionality. You do this using CKAN's plugin architecture. We'll look at this in the next section.

2.4.2 Plugins: An Overview

Plugin interfaces provide a specification which extensions can implement in order to "hook into" core CKAN functionality.

The CKAN plugin implementation is based on the PyUtilib component architecture (PCA). Here's a quick summary, we'll go through all this in much more detail in a minute:

1. The CKAN core contains various *plugin interfaces*, each specifying a set of methods where plugins may hook into the software. For example a plugin wanting to hook into the SQLAlchemy mapping layer would need to implement the `IMapperExtension` interface.
2. A plugin is a class that derives from `ckan.plugins.Plugin` or more commonly `SingletonPlugin`. It must also implement one of the plugin interfaces exposed in `ckan.plugins.interfaces`. The choice interface determines the functionality the plugin is expected to provide.
3. Plugin objects must be registered as setuptools entry points. The `ckan.plugins` configuration directive is searched for names of plugin entry points to load and activate.

Here's a list of some of the more commonly used plugin interfaces:

`IDatasetForm` Provide a custom dataset form and schema.

`IMapper` Listens and react to every database change.

`IRoutes` and `IController` Provide an implementation to handle a particular URL.

`IGenshiStreamFilter` Intercept template rendering to modify the output.

`IDomainObjectModification` Listens for changes to CKAN domain objects.

`IGroupController` Plugins for in the groups controller. These will usually be called just before committing or returning the respective object, i.e. all validation, synchronization and authorization setup are complete.

`IConfigurable` Pass configuration to plugins and extensions.

`IAuthorizer` Allows customisation of the default Authorization behaviour.

If you look in ckan/plugins/interfaces.py you will see the latest plugin interfaces. Alternatively see the Plugin API documentation below.

2.4.3 Example CKAN Extension

A example CKAN extension can be found at http://github.com/okfn/ckanext-example. Have a look at the README file for installation instructions.

2.4.4 Publishing Extensions

At this point you might want to share your extension with the public.

First check you have chosen an open source licence (e.g. the MIT licence) and then update the `long_description` variable in `setup.py` to explain what the extension does and which entry point names a user of the extension will need to add to their `ckan.plugins` configuration.

Once you are happy, run the following commands to register your extension on the Python Package Index:

```
python setup.py register
python setup.py sdist upload
```

You'll then see your extension at http://pypi.python.org/pypi. Others will be able to install your plugin with `pip`.

Finally, please also add a summary of your extension and its entry points to the Extensions page at http://wiki.ckan.org/Extensions.

2.4.5 Writing a Plugin Interface

This describes how to add a plugin interface to make core CKAN code pluggable.

Suppose you have a class such as this:

```
class DataInput(object):

    def accept_new_data(self, data):
        self.data = data
```

And you want plugins to hook into `accept_new_data` to modify the data.

You would start by declaring an interface specifying the methods that plugin classes must provide. You would add the following code to `ckan/plugins/interfaces.py`:

```
class IDataMunger(Interface):

    def munge(self, data):
        return data
```

Now you can tell this class that its plugins are anything that implements `IDataMunger` like this:

```
from ckan.plugins import PluginImplementations, IDataMunger

class DataInput(object):

    plugins = PluginImplementations(IDataMunger)

    def accept_new_data(self, data):
       for plugin in self.plugins:
           data = plugin.munge(data)
       self.data = data
```

Any registered plugins that implement `IDataMunger` will then be available in your class via `self.plugin`.

See the pyutilib documentation for more information on creating interfaces and plugins. However, be aware that pyutilib uses slightly different terminology. It calls `PluginImplementations` `ExtensionPoint` and it calls instances of a plugin object a *service*.

2.4.6 Testing

Testing CKAN Extensions

CKAN extensions ordinarily have their own `test.ini` that refers to the CKAN `test.ini`, so you can run them in exactly the same way. For example:

```
cd ckanext-dgu
nosetests ckanext/dgu/tests --ckan
nosetests ckanext/dgu/tests --ckan --with-pylons=test-core.ini
```

To test your changes you'll need to use the `paster serve` command from the `ckan` directory:

```
paster serve --reload -c <path to your CKAN config file>
```

You should also make sure that your CKAN installation passes the developer tests, as described in *Testing for Developers*.

Testing Plugins

When writing tests for your plugin code you will need setup and teardown code similar to the following to ensure that your plugin is loaded while testing:

```
from ckan import plugins

class TestMyPlugin(TestCase):

    @classmethod
    def setup_class(cls):
        # Use the entry point name of your plugin as declared
        # in your package's setup.py
        plugins.load('my_plugin')

    @classmethod
    def teardown_class(cls):
        plugins.reset()
```

The exception to using `plugins.load()` is for when your plug-in is for routes. In this case, the plugin must be configured before the WSGI app is started. Here is an example test set-up:

```
from paste.deploy import appconfig
import paste.fixture
from ckan.config.middleware import make_app
from ckan.tests import conf_dir

class TestMyRoutesPlugin(TestCase):

     @classmethod
     def setup_class(cls):
          config = appconfig('config:test.ini', relative_to=conf_dir)
          config.local_conf['ckan.plugins'] = 'my_routes_plugin'
          wsgiapp = make_app(config.global_conf, **config.local_conf)
          cls.app = paste.fixture.TestApp(wsgiapp)
```

At this point you should be able to write your own plugins and extensions together with their tests.

2.4.7 Ordering of Extensions

> **Caution:** The order in which extensions are initially loaded is **different** to the order that their plugins are run.

The order in which extensions are initially loaded is as follows:

1. System plugins (in setup.py under `ckan.system_plugins`).
2. In order of the plugins specified in the config file: `plugins =`.
3. If more than one module has a plugin with the same name specified in the config, then all those are loaded, in the order the modules appear in `sys.path`.

The order that a plugins are run in, for example the order that IRoutes extensions have their `before_map` method run, is alphabetical by the plugin class.

e.g. here is the order for these four extensions: `<Plugin DguInventoryPlugin>`, `<Plugin FormApiPlugin>`, `<Plugin StatsPlugin>`, `<Plugin WalesThemePlugin>`

(This alphabetical ordering is done by `pyutilib.component.core:ExtensionPoint.extensions()`)

Plugin API Documentation

2.5 Customizing Forms

The forms used to edit datasets and groups in CKAN can be customized. This lets you tailor them to your needs, helping your users choose from sensible options or use different data formats. This document explains how to customize the dataset and group forms you offer to your users.

> **Warning:** This is an advanced topic. Ensure you are familiar with *Add Extensions* before attempting to customize forms.

Note: This document describes the process for creating dataset forms that is used in the `ckanext-example` extension. The source code is available at http://github.com/okfn/ckanext-example. Group forms can be customised in a similar way, see `ckanext-example` again for reference.

2.5.1 Creating a Dataset Form

The recommended way to create a dataset form is by using a CKAN extension.

First, create a new plugin that implements `ckan.plugins.IDatasetForm`. It is also useful to implement `ckan.plugins.IConfigurer` as well, so that you can add the directory that will contain your new dataset form template to CKAN's list of template paths.

```
import ckan.plugins

class ExampleDatasetForm(ckan.plugins.SingletonPlugin):
    implements(ckan.plugins.IDatasetForm, inherit=True)
    implements(ckan.plugins.IConfigurer, inherit=True)
```

Next, create a directory in your CKAN extension to store your HTML template(s) and add an `update_config` method to make sure that this template is on CKAN's template path. Here we add the `ckanext/example/theme/templates` directory to the path.

```
def update_config(self, config):
    here = os.path.dirname(__file__)
    rootdir = os.path.dirname(os.path.dirname(here))
    template_dir = os.path.join(rootdir, 'ckanext', 'example', 'theme', 'templates')
    config['extra_template_paths'] = ','.join([
        template_dir, config.get('extra_template_paths', '')
    ])
```

You can now add your HTML template(s) to the templates directory. It is recommended that you copy the existing CKAN dataset form (`ckan/templates/package/new_package_form.html`) and make modifications to it. However, it is possible to start from scratch.

If you create a template in your extension templates directory at `package/new_package_form.html`, it will override the default CKAN template. This applies to any template in the CKAN templates directory. You can also override them by changing the paths returned by the following methods in your extension:

```
def package_form(self):
    return 'package/new_package_form.html'

def new_template(self):
    return 'package/new.html'
```

```
def comments_template(self):
    return 'package/comments.html'

def search_template(self):
    return 'package/search.html'

def read_template(self):
    return 'package/read.html'

def history_template(self):
    return 'package/history.html'
```

Note: The `package_form` and `*_template` methods (above) are required in order to implement IDatasetForm.

Two other methods must be provided by your extension when implementing the IDatasetForm interface:

```
def package_types(self):
    return ['dataset']

def is_fallback(self):
    return True
```

`package_types` sets the dataset type associated with your extension, and updates the Pylons routing so that datasets of this type can be found at the `/<type>` URL in your CKAN instance.

For example, changing `package_type` to return `['catalog']` would mean that any visits to `/catalog/new`, `/catalog/edit`, etc. would use your extension's dataset form, but going to `/dataset/new`, `/dataset/edit`, etc. would still return CKAN's default dataset form.

`is_fallback` means that this extension should be the default dataset type. If `True`, even when the return value of `package_types` is changed, going to `/dataset/new` will still use the extension's dataset form instead of CKAN's default.

2.5.2 Passing Data to Templates

Your IDatasetForm extension can define a `_setup_template_variables` method, and use it to add data to the Pylons `c` object (which is passed to the templates).

For example, you can define `_setup_template_variables` as follows:

```
def setup_template_variables(self, context, data_dict=None, package_type=None):
    from ckan.lib.base import c
    from ckan import model
    c.licences = model.Package.get_license_options()
```

and then use it in your HTML template:

```
<dd class="license-field">
  <select id="license_id" name="license_id">
    <py:for each="licence_desc, licence_id in c.licences">
      <option value="${licence_id}">${licence_desc}</option>
    </py:for>
  </select>
</dd>
```

2.5.3 Custom Schemas

Note: As of CKAN 1.7 custom schema functions apply to both the web user interface and the API.

An example of the use of these methods can be found in the `ckanext-example` extension.

The data fields that are accepted and returned by CKAN for each dataset can be changed by an IDatasetForm extension by overriding the following methods:

```
def form_to_db_schema_options(self, options)
```

This allows us to select different schemas for different purpose eg via the web interface or via the api or creation vs updating. It is optional and if not available form_to_db_schema should be used.

```
_form_to_db_schema(self)
```

This defines a navl schema to customize validation and conversion to the database.

```
_db_to_form_schema(self)
```

This defines a navl schema to customize conversion from the database to the form.

```
_db_to_form_schema_options(self, options)
```

Like `_form_to_db_schema_options()`, this allows different schemas to be used for different purposes. It is optional, and if it is not available then `form_to_db_schema` is used.

2.5.4 Example: Geospatial Tags

In this example we look at how create a plugin that adds a new dataset field called `geographical_coverage`. This field allows the user to specify one or more country-code tags in order to indicate which countries are covered by the dataset. Additionally, the tags must be part of a fixed CKAN tag vocabularly called `country_codes`.

More information about tag vocabularies can be found in *Tag vocabularies*.

1. Creating the Tag Vocabulary

First we are going to create the `country_codes` vocabulary and add a few tags to it. The following code can be saved to a python script and then run from the command line.

```
import json
import requests

ckan_url = 'http://127.0.0.1:5000'
api_key = 'xxxx'

geo_tags = [u'uk', u'ie', u'de', u'fr', u'es']
headers = {'Authorization': api_key}
```

We are going to use the requests module (tested with version 0.10.7, available on PyPI) to make our POST requests.

Replace the values of `ckan_url` and `api_key` with the URL to your CKAN instance and your API key respectively. You must be a system administrator in order to create tag vocabularies.

We also define the 5 tags that we will add to the vocabulary here, and set the `Authorization` header to the value of our API key.

```
# create the vocabulary
data = json.dumps({'name': u'country_codes'})
r = requests.post(ckan_url + '/api/action/vocabulary_create',
                  data=data,
                  headers=headers)
vocab_id = json.loads(r.text)['result']['id']
```

This creates our `country_codes` vocabulary, and saves a reference to the vocabulary ID that is returned by CKAN.

```
# add tags
for geo_tag in geo_tags:
    data = json.dumps({'name': geo_tag, 'vocabulary_id': vocab_id})
    r = requests.post(ckan_url + '/api/action/tag_create',
                      data=data,
                      headers=headers)
```

We then add each of our tags, making sure to set their vocabulary ID.

2. Creating the Plugin

First we create a CKAN plugin that implements IDatasetForm:

```
import ckan.plugins

class GeospatialTagDatasetForm(SingletonPlugin):
    implements(IDatasetForm, inherit=True)

    def package_form(self):
        return 'package/new_package_form.html'

    def new_template(self):
        return 'package/new.html'

    def comments_template(self):
        return 'package/comments.html'

    def search_template(self):
        return 'package/search.html'

    def read_template(self):
        return 'package/read.html'

    def history_template(self):
        return 'package/history.html'

    def is_fallback(self):
        return True

    def package_types(self):
        return ['dataset']
```

We want to pass the list of country code tags through to our dataset form, so we define a `setup_template_variables` function which stores the tags as a `geographical_coverage` against the Pylons `c` object.

```
def setup_template_variables(self, context, data_dict=None, package_type=None):
    try:
        data = {'vocabulary_id': u'country_codes'}
```

```
            c.geographical_coverage = get_action('tag_list')(context, data)
        except NotFound:
            c.geographical_coverage = []
```

Finally we have to update our dataset schema so that we can store the country code data.

```
def form_to_db_schema(self, package_type=None):
    from ckan.logic.schema import package_form_schema
    from ckan.lib.navl.validators import ignore_missing
    from ckan.logic.converters import convert_to_tags

    schema = package_form_schema()
    schema.update({
        'geographical_coverage': [ignore_missing, convert_to_tags('country_codes')]
    })
    return schema

def db_to_form_schema(data, package_type=None):
    from ckan.logic.converters import convert_from_tags, free_tags_only
    from ckan.lib.navl.validators import ignore_missing, keep_extras

    schema = package_form_schema()
    schema.update({
        'tags': {
            '__extras': [keep_extras, free_tags_only]
        },
        'geographical_coverage': [convert_from_tags('country_codes'), ignore_missing],
    })
    return schema
```

Here were use the `convert_to_tags` and `convert_from_tags` converters, so that our country codes are stored as normal CKAN tags. We also apply the `free_tags_only` converter to the `tags` field when displaying datasets in order to remove our geospatial tags from this list and display them separately.

3. Updating the Template

Note: To edit fixed tag vocabulary fields, we recommend using a HTML multiple select tag together with the JQuery *Chosen* plugin (included in CKAN core).

You must add a new field to your dataset form in order to display (and edit) the new geographical coverage tags. The following HTML segment creates a multiple select element to display the tags, marking any tags that are currently saved as 'selected'.

```
<select id="geographical_coverage" class="chzn-select"
        name="geographical_coverage" multiple="multiple">
  <py:for each="tag in c.geographical_coverage">
    <py:choose test="">
    <option py:when="tag in data.get('geographical_coverage', [])"
            selected="selected" value="${tag}">${tag}</option>
    <option py:otherwise="" value="${tag}">${tag}</option>
    </py:choose>
  </py:for>
</select>
```

2.6 Tag vocabularies

New in version 1.7.

CKAN tags can belong to a vocabulary, which is a way of grouping related tags together.

2.6.1 Properties

- A CKAN instance can have any number of vocabularies.
- Each vocabulary consists of an ID, name and description.
- Each tag can be assigned to a single vocabulary (or have no vocabulary).
- A dataset can have more than one tag from the same vocabulary, and can have tags from more than one vocabulary.
- **Vocabularies can be of two types:**
  - Controlled: the list of possible tags is pre-defined
  - Free: users enter their own terms

2.6.2 Using Vocabularies

A CKAN developer/sysadmin user will have to do a number of things to add some custom vocabs to their CKAN instance:

1. Call CKAN API functions to add the vocabularies and terms to the db.
2. Implement a CKAN extension (with a custom form), including the tag vocabularies in the schema.
3. Provide dataset view, edit and create templates with the new schemas in them.

1. Adding Vocabularies

This needs to be done via the action API (*Model, Search and Action API: Version 3*). Please check the examples section to see which calls are needed.

2. Custom Form Schema

- Make a plugin that implements `IDatasetForm` (for example, see `ExampleDatasetForm` in https://github.com/okfn/ckanext-example/blob/master/ckanext/example/forms.py).
- Override `form_to_db_schema`. Add a new field for your vocabulary tags, making sure that it uses the `convert_to_tags` converter with the name of the vocabulary. For example, the following will add a new field called `vocab_tags`, with each tag assigned to the vocabulary `EXAMPLE_VOCAB`:

```
def form_to_db_schema(self):
    schema = package_form_schema()
    schema.update({'vocab_tags': [ignore_missing, convert_to_tags('EXAMPLE_VOCAB')]})
    return schema
```

- Override `db_to_form_schema`. Add your new field to the schema, making sure that it uses the `convert_from_tags` validator. If you don't want the tags with vocabularies to be listed along with normal tags (on the web page or via API calls), then make sure that the normal tags field has the `free_tags_only` converter applied. For example:

```
def db_to_form_schema(self):
    schema = package_form_schema()
    schema.update({
        'tags': {'__extras': [keep_extras, free_tags_only]},
        'vocab_tags': [convert_from_tags('EXAMPLE_VOCAB'), ignore_missing]
    })
    return schema
```

3. Adding To Templates

- If the vocabulary is restricted, you may want to pass a list of all tags in the vocabulary to the template so that they can be displayed as options in a select (or multi-select) box. This should be done by overriding the `setup_template_variables` method in your class that implements `IDatasetForm`. For example:

  ```
  def setup_template_variables(self, context, data_dict=None):
      c.vocab_tags = get_action('tag_list')(context, {'vocabulary_id': 'EXAMPLE_VOCAB'})
  ```

- The custom tags must be added to the template in order to be displayed or edited. For fixed vocabularies, we recommend adding them as a multi-select tag and using the JQuery `Chosen` plugin (included in CKAN core).

2.6.3 Examples

- An example of how to implement it can be found in the section *Customizing Forms Example: Geospatial Tags*.
- A complete, working example of a custom form that uses tag vocabularies can be found in the CKAN Example extension (http://github.com/okfn/ckanext-example).

2.7 Form Integration

CKAN allows you to integrate its Edit Dataset and New Dataset forms forms into an external front-end. To that end, CKAN also provides a simple way to redirect these forms back to the external front-end upon submission.

2.7.1 Redirecting CKAN Forms

It is obviously simple enough for an external front-end to link to CKAN's Edit Dataset and New Dataset forms, but once the forms are submitted, it would be desirable to redirect the user back to the external front-end, rather than CKAN's dataset read page.

This is achieved with a parameter to the CKAN URL. The 'return URL' can be specified in two places:

1. Passed as a URL-encoded value with the parameter `return_to` in the link to CKAN's form page.
2. Specified in the CKAN config keys `package_new_return_url` and `package_edit_return_url` (see *package_new_return_url & package_edit_return_url*).

(If the 'return URL' is supplied in both places, then the first takes precedence.)

Since the 'return URL' may need to include the dataset name, which could be changed by the user, CKAN replaces a known placeholder `<NAME>` with this value on redirect.

Note: Note that the downside of specifying the 'return URL' in the CKAN config is that the CKAN web interface becomes less usable on its own, since the user is hampered by the redirects to the external interface.

Example

An external front-end displays a dataset 'ontariolandcoverv100' here:

```
http://datadotgc.ca/dataset/ontariolandcoverv100
```

It displays a link to edit this dataset using CKAN's form, which without the redirect would be:

```
http://ca.ckan.net/dataset/edit/ontariolandoverv100
```

At first, it may seem that the return link should be `http://datadotgc.ca/dataset/ontariolandcoverv100`. But when the user edits this dataset, the name may change. So the return link needs to be:

```
http://datadotgc.ca/dataset/<NAME>
```

And this is URL-encoded to become:

```
http%3A%2F%2Fdatadotgc.ca%2Fdataset%2F%3CNAME%3E
```

So, in summary, the edit link becomes:

```
http://ca.ckan.net/dataset/edit/ontariolandoverv100?return_to=http%3A%2F%2Fdatadotgc.ca%2Fdataset%2F%
```

During editing the dataset, the user changes the dataset name to *canadalandcover*, presses 'preview' and finally 'commit'. The user is now redirected back to the external front-end at:

```
http://datadotgc.ca/dataset/canadalandcover
```

The same functionality could be achieved by this line in the config file (`ca.ckan.net.ini`):

```
...

[app:main]
package_edit_return_url = http://datadotgc.ca/dataset/<NAME>

...
```

2.8 Linked Data and RDF

CKAN has extensive support for linked data and RDF. In particular, there is complete and functional mapping of the CKAN dataset schema to linked data formats.

2.8.1 Enabling and Configuring Linked Data Support

In CKAN <= 1.6 please install the RDF extension: https://github.com/okfn/ckanext-rdf

In CKAN >= 1.7, basic RDF support will be available directly in core.

Configuration

When using the built-in RDF support (CKAN >= 1.7) there is no configuration required. By default requests for RDF data will return the RDF generated from the built-in 'packages/read.rdf' template, which can be overridden using the extra-templates directive.

2.8.2 Accessing Linked Data

To access linked data versions just access the *CKAN API* in the usual way but set the Accept header to the format you would like to be returned. For example:

```
curl -L -H "Accept: application/rdf+xml" http://thedatahub.org/dataset/gold-prices
curl -L -H "Accept: text/n3" http://thedatahub.org/dataset/gold-prices
```

An alternative method of retrieving the data is to add .rdf to the name of the dataset to download:

```
curl -L http://thedatahub.org/dataset/gold-prices.rdf
curl -L http://thedatahub.org/dataset/gold-prices.n3
```

2.8.3 Schema Mapping

There are various vocabularies that can be used for describing datasets:

- Dublin core: these are the most well-known and basic. Dublin core terms includes the class *dct:Dataset*.
- DCAT - vocabulary for catalogues of datasets
- VoID - vocabulary of interlinked datasets. Specifically designed for describing *rdf* datasets. Perfect except for the fact that it is focused on RDF
- SCOVO: this is more oriented to statistical datasets but has a *scovo:Dataset* class.

At the present CKAN uses mostly DCAT and Dublin Core.

An example schema might look like:

```
<rdf:RDF xmlns:foaf="http://xmlns.com/foaf/0.1/" xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dcat="http://www.w3.org/ns/dcat#"
  xmlns:dct="http://purl.org/dc/terms/">
<dcat:Dataset rdf:about="http://127.0.0.1:5000/dataset/worldwide-shark-attacks">
  <owl:sameAs rdf:resource="urn:uuid:424bdc8c-038d-4b44-8f1d-01227e920b69"></owl:sameAs>
  <dct:description>Shark attacks worldwide</dct:description>
  <dcat:keyword>sharks</dcat:keyword>
  <dcat:keyword>worldwide</dcat:keyword>
  <foaf:homepage rdf:resource="http://127.0.0.1:5000/dataset/worldwide-shark-attacks"></foaf:homepage
  <rdfs:label>worldwide-shark-attacks</rdfs:label>
  <dct:identifier>worldwide-shark-attacks</dct:identifier>
  <dct:title>Worldwide Shark Attacks</dct:title>
  <dcat:distribution>
      <dcat:Distribution>
          <dcat:accessURL rdf:resource="https://api.scraperwiki.com/api/1.0/datastore/sqlite?format=
      </dcat:Distribution>
  </dcat:distribution>
  <dcat:distribution>
      <dcat:Distribution>
          <dcat:accessURL rdf:resource="https://api.scraperwiki.com/api/1.0/datastore/sqlite?format=
      </dcat:Distribution>
  </dcat:distribution>
  <dct:creator>
    <rdf:Description>
      <foaf:name>Ross</foaf:name>
      <foaf:mbox rdf:resource="mailto:ross.jones@okfn.org"></foaf:mbox>
    </rdf:Description>
  </dct:creator>
```

```
  <dct:contributor>
    <rdf:Description>
      <foaf:name>Ross</foaf:name>
      <foaf:mbox rdf:resource="mailto:ross.jones@okfn.org"></foaf:mbox>
    </rdf:Description>
  </dct:contributor>
  <dct:rights rdf:resource="http://www.opendefinition.org/licenses/odc-pddl"></dct:rights>
  </dcat:Dataset>
</rdf:RDF>
```

2.9 FileStore and File Uploads

CKAN allows users to upload files directly to file storage either on the local file system or to online 'cloud' storage like Amazon S3 or Google Storage. The uploaded files will be stored in the configured location.

2.9.1 Setup and Configuration

By default storage is disabled. To enable it, all you need to do is configure where files will be stored.

Common configuration options:

```
## Required
## 'Bucket' (subdirectory for file based storage) to use for file storage
ckan.storage.bucket = my-bucket-name

## Optional
## maximum content size for uploads in bytes, defaults to 1Gb
# ckanext.storage.max_content_length = 1000000000
```

Local File Storage

Important: you must install pairtree library for local storage to function:

```
pip install pairtree
```

For local file storage add to your config:

```
## OFS configuration
ofs.impl = pairtree
# directory on disk for data storage (should be empty)
ofs.storage_dir = /my/path/to/storage/root/directory
```

Cloud Storage

Important: you must install boto library for cloud storage to function:

```
pip install boto
```

In your config for google:

```
## OFS configuration
ofs.impl = google
ofs.gs_access_key_id = GOOG....
ofs.gs_secret_access_key = ....
```

For S3:

```
## OFS configuration
ofs.impl = s3
ofs.aws_access_key_id = ....
ofs.aws_secret_access_key = ....
```

2.9.2 Storage Web Interface

Upload of files to storage is integrated directly into the the Dataset creation and editing system with files being associated to Resources.

2.9.3 Storage API

Metadata API

The API is located at:

```
/api/storage/metadata/{label}
```

It supports the following methods:

- GET will return the metadata
- POST will add/update metadata
- PUT will replace metadata

Metadata is a json dict of key values which for POST and PUT should be send in body of request.

A standard response looks like:

```
{
  "_bucket": "ckannet-storage",
  _content_length: 1074
  _format: "text/plain"
  _label: "/file/8630a664-0ae4-485f-99c2-126dae95653a"
  _last_modified: "Fri, 29 Apr 2011 19:27:31 GMT"
  _location: "some-location"
  _owner: null
  uploaded-by: "bff737ef-b84c-4519-914c-b4285144d8e6"
}
```

Note that values with '_' are standard OFS metadata and are mostly read-only – _format i.e. content-type can be set).

Form Authentication

Provides credentials for doing operations on storage directly from a client (using web form style POSTs).

The API is located at:

```
/api/storage/auth/form/{label}
```

Provide fields for a form upload to storage including authentication:

```
:param label: label.
:return: json-encoded dictionary with action parameter and fields list.
```

Request Authentication API

Provides credentials for doing operations on storage directly from a client.

> **Warning:** this API is currently disabled and will likely be deprecated. Use the form authentication instead.

The API is at:

```
/api/storage/auth/request/{label}
```

Provide authentication information for a request so a client can interact with backend storage directly:

```
:param label: label.
:param kwargs: sent either via query string for GET or json-encoded
    dict for POST). Interpreted as http headers for request plus an
    (optional) method parameter (being the HTTP method).

    Examples of headers are:

        Content-Type
        Content-Encoding (optional)
        Content-Length
        Content-MD5
        Expect (should be '100-Continue')

:return: is a json hash containing various attributes including a
headers dictionary containing an Authorization field which is good for
15m.
```

2.9.4 DataStore Integration

It is also possible to have uploaded files (if of a suitable format) stored in the DataStore which will then provides an API to the data. See *DataStorer: Automatically Add Data to the DataStore* for more details.

2.10 DataStore and the Data API

The CKAN DataStore provides a database for structured storage of data together with a powerful Web-accesible Data API, all seamlessly integrated into the CKAN interface and authorization system.

2.10.1 Overview

The following short set of slides provide a brief overview and introduction to the DataStore and the Data API.

2.10.2 Relationship to FileStore

The DataStore is distinct but complementary to the FileStore (see *FileStore and File Uploads*). In contrast to the the FileStore which provides 'blob' storage of whole files with no way to access or query parts of that file, the DataStore is like a database in which individual data elements are accessible and queryable. To illustrate this distinction consider storing a spreadsheet file like a CSV or Excel document. In the FileStore this filed would be stored directly. To access it you would download the file as a whole. By contrast, if the spreadsheet data is stored in the DataStore one would be able to access individual spreadsheet rows via a simple web-api as well as being able to make queries over the spreadsheet contents.

2.10.3 The DataStore Data API

The DataStore's Data API, which derives from the underlying ElasticSearch data-table, is RESTful and JSON-based with extensive query capabilities.

Each resource in a CKAN instance has an associated DataStore 'table'. This table will be accessible via a web interface at:

```
/api/data/{resource-id}
```

This interface to this data is *exactly* the same as that provided by ElasticSearch to documents of a specific type in one of its indices.

For a detailed tutorial on using this API see *Using the Data API*.

2.10.4 Installation and Configuration

The DataStore uses ElasticSearch as the persistence and query layer with CKAN wrapping this with a thin authorization and authentication layer.

It also requires the use of Nginx as your webserver as its XSendfile feature is used to transparently hand off data requests to ElasticSeach internally.

1. Install ElasticSearch

Please see the ElasticSearch documentation.

2. Configure Nginx

You must add to your Nginx CKAN site entry the following:

```
location /elastic/ {
    internal;
    # location of elastic search
    proxy_pass http://0.0.0.0:9200/;
    proxy_set_header Host $host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
}
```

Note: update the proxy_pass field value to point to your ElasticSearch instance (if it is not localhost and default port).

3. Enable datastore features in CKAN

In your config file set:

```
ckan.datastore.enabled = 1
```

2.10.5 DataStorer: Automatically Add Data to the DataStore

Often, one wants data that is added to CKAN (whether it is linked to or uploaded to the *FileStore*) to be automatically added to the DataStore. This requires some processing, to extract the data from your files and to add it to the DataStore in the format the DataStore can handle.

This task of automatically parsing and then adding data to the datastore is performed by a DataStorer, a queue process that runs asynchronously and can be triggered by uploads or other activities. The DataStorer is an extension and can be found, along with installation instructions, at:

https://github.com/okfn/ckanext-datastorer

2.10.6 How It Works (Technically)

1. Request arrives at e.g. /dataset/{id}/resource/{resource-id}/data
2. CKAN checks authentication and authorization.
3. (Assuming OK) CKAN hands (internally) to ElasticSearch which handles the request
    - To do this we use Nginx's Sendfile / Accel-Redirect feature. This allows us to hand off a user request *directly* to ElasticSearch after the authentication and authorization. This avoids the need to proxy the request and results through CKAN code.

2.11 Background Tasks

CKAN allows you to create tasks that run in the 'background', that is asynchronously and without blocking the main application (these tasks can also be automatically retried in the case of transient failures). Such tasks can be created in *Extensions* or in core CKAN.

Background tasks can be essential to providing certain kinds of functionality, for example:

- Creating webhooks that notify other services when certain changes occur (for example a dataset is updated)
- Performing processing or validation or on data (as done by the Archiver and DataStorer Extensions)

2.11.1 Enabling Background Tasks

To manage and run background tasks requires a job queue and CKAN uses celery (plus the CKAN database) for this purpose. Thus, to use background tasks you need to install and run celery. As of CKAN 1.7, celery is a required library and will be already installed after a default CKAN install.

Installation of celery will normally be taken care of by whichever component or extension utilizes it so we skip that here.

To run the celery daemon you have two options:

1. In development setup you can just use paster. This can be done as simply as:

```
paster celeryd
```

This only works if you have a `development.ini` file in ckan root.

2. In production, the daemon should be run with a different ini file and be run as an init script. The simplest way to do this is to install supervisor:

   ```
   apt-get install supervisor
   ```

   Using this file as a template and copy to `/etc/supservisor/conf.d`:

   ```
   https://github.com/okfn/ckan/blob/master/ckan/config/celery-supervisor.conf
   ```

   Alternatively, you can run:

   ```
   paster celeryd --config=/path/to/file.ini
   ```

2.11.2 Writing Background Tasks

These instructions should show you how to write an background task and how to call it from inside CKAN or another extension using celery.

Examples

Here are some existing real examples of writing CKAN tasks:

- https://github.com/okfn/ckanext-archiver
- https://github.com/okfn/ckanext-qa
- https://github.com/okfn/ckanext-datastorer

Setup

An entry point is required inside the `setup.py` for your extension, and so you should add something resembling the following that points to a function in a module. In this case the function is called task_imports in the `ckanext.NAME.celery_import` module:

```
entry_points = """
  [ckan.celery_task]
  tasks = ckanext.NAME.celery_import:task_imports
"""
```

The function, in this case `task_imports` should be a function that returns fully qualified module paths to modules that contain the defined task (see the next section). In this case we will put all of our tasks in a file called `tasks.py` and so `task_imports` should be in a file called `ckanext/NAME/celery_import.py`:

```
def task_imports():
  return ['ckanext.NAME.tasks']
```

This returns an iterable of all of the places to look to find tasks, in this example we are only putting them in one place.

Implementing the tasks

The most straightforward way of defining tasks in our `tasks.py` module, is to use the decorators provided by celery. These decorators make it easy to just define a function and then give it a name and make it accessible to celery. Make sure you import celery from ckan.lib.celery_app:

```
from ckan.lib.celery_app import celery
```

Implement your function, specifying the arguments you wish it to take. For our sample we will use a simple echo task that will print out its argument to the console:

```
def echo( message ):
  print message
```

Next it is important to decorate your function with the celery task decorator. You should give the task a name, which is used later on when calling the task:

```
@celery.task(name = "NAME.echofunction")
def echo( message ):
  print message
```

That's it, your function is ready to be run asynchronously outside of the main execution of the CKAN app. Next you should make sure you run `python setup.py develop` in your extensions folder and then go to your CKAN installation folder (normally pyenv/src/ckan/) to run the following command:

```
paster celeryd
```

Once you have done this your task name `NAME.echofunction` should appear in the list of tasks loaded. If it is there then you are all set and ready to go. If not then you should try the following to try and resolve the problem:

1. Make sure the entry point is defined correctly in your `setup.py` and that you have executed `python setup.py develop`
2. Check that your task_imports function returns an iterable with valid module names in
3. Ensure that the decorator marks the functions (if there is more than one decorator, make sure the celery.task is the first one - which means it will execute last).
4. If none of the above helps, go into #ckan on irc.freenode.net where there should be people who can help you resolve your issue.

Calling the task

Now that the task is defined, and has been loaded by celery it is ready to be called. To call a background task you need to know only the name of the task, and the arguments that it expects as well as providing it a task id.:

```
import uuid
from ckan.lib.celery_app import celery
celery.send_task("NAME.echofunction", args=["Hello World"], task_id=str(uuid.uuid4()))
```

After executing this code you should see the message printed in the console where you ran `paster celeryd`.

Retrying on errors

Should your task fail to complete because of a transient error, it is possible to ask celery to retry the task, after some period of time. The default wait before retrying is three minutes, but you can optionally specify this in the call to retry via the countdown parameter, and you can also specify the exception that triggered the failure. For our example

the call to retry would look like the following - note that it calls the function name, not the task name given in the decorator:

```
try:
  ... some work that may fail, http request?
except Exception, e:
  # Retry again in 2 minutes
  echo.retry(args=(message), exc=e, countdown=120, max_retries=10)
```

If you don't want to wait a period of time you can use the eta datetime parameter to specify an explicit time to run the task (i.e. 9AM tomorrow)

2.12 Import ("Harvest") Data from Other Sites

The ckanext-harvest extension can automatically import ("harvest") datasets from multiple CKAN websites into a single CKAN website, and also provides a framework for writing custom harvesters to import data from non-CKAN sources.

2.12.1 CKAN harvester

The CKAN harvester plugin makes it really easy to import datasets from a remote CKAN instance into your own CKAN instance. It is highly customizable, allowing you to define default tags, groups, users and permissions for the imported datasets. Please refer to the documentation for more details:

https://github.com/okfn/ckanext-harvest#the-ckan-harvester

2.12.2 Other harvesters

There are other extensions offer different harvesters for other metadata sources. For instance, ckanext-inspire provides harvesters for CSW records that follow the ISO-19193 encoding.

See *CSW support* for more details.

2.12.3 Build your own

The harvesting extension provides an interface for building custom harvesters. The interface has three stages:

1. The *gather* stage compiles all the resource identifiers that need to be fetched in the next stage.
2. The *fetch* stage gets the contents of the remote objects and stores them in the database.
3. The *import* stage performs any necessary actions on the fetched resource.

See the following section in the ckanext-harvest README for more details:

https://github.com/okfn/ckanext-harvest#the-harvesting-interface

The CKAN harvester itself uses this interface:

https://github.com/okfn/ckanext-harvest/blob/master/ckanext/harvest/harvesters/ckanharvester.py

Here you can also find other examples of custom harvesters:

https://github.com/okfn/ckanext-pdeu/tree/master/ckanext/pdeu/harvesters

2.13 Geospatial Capabilities

CKAN offers a powerful set of geospatial features that allow adding spatial information to your datasets.

2.13.1 Spatial metadata for datasets

All the features in this section are provided by the ckanext-spatial extension (check the README for requirements and installation).

- Associate geographic data with datasets, with automatic geo-indexing of datasets (**spatial_metadata** plugin)

  When activated, this plugin allows users to associate a geometry with a CKAN dataset, by adding a GeoJSON extra value to the dataset with key **spatial**. For example:

  ```
  {"type":"Polygon","coordinates":[[[2.05827, 49.8625],[2.05827, 55.7447], [-6.41736, 55.7447], [-
  ```

  Or:

  ```
  { "type": "Point", "coordinates": [-3.145,53.078] }
  ```

  Datasets with spatial values are automatically geo-indexed, for example so that they can be searched using spatial filters.

  The spatial model requires the installation and configuration of PostGIS on your database backend, as described on the installation instructions.

- Search for datasets using spatial queries (**spatial_query** plugin)

  Once your datasets are geo-indexed, you can perform spatial queries by bounding box, via the following API call:

  ```
  /api/2/search/dataset/geo?bbox={minx,miny,maxx,maxy}[&crs={srid}]
  ```

  Or, starting on CKAN 1.6, as part of the default search:

  ```
  POST http://localhost:5000/api/action/package_search
  {
      "q": "Pollution",
      "extras": {
          "ext_bbox": "-7.535093,49.208494,3.890688,57.372349"
      }
  }
  ```

  Check the documentation on the ckanext-spatial README for more details:

  https://github.com/okfn/ckanext-spatial#spatial-query

- Add a map widget for spatial search queries (**spatial_query_widget** plugin)

  You can display a small map widget on the dataset search form, users can draw a rectangle on the map to add a spatial filter to their search queries.

  Check the documentation on the ckanext-spatial README for more details:

  https://github.com/okfn/ckanext-spatial#spatial-query-widget

- Add map widgets to datasets (**dataset_extent_map** plugin)

  You can add map widgets to dataset pages, to show the geometric extents of the datasets. Check the documentation in the ckanext-spatial README for more details:

  https://github.com/okfn/ckanext-spatial#dataset-extent-map

2.13.2 Metadata Conventions

Over time some conventions have emerged for storing geospatial information as extra metadata fields in datasets. Follow these conventions when storing information in CKAN datasets, so that your datasets will easily integrate with CKAN extensions and built-in functions, such as the automatic geo-indexing of datasets. The following metadata keys are used:

- spatial-text: Textual representation of the extent / location of the package
- spatial: GeoJSON representation of the extent of the package (Polygon or Point)
- spatial-uri: Linked Data URI representing the place name

For example:

```
* spatial-text: United Kingdom
* spatial: {"type": "Polygon", "coordinates": [[[0.50, 49.74], [0.5, 59.25], [-6.88, 59.25], [-6.88,
* spatial-uri: http://www.geonames.org/2635167
```

or:

```
* spatial-text: Matsushima
* spatial: {"type": "Point", "coordinates": [38.36, 141.07]}
* spatial-uri: http://www.geonames.org/2111964
```

2.13.3 CSW support

CKAN offers support for the Catalogue Service for the Web (CSW) standard. This support consist of:

- Ability to harvest records from remote CSW servers (as well as individual documents available online or Web Accessible Folders of them). CKAN supports the ISO-19139 encoding.
- Basic CSW interface (for the harvested records)

This is done via different extensions: ckanext-harvest (See *Import ("Harvest") Data from Other Sites*) offers the harvesting infrastucture, ckanext-inspire implements the harvesters for CSW servers and ckanext-csw is required by the previous one and offers the CSW interface.

Please refer to the README files of these extensions for instructions on how to install and configure them.

2.14 Translating Datasets, Groups and Tags

For translating CKAN's web interface see *Internationalize CKAN*. In addition to user interface internationalization, a CKAN administrator can also enter translations into CKAN's database for terms that may appear in the contents of datasets, groups or tags created by users. When a user is viewing the CKAN site, if the translation terms database contains a translation in the user's language for the name or description of a dataset or resource, the name of a tag or group, etc. then the translated term will be shown to the user in place of the original.

2.14.1 Setup and Configuration

By default term translations are disabled. To enable them, you have to specify the multilingual plugins using the `ckan.plugins` setting in your CKAN configuration file, for example:

```
# List the names of CKAN extensions to activate.
ckan.plugins = multilingual_dataset multilingual_group multilingual_tag
```

Of course, you won't see any terms getting translated until you load some term translations into the database. You can do this using the `term_translation_update` and `term_translation_update_many` actions of the CKAN API, See *CKAN API* for more details.

2.14.2 Loading Test Translations

If you want to quickly test the term translation feature without having to provide your own translations, you can load CKAN's test translations into the database by running this command from your shell:

```
paster --plugin=ckan create-test-data translations
```

See *Common CKAN Administrator Tasks* for more details.

2.14.3 Testing The Multilingual Extension

If you have a source installation of CKAN you can test the multilingual extension by running the tests located in `ckanext/multilingual/tests`. You must first install the packages needed for running CKAN tests into your virtual environment, and then run this command from your shell:

```
nosetests --ckan ckanext/multilingual/tests
```

See *Basic Tests* for more information.

CHAPTER 3

Publishing Datasets

3.1 Publishing Datasets

This tutorial provides a walk-through of publishing data on CKAN.

3.1.1 Quickstart Tutorial

3.1.2 Step-by-Step Tutorial (In Progress)

Step 0 - Identify Some Data to Use

For the purposes of this tutorial you will want to have some data which you want to publish on your CKAN instance (we will be using the DataHub in our examples but please replace with your own instance). If you don't have one to hand we suggest you just use some of the raw data from this Gold Prices Dataset on the DataHub (just pretend you dug it up somewhere on the Internet!).

3.1.3 Walkthroughs

Walkthrough of publishing some data and uploading it to the FileStore: http://ckan.org/2011/09/26/ux-improvements-file-uploading/

Walkthrough of publishing a dataset and uploading data to both FileStore and the DataStore: http://ckan.org/2012/03/27/ckan-datastore-and-data-api/

3.2 Load Datasets

You can upload individual datasets through the CKAN front-end, but for importing datasets on masse, you have two choices:

- *Import Data with the CKAN API*. You can use the CKAN API to script import. To simplify matters, we offer provide standard loading scripts for Google Spreadsheets, CSV and Excel.
- *Import Data with the Harvester Extension*. The CKAN harvester extension provides web and command-line interfaces for larger import tasks.

If you need advice on data import, contact the ckan-dev mailing list.

Note: If loading your data requires scraping a web page regularly, you may find it best to write a scraper on Scraper-

Wiki and combine this with either of the methods above.

3.2.1 Import Data with the CKAN API

You can use the CKAN API to upload datasets directly into your CKAN instance.

The Simplest Approach - ckanclient

The most basic way to automate dataset loading is with a Python script using the ckanclient library. You will need to register for an API key first.

You can install ckanclient with:

```
pip install ckanclient
```

Here is an example script to register a new dataset:

```
import ckanclient
# Instantiate the CKAN client.
ckan = ckanclient.CkanClient(api_key=my_api_key, base_location="http://myckaninstance.com/api")
# Describe the dataset.
dataset_entity = {
      'name': my_dataset_name,
      'url': my_dataset_url,
      'download_url': my_dataset_download_url,
      'tags': my_dataset_keywords,
      'notes': my_dataset_long_description,
}
# Register the dataset.
ckan.package_register_post(dataset_entity)
```

Loader Scripts

'Loader scripts' provide a simple way to take any format metadata and bulk upload it to a remote CKAN instance.

Essentially each set of loader scripts converts the dataset metadata to the standard 'dataset' format, and then loads it into CKAN.

To get a flavour of what loader scripts look like, take a look at the ONS scripts.

Loader Scripts for CSV and Excel

For CSV and Excel formats, the *SpreadsheetPackageImporter* (found in `ckanext-importlib/ckanext/importlib/spreadsheet_importer.py`) loader script wraps the file in *SpreadsheetData* before extracting the records into *SpreadsheetDataRecords*.

SpreadsheetPackageImporter copes with multiple title rows, data on multiple sheets, dates. The loader can reload datasets based on a unique key column in the spreadsheet, choose unique names for datasets if there is a clash, add/merge new resources for existing datasets and manage dataset groups.

Loader Scripts for Google Spreadsheets

The *SimpleGoogleSpreadsheetLoader* class (found in `ckanclient.loaders.base`) simplifies the process of loading data from Google Spreadsheets (there is an additional dependency on the `gdata` Python package).

This script has a simple example of loading data from Google Spreadsheets.

Write Your Own Loader Script

this needs work

First, you need an importer that derives from *PackageImporter* (found in `ckan/lib/importer.py`). This takes whatever format the metadata is in and sorts it into records of type *DataRecord*.

Next, each DataRecord is converted into the correct fields for a dataset using the *record_2_package* method. This results in dataset dictionaries.

The *PackageLoader* takes the dataset dictionaries and loads them onto a CKAN instance using the ckanclient. There are various settings to determine:

- ##how to identify the same dataset, previously been loaded into CKAN.## This can be simply by name or by an identifier stored in another field.
- how to merge in changes to an existing datasets. It can simply replace it or maybe merge in resources etc.

The loader should be given a command-line interface using the *Command* base class (`ckanext/command.py`).

You need to add a line to the CKAN `setup.py` (under `[console_scripts]`) and when you run `python setup.py develop` it creates a script for you in your Python environment.

3.2.2 Import Data with the Harvester Extension

The CKAN harvester extension provides useful tools for more advanced data imports.

These include a command-line interface and a web user interface for running harvesting jobs.

To use the harvester extension, create a class that implements the *harvester interface <https://github.com/okfn/ckanext-harvest/blob/master/ckanext/harvest/interfaces.py>* derived from the base class of the harvester extension.

For more information on working with extensions, see *Add Extensions*.

3.3 Set and Manage Permissions

CKAN implements a fine-grained role-based access control system.

This section describes:

- *Overview*. An overview of the concepts underlying CKAN authorization: objects, actions and roles.
- *Default Permissions*. The default permissions in CKAN.
- *Managing Permissions*. Managing and setting permissions.
- *Openness Modes*. Suitable for systems where you want to limit write access to CKAN.

3.3.1 Overview

In a nutshell: for a particular **object** (e.g. a dataset) a CKAN **user** can be assigned a **role** (e.g. editor) which allows permitted **actions** (e.g. read, edit).

In more detail, these concepts are as follows:

- There are **objects** to which access can be controlled, such as datasets and groups.
- For each object there are a set of relevant **actions**, such as create and edit, which users can perform on the object.
- To simplify mapping users to actions and objects, actions are aggregated into a set of **roles**. For example, an editor role would automatically have edit and read actions.
- Finally, CKAN has registered **users**.

Recent support for authorization profiles has been implemented using a publisher/group based profile that is described in *Publisher Profile and Workflow*.

Objects

Permissions are controlled per object: access can be controlled for an individual dataset, group or authorization group instance. Current objects include **datasets**, dataset **groups**, **authorization groups** and the **system**.

- A dataset is the basic CKAN concept of metadata about a dataset.
- A group of datasets can be set up to specify which users have permission to add or remove datasets from the group.
- Users can be assigned to authorization groups, to increase flexibility. Instead of specifying the privileges of specific users on a dataset or group, you can also specify a set of users that share the same rights. To do that, an authorization group can be set up and users can be added to it. Authorization groups are both the object of authorization (i.e. one can have several roles with regards to an authorization group, such as being allowed to read or edit it) and the subject of authorization (i.e. they can be assigned roles on other objects which will apply to their members, such as the group having edit rights on a particular group).
- Finally, the system object is special, serving as an object for assignments that do not relate to a specific object. For example, creating a dataset cannot be linked to a specific dataset instance, and is therefore a operation.

Actions

Actions are defined in the Action enumeration in `ckan/model/authz.py` and currently include: **edit**, **change-state**, **read**, **purge**, **edit-permissions**, **create-dataset**, **create-group**, **create-authorization-group**, **read-site**, **read-user**, **create-user**.

As noted above, some of these (e.g. **read**) have meaning for any type of object, while some (e.g. **create-dataset**) can not be associated with any particular object, and are therefore only associated with the system object.

The **read-site** action (associated with the system object) allows or denies access to pages not associated with specific objects. These currently include:

- Dataset search
- Group index
- Tags index
- Authorization Group index
- All requests to the API (on top of any other authorization requirements)

There are also some shortcuts that are provided directly by the authorization system (rather than being expressed as subject-object-role tuples):

- A user given the **admin** right for the **system** object is a 'sysadmin' and can perform any action on any object. (A shortcut for creating a sysadmin is by using the `paster sysadmin` command.)
- A user given the **admin** right for a particular object can perform any action on that object.

Roles

Each **role** has a list of permitted actions appropriate for a protected object.

Currently there are four basic roles:

- **reader**: can read the object
- **anon_editor**: anonymous users (i.e. not logged in) can edit and read the object
- **editor**: can edit, read and create new objects
- **admin**: admin can do anything including: edit, read, delete, update-permissions (change authorizations for that object)

You can add other roles if these defaults are not sufficient for your system.

Warning: If the broad idea of these basic roles and their actions is not suitable for your CKAN system, we suggest you create new roles, rather than edit the basic roles. If the definition of a role changes but its name does not, it is likely to confuse administrators and cause problems for CKAN upgrades and extensions.

Note: When you install a new CKAN extension, or upgrade your version of CKAN, new actions may be created, and permissions given to these basic roles, in line with the broad intention of the roles.

Users

You can manage CKAN users via the command line with the `paster user` command - for more information, see *user: Create and manage users*.

There are two special *pseudo-users* in CKAN, **visitor** and **logged-in**. These are used to refer to special sets of users, respectively those who are a) not logged-in ("visitor") and b) logged-in ("logged-in").

The `default_roles` config option in the CKAN config file lets you set the default authorization roles (i.e. permissions) for these two types of users. For more information, see *Reference: CKAN Configuration Options*.

3.3.2 Default Permissions

CKAN ships with the following default permissions:

- When a new dataset is created, its creator automatically becomes **admin** for it. This user can then change permissions for other users.
- By default, any other user (including both visitors and logged-ins) can read and write to this dataset.

These defaults can be changed in the CKAN config - see `default_roles` in *Reference: CKAN Configuration Options*.

3.3.3 Managing Permissions

The assignment of users and authorization groups to roles on a given protected object (such as a dataset) can be done by 'admins' via the 'authorization' tab of the web interface (or by sysadmins via that interface or the system admin interface).

There is also a command-line authorization manager, detailed below.

Command-line authorization management

Although the admin extension provides a Web interface for managing authorization, there is a set of more powerful `paster` commands for fine-grained control (see *Common CKAN Administrator Tasks*).

The `rights` command is used to configure the authorization roles of a specific user on a given object within the system.

For example, to list all assigned rights in the system (which you can then grep if needed):

```
paster --plugin=ckan rights -c my.ini list
```

The `rights make` command lets you assign specific permissions. For example, to give the user named **bar** the **admin** role on the dataset foo:

```
paster --plugin=ckan rights -c my.ini make bar admin dataset:foo
```

As well as users and datasets, you can assign rights to other objects. These include authorization groups, dataset groups and the system as a whole.

For example, to make the user 'chef' a system-wide admin:

```
paster --plugin=ckan rights -c my.ini make chef admin system
```

Or to allow all members of authorization group 'foo' to edit group 'bar':

```
paster --plugin=ckan rights -c my.ini make agroup:foo edit \
    group:bar
```

To revoke one of the roles assigned using `rights make`, the `rights remove` command is available. For example, to remove **bar**'s **admin** role on the foo dataset:

```
paster --plugin=ckan rights -c my.ini remove bar admin dataset:foo
```

The `roles` command lists and modifies the assignment of actions to roles.

To list all role assignments:

```
paster --plugin=ckan roles -c my.ini list
```

To remove the 'create-package' action from the 'editor' role:

```
paster --plugin=ckan roles -c my.ini deny editor create-package
```

And to re-assign 'create-package' to the 'editor' role:

```
paster --plugin=ckan roles -c my.ini allow editor create-package
```

For more help on either of these commands, you can use `--help` (as described in *Getting Help on Paster*):

```
paster --plugin=ckan roles --help
paster --plugin=ckan rights --help
```

3.3.4 Openness Modes

CKAN instances can be configured to operate in a range of authorization modes, with varying openness to edit. Here are some examples with details of how to set-up and convert between them.

1. Anonymous Edit Mode

Anyone can edit and create datasets without logging in. This is the default for CKAN out of the box.

2. Logged-in Edit Mode

You need to log-in and create/edit datasets. Anyone can create an account.

To operate in this mode:

1. First, change the visitor (any non-logged in user) rights from being able to create and edit datasets to just reading them:

```
paster rights make visitor reader system
paster rights make visitor reader package:all
paster rights remove visitor anon_editor package:all
paster rights remove visitor anon_editor system
```

2. Change the default rights for newly created datasets. Do this by using these values in your config file (see *Reference: CKAN Configuration Options*):

```
ckan.default_roles.Package = {"visitor": ["reader"], "logged_in": ["editor"]}
ckan.default_roles.Group = {"visitor": ["reader"], "logged_in": ["editor"]}
ckan.default_roles.System = {"visitor": ["reader"], "logged_in": ["editor"]}
ckan.default_roles.AuthorizationGroup = {"visitor": ["reader"], "logged_in": ["editor"]}
```

3. Publisher Mode

This allows edits only from authorized users. It is designed for installations where you wish to limit write access to CKAN and orient the system around specific publishing groups (e.g. government departments or specific institutions).

The key features are:

- Datasets are assigned to a specific publishing group.
- Only users associated to that group are able to create or update datasets associated to that group.

To operate in this mode:

1. First, remove the general public's rights to create and edit datasets:

```
paster rights remove visitor anon_editor package:all
paster rights remove logged_in editor package:all
paster rights remove visitor anon_editor system
paster rights remove logged_in editor system
```

2. If logged-in users have already created datasets in your system, you may also wish to remove their admin rights. For example:

```
paster rights remove bob admin package:all
```

3. Change the default rights for newly created datasets. Do this by using these values in your config file (see *Reference: CKAN Configuration Options*):

```
ckan.default_roles.Package = {"visitor": ["reader"], "logged_in": ["reader"]}
ckan.default_roles.Group = {"visitor": ["reader"], "logged_in": ["reader"]}
ckan.default_roles.System = {"visitor": ["reader"], "logged_in": ["reader"]}
ckan.default_roles.AuthorizationGroup = {"visitor": ["reader"], "logged_in": ["reader"]}
```

Note you can also restrict dataset edits by a user's authorization group.

3.4 Publisher Profile and Workflow

New in version 1.7.

The Publisher/Organization workflow in CKAN is designed to support a setup in which datasets are managed by a "Publisher" organization. Users can become members of one (or more) Organizations and their membership determines what datasets they have access to.

Specifically, the workflow looks like:

- A User joins or creates an Organization
  - If the user is the creator of the Organization then they become administrator of the Organization.
  - Otherwise they become a Member.
- New Members must be added by the Organization Administrator, although anyone can request to join an Organization
- User creates a dataset. On creation User must assign this dataset to a specific organization (and can only assign to a organization of which User is a member)
  - Other members of that Organization can then edit and update this dataset

This setup is a natural one for many situations. For example:

- Government. Organizations correspond to Departments or Ministries (or other organizational groups)
- Academia: Organizations again correspond to Departments or research groups

Whilst organizations can currently belong to other organizations the publisher authorization profile currently only checks membership of the current organization. Future versions of this extension may provide a configuration option to apply authorization checks hierarchically.

3.4.1 Enabling and Configuring the Publisher Profile

To switch CKAN to use the authorization publisher profile you need to set the following configuration option:

```
ckan.auth.profile = publisher
```

Setting auth.profile to publisher will enable the publisher authorization profile. Setting it to nothing, or if it is not present will force CKAN to use the default profile.

To enable the default organization and organization dataset forms you should include the following plugins in your configuration file:

```
ckan.plugins = organizations organizations_dataset
```

3.4.2 Technical Overview

- Organizations are a specialization of CKAN Groups. As such they retain many of their features.
- Authorization for most actions is granted based on shared membership of a group between the **user** and the **object** being manipulated.
- You can design custom forms for publisher sign up and editing.

In more detail, these concepts are as follows:

- *Domain Objects* such as *groups*, *datasets* and *users* can be added as members of groups.
- Each user within a group has a capacity with which it can interact with the group, currently these are *editor* and *administrator*.
- Even though groups are hierarchical there must be an intersection of the user's groups and the **object**'s groups for permission to be granted, as long as the capacity is appropriate. For instance, being an *editor* within a group does not necessarily grant authorization to edit the group.
- This means that individual permissions do not need to granted on a *user* by *user* basis, instead the user can just be added to the appropriate group.

3.5 Data Viewer

CKAN's resource page can provide a preview of the resource's data if it is of an appropriate format. If the data is available through the CKAN DataStore API, or if the data is a `csv` or `xls` file; then Recline's Data Explorer is used. If the data is another webpage; a google doc; or an image; then it is embedded in an iframe for viewing. Or if the data is text-like, then it's raw contents are displayed.

3.5.1 Data Explorer

The Recline Data Explorer provides a rich, queryable view of the data. The data can be filtered, faceted, graphed and mapped. Furthermore, the grid, graph or map can then be embedded into your own site using the **Embed** button, and copying the provided html snippet into your webpage.

3.5.2 How It Works (Technically)

The relevant code for setting up the data viewer is found in `application.js`.

All resources available through the DataStore API are available for viewing through the Data Explorer. using recline's `elasticsearch` backend. If the datastore is not available, and the filetype is normalized to `csv` or `xls`, then a dataproxy is used to attempt to view the data (using recline's `dataproxy` backend).

Embedding

If a resource is viewable through the Data Explorer, then it is also embeddable in third-party web pages. `/dataset/{name}/resource/{resource_id}/embed` provides a stripped-down page containing the data explorer. The data explorer's state is passed through using the url's query parameters.

CHAPTER 4

The CKAN API

4.1 CKAN API

The CKAN platform is not only available in a web browser, but also via its Application Programming Interface (API). The API provides programmatic access to the CKAN system. The API is very powerful and allows you do everything (and more) you can do via the web interface.

Using the API you can do things like:

- Access any bit of information in CKAN (if you are authorized!)
- Edit any piece of information in CKAN
- Create a whole new web front-end for CKAN (if you want!)

The CKAN API follows the RESTful (Representational State Transfer) style and uses JSON by default.

4.1.1 Tools for Accessing the API

In using the API you will be performing HTTP requests to urls like: `/api/rest/dataset` with this returning data in JSON format.

There are several ways you can access this URL directly:

- Put this URL into your web browser and view or save the resulting response – there are plugins for browsers like Firefox and Chrome that allow you to see JSON nicely formatted in your browser (e.g. JSONView for Chrome)
- Use a command-line program such as curl
- Write a program in your favourite language that uses an http library to access the URL

Clients

Alternatively, you can access the API using one the dedicated tools or libraries written specifically for CKAN. The following clients are available:

- dpm (data package manager): command-line client and Python library (maintained by core CKAN team)
- ckanclient - CKAN Python Client: Python client maintained by the core CKAN team
- CKAN Ruby: Ruby Client
- Ckan_client-PHP: PHP client
- Ckan_client-J: Java client

- net-ckan: PERL client
- ckanjs: sophisticated Javascript client built on Backbone.
- Google Refine CKAN Extension: Google Refine client which allows you to get and push data to and from a CKAN instance using Google Refine.

4.1.2 Quickstart Tutorial

CKAN API: Quickstart Tutorial

A quickstart tutorial for the CKAN API that walks through some of the main features. For full details of the API see *CKAN API* and its *API: Reference* section.

The following examples use the demonstration instance at http://test.ckan.org/. You may obviously replace this with your own site.

Details of tools for accessing the API including client libraries can be found in the *main API section here*.

Example queries

Search http://test.ckan.org/api/search/dataset?q=open+street+map

```
{"count": 4, "results": ["uk-naptan-osm", "osm-uk", "osm", "naptan"]}
```

Get dataset http://test.ckan.org/api/rest/dataset/osm

```
{"id": "a3dd8f64-9078-4f04-845c-e3f047125028", "name": "osm", "title": "Open Street Map", ...
```

Create a dataset

Note: You'll need an *api key*.

```
curl http://test.ckan.org/api/rest/dataset -d '{"name":"test", "title":"Test dataset"}' -H "Authoriza
```

Update a dataset (Using POST or PUT):

```
curl http://test.ckan.org/api/rest/dataset/test -d '{"name":"test", "title":"Changed Test dataset"}'
```

Viewing permissions To view authorization roles on a dataset:

```
curl http://test.ckan.org/api/action/roles_show -d '{"domain_object": "freshwateratlasrivers"}'
```

```
{
  "help": "Returns the roles that users (and authorization groups) have on a\n    particular domain_
  "result": {
    "domain_object_id": "9da77628-2ac5-4965-af12-c7c51cc1d99a",
    "domain_object_type": "Package",
    "roles": [
      {
        "authorized_group_id": null,
        "context": "Package",
        "id": "481b6cd8-350b-4599-bd20-5e3c0ed0a8cb",
        "package_id": "9da77628-2ac5-4965-af12-c7c51cc1d99a",
```

```
        "role": "editor",
        "user_id": "4229c297-fe28-4597-a191-3ebbbee6c47a",
        "user_object_role_id": "481b6cd8-350b-4599-bd20-5e3c0ed0a8cb"
      },
      {
        "authorized_group_id": null,
        "context": "Package",
        "id": "aba38fa7-2fb4-4f84-98e1-02cb76c5d95a",
        "package_id": "9da77628-2ac5-4965-af12-c7c51cc1d99a",
        "role": "admin",
        "user_id": "e7f30c0d-944b-4a69-84c4-61b08bbf6b98",
        "user_object_role_id": "aba38fa7-2fb4-4f84-98e1-02cb76c5d95a"
      },
      {
        "authorized_group_id": null,
        "context": "Package",
        "id": "e06b1293-86ec-4417-8e28-b9499161348e",
        "package_id": "9da77628-2ac5-4965-af12-c7c51cc1d99a",
        "role": "reader",
        "user_id": "41cb1162-3d61-4b16-a3af-4cae27836ac5",
        "user_object_role_id": "e06b1293-86ec-4417-8e28-b9499161348e"
      }
    ]
  },
  "success": true
}
```


Looking at the list of "roles" we can see who has what permissions on this dataset. User with id="4229..." is an "editor", id="e7f3..." is an "admin", "41cb..." is a "reader". By using the user_show call to reveal the names of the users, we see that "visitor" (i.e. anyone who is not logged in) is the "reader" (no write permission), "logged-in" (any logged-in user) is the "editor" and the admin is "OKFN" who is the user who created this dataset in the first place and can therefore confer permissions to other users.

Adding permissions To give user "dread" the "admin" authorization role on dataset "freshwateratlasrivers":

```
curl http://test.ckan.org/api/action/user_role_update -d '{"user": "dread", "domain_object": "freshwa
```

Javascript examples

See http://okfnlabs.org/ckanjs/ (demo search and count widgets)

4.1.3 Authentication and API Keys

CKAN can be configured to only allow authorized users to carry out certain actions (see *Set and Manage Permissions* for more details). The authorization configuration is the same for actions done over the API as for those carried out in the CKAN web interfac, so a user has the same permissions, whichever way he/she accesses CKAN data.

Thus, all actions **not** permitted to anonymous users, will require a user to identify themselves via some authentication method.

Note: Depending on the authorization settings of the CKAN instance, a user may need to authenticate themselves for all operations including read. However, by default, CKAN allows anonymous read access and this setup is assumed for the API usage examples.

The standard authentication method utilized by CKAN is API keys and a user authenticates his/her user identity by supplying a header in the request containing the API key. The header field is either `Authorization`, `X-CKAN-API-Key` or configured with the *apikey_header_name* option. (Details of how to obtain an API key are below).

For example:

```
curl http://thedatahub.org/api/rest/package -d '{"name": "test"}' -H 'Authorization: fde34a3c-b716-4c
```

If requests that are required to be authorized are not sent with a valid Authorization header, for example the user associated with the key is not authorized for the operation, or the header is somehow malformed, then the requested operation will not be carried out and the CKAN API will respond with status code 403.

Obtaining an API key

1. Log-in to the particular CKAN website: `/user/login`
2. Your user page will show the API Key in your details section at the top of the screen

4.1.4 JSONP Support

To cater for scripts from other sites that wish to access the API, the data can be returned in JSONP format, where the JSON data is 'padded' with a function call. The function is named in the 'callback' parameter.

Example normal request:

```
GET /api/rest/dataset/pollution_stats
returns: {"name": "pollution_stats", ... }
```

but now with the callback parameter:

```
GET /api/rest/dataset/pollution_stats?callback=name-of-callback-function
returns: jsoncallback({"name": "pollution_stats", ... });
```

This parameter can apply to all POST requests to the Action API and GET requests to the Search API and v1/v2/v3 APIs.

4.1.5 API: Reference

Model, Search and Actions APIs

These sections describe the resource locations, data formats, and status codes which comprise the CKAN API.

The CKAN API is versioned, so that backwards incompatible changes can be introduced without removing existing support. A particular version of the API can be used by including its version number after the API location and before the resource location.

If the API version is not specified in the request, then the API will default to version 1.

Version 1 and 2

API Details - Versions 1 & 2 The CKAN API version 1 & 2 is separated into three parts.

- Model API
- Search API

The resources, methods, and data formats of each are described below.

Versions 1 & 2 These are very similar, but when the API returns a reference to an object, Version 1 API will return the Name of the object (e.g. "river-pollution") and Version 2 API will return the ID of the object (e.g. "a3dd8f64-9078-4f04-845c-e3f047125028").

The reason for this is that Names can change, so to reliably refer to the same dataset every time, you will want to use the ID and therefore use API v2. Alternatively, many people prefer to deal with Names, so API v1 suits them.

When making requests, you can call objects by either their Name or ID, interchangeably.

The only exception for this is for Tag objects. Since Tag names are immutable, they are always referred to with their Name.

Locators The locator for a given resource can be formed by appending the relative path for that resource to the API locator.

```
Resource Locator = API Locator + Resource Path
```

The API locators for the CKAN APIs (by version) are:

- `/api` (version 1)
- `/api/1` (version 1)
- `/api/2` (version 2)

The relative paths for each resource are listed in the sections below.

Model API Model resources are available at published locations. They are represented with a variety of data formats. Each resource location supports a number of methods.

The data formats of the requests and the responses are defined below.

Model Resources Here are the resources of the Model API.

Model Resource	Location
Dataset Register	`/rest/dataset`
Dataset Entity	`/rest/dataset/DATASET-REF`
Group Register	`/rest/group`
Group Entity	`/rest/group/GROUP-REF`
Tag Register	`/rest/tag`
Tag Entity	`/rest/tag/TAG-NAME`
Rating Register	`/rest/rating`
Dataset Relationships Register	`/rest/dataset/DATASET-REF/relationships`
Dataset Relationships Register	`/rest/dataset/DATASET-REF/RELATIONSHIP-TYPE`
Dataset Relationships Register	`/rest/dataset/DATASET-REF/relationships/DATASET-REF`
Dataset Relationship Entity	`/rest/dataset/DATASET-REF/RELATIONSHIP-TYPE/DATASET-REF`
Dataset's Revisions Entity	`/rest/dataset/DATASET-REF/revisions`
Revision Register	`/rest/revision`
Revision Entity	`/rest/revision/REVISION-ID`
License List	`/rest/licenses`

Possible values for DATASET-REF are the dataset id, or the current dataset name.

Possible values for RELATIONSHIP-TYPE are described in the Relationship-Type data format.

Model Methods Here are the methods of the Model API.

Resource	Method	Request	Response
Dataset Register	GET		Dataset-List
Dataset Register	POST	Dataset	
Dataset Entity	GET		Dataset
Dataset Entity	PUT	Dataset	
Group Register	GET		Group-List
Group Register	POST	Group	
Group Entity	GET		Group
Group Entity	PUT	Group	
Tag Register	GET		Tag-List
Tag Entity	GET		Dataset-List
Rating Register	POST	Rating	
Rating Entity	GET		Rating
Dataset Relationships Register	GET		Pkg-Relationships
Dataset Relationship Entity	GET		Pkg-Relationship
Dataset Relationships Register	POST	Pkg-Relationship	
Dataset Relationship Entity	PUT	Pkg-Relationship	
Dataset's Revisions Entity	GET		Pkg-Revisions
Revision List	GET		Revision-List
Revision Entity	GET		Revision
License List	GET		License-List

In general:

- GET to a register resource will *list* the entities of that type.
- GET of an entity resource will *show* the entity's properties.
- POST of entity data to a register resource will *create* the new entity.
- PUT of entity data to an existing entity resource will *update* it.

It is usually clear whether you are trying to create or update, so in these cases, HTTP POST and PUT methods are accepted by CKAN interchangeably.

Model Formats Here are the data formats for the Model API:

Name	Format
Dataset-Ref	Dataset-Name-String (API v1) OR Dataset-Id-Uuid (API v2)
Dataset-List	[Dataset-Ref, Dataset-Ref, Dataset-Ref, ...]
Dataset	{ id: Uuid, name: Name-String, title: String, version: String, url: String, resources: [Resource, Resource, ...], author: String, author_email: String, maintainer: String, maintainer_email: String, license_id: String, tags: Tag-List, notes: String, extras: { Name-String: String, ... } } See note below on additional fields upon GET of a dataset.
Group-Ref	Group-Name-String (API v1) OR Group-Id-Uuid (API v2)
Group-List	[Group-Ref, Group-Ref, Group-Ref, ...]
Group	{ name: Group-Name-String, title: String, description: String, packages: Dataset-List }
Tag-List	[Name-String, Name-String, Name-String, ...]
Tag	{ name: Name-String }
Resource	{ url: String, format: String, description: String, hash: String }
Rating	{ dataset: Name-String, rating: int }
Pkg-Relationships	[Pkg-Relationship, Pkg-Relationship, ...]
Pkg-Relationship	{ subject: Dataset-Name-String, object: Dataset-Name-String, type: Relationship-Type, comment: String }
Pkg-Revisions	[Pkg-Revision, Pkg-Revision, Pkg-Revision, ...]
Pkg-Revision	{ id: Uuid, message: String, author: String, timestamp: Date-Time }
Relationship-Type	One of: 'depends_on', 'dependency_of', 'derives_from', 'has_derivation', 'child_of', 'parent_of', 'links_to', 'linked_from'.
Revision-List	[revision_id, revision_id, revision_id, ...]
Revision	{ id: Uuid, message: String, author: String, timestamp: Date-Time, datasets: Dataset-List }
License-List	[License, License, License, ...]
License	{ id: Name-String, title: String, is_okd_compliant: Boolean, is_osi_compliant: Boolean, tags: Tag-List, family: String, url: String, maintainer: String, date_created: Date-Time, status: String }

To send request data, create the JSON-format string (encode in UTF8) put it in the request body and send it using PUT or POST.

Response data will be in the response body in JSON format.

Notes:

- When you update an object, fields that you don't supply will remain as they were before.
- To delete an 'extra' key-value pair, supply the key with JSON value: `null`
- When you read a dataset, some additional information is supplied that you cannot modify and POST back to the CKAN API. These 'read-only' fields are provided only on the Dataset GET. This is a convenience to clients, to save further requests. This applies to the following fields:

Key	Description
id	Unique Uuid for the Dataset
revision_id	Latest revision ID for the core Package data (but is not affected by changes to tags, groups, extras, relationships etc)
metadata_created	Date the Dataset (record) was created
metadata_modified	Date the Dataset (record) was last modified
relationships	info on Dataset Relationships
ratings_average	
ratings_count	
ckan_url	full URL of the Dataset
download_url (API v1)	URL of the first Resource
isopen	boolean indication of whether dataset is open according to Open Knowledge Definition, based on other fields
notes_rendered	HTML rendered version of the Notes field (which may contain Markdown)

Search API Search resources are available at published locations. They are represented with a variety of data formats. Each resource location supports a number of methods.

The data formats of the requests and the responses are defined below.

Search Resources Here are the published resources of the Search API.

Search Resource	Location
Dataset Search	`/search/dataset`
Resource Search	`/search/resource`
Revision Search	`/search/revision`
Tag Counts	`/tag_counts`

See below for more information about dataset and revision search parameters.

Search Methods Here are the methods of the Search API.

Resource	Method	Request	Response
Dataset Search	POST	Dataset-Search-Params	Dataset-Search-Response
Resource Search	POST	Resource-Search-Params	Resource-Search-Response
Revision Search	POST	Revision-Search-Params	Revision-List
Tag Counts	GET		Tag-Count-List

It is also possible to supply the search parameters in the URL of a GET request, for example `/api/search/dataset?q=geodata&allfields=1`.

Search Formats Here are the data formats for the Search API.

Name	Format
Dataset-Search-Params Resource-Search-Params Revision-Search-Params	{ Param-Key: Param-Value, Param-Key: Param-Value, ... } See below for full details of search parameters across the various domain objects.
Dataset-Search-Response	{ count: Count-int, results: [Dataset, Dataset, ...] }
Resource-Search-Response	{ count: Count-int, results: [Resource, Resource, ...] }
Revision-List	[Revision-Id, Revision-Id, Revision-Id, ...] NB: Ordered with youngest revision first
Tag-Count-List	[[Name-String, Integer], [Name-String, Integer], ...]

The `Dataset` and `Revision` data formats are as defined in Model Formats.

Dataset Parameters

Param-Key	Param-Value	Examples	Notes
q	Search-String	q=geodata q=government+sweden q=%22drug%20abuse%22 q=tags:"river pollution"	Criteria to search the dataset fields for. URL-encoded search text. (You can also concatenate words with a '+' symbol in a URL.) Search results must contain all the specified words. You can also search within specific fields.
qjson	JSON encoded options	['q':'geodata']	All search parameters can be json-encoded and supplied to this parameter as a more flexible alternative in GET requests.
title, tags, notes, groups, author, maintainer, update_frequency, or any 'extra' field name e.g. department	Search-String	title=uk&tags=health department=environment tags=health&tags=pollution tags=river%20pollution	Search in a particular a field.
order_by	field-name (default=rank)	order_by=name	Specify either rank or the field to sort the results by
offset, limit	result-int (defaults: offset=0, limit=20)	offset=40&limit=20	Pagination options. Offset is the number of the first result and limit is the number of results to return.
all_fields	0 (default) or 1	all_fields=1	Each matching search result is given as either a dataset name (0) or the full dataset record (1).

Note: filter_by_openness and filter_by_downloadable were dropped from CKAN version 1.5 onwards.

Resource Parameters

Param-Key	Param-Value	Example	Notes
url, format, description	Search-String	url=statistics.org format=xls description=Research+Institute	Criteria to search the dataset fields for. URL-encoded search text. This search string must be found somewhere within the field to match. Case insensitive.
qjson	JSON encoded options	['url':'www.statistics.org']	All search parameters can be json-encoded and supplied to this parameter as a more flexible alternative in GET requests.
hash	Search-String	hash=b0d7c260-35d4-42ab-9e3d-c1f4db9bc2f0	Searches for an match of the hash field. An exact match or match up to the length of the hash given.
all_fields	0 (default) or 1	all_fields=1	Each matching search result is given as either an ID (0) or the full resource record
offset, limit	result-int (defaults: offset=0, limit=20)	offset=40&limit=20	Pagination options. Offset is the number of the first result and limit is the number of results to return.

Note: Powerful searching from the command-line can be achieved with curl and the qjson parameter. In this case you need to remember to escapt the curly braces and use url encoding (e.g. spaces become `%20`). For example:

```
curl 'http://thedatahub.org/api/search/dataset?qjson=\{"author":"The%20Stationery%20Office%20Limited"
```

Revision Parameters

Param-Key	Param-Value	Example	Notes
since_time	Date-Time	since_time=2010-05-05T19:42:45.854533	The time can be less precisely stated (e.g 2010-05-05).
since_id	Uuid	since_id=6c9f32ef-1f93-4b2f-891b-fd01924ebe08	The stated id will not be included in the results.

Status Codes Standard HTTP status codes are used to signal method outcomes.

Code	Name
200	OK
201	OK and new object created (referred to in the Location header)
301	Moved Permanently
400	Bad Request
403	Not Authorized
404	Not Found
409	Conflict (e.g. name already exists)
500	Service Error

Version 3

This version is in beta.

Model, Search and Action API: Version 3

API Versions

To use a particular version of the API, include the version number in the base of the URL:

- `http://ckan.net/api/1` (version 1)
- `http://ckan.net/api/2` (version 2)
- `http://ckan.net/api/3` (version 3)

e.g. Searching using API version 2:: http://ckan.net/api/2/search/dataset?q=spending

If you don't specify the version number then you will default to version 1 of the Model, Search and Util APIs and version 3 of th

- `http://ckan.net/api/rest` (version 1)
- `http://ckan.net/api/search` (version 1)
- `http://ckan.net/api/util` (version 1)
- `http://ckan.net/api/action` (version 3)

Action API

> **Warning:** The Action API is in beta in this release. Please feed back comments and problems. There is a known issue that incorrect parameters cause unhelpful errors with status 500.

Overview The Action API is a powerful RPC-style way of accessing CKAN data. Its intention is to have access to all the core logic in ckan. It calls exactly the same functions that are used internally which all the other CKAN interfaces (Web interface / Model API) go through. Therefore it provides the full gamut of read and write operations, with all possible parameters.

A client supplies parameters to the Action API via a JSON dictionary of a POST request, and returns results, help information and any error diagnostics in a JSON dictionary too. This is a departure from the CKAN API versions 1 and 2, which being RESTful required all the request parameters to be part of the URL.

Requests

URL The basic URL for the Action API is:

```
/api/action/{logic_action}
```

Examples:

```
/api/action/package_list
/api/action/package_show
/api/action/user_create
```

Actions get.py:

Logic Action	Parameter keys
site_read	(none)
package_list	(none)

Continued on next page

Table 4.1 – continued from previous page

Logic Action	Parameter keys
current_package_list_with_resources	limit, page
revision_list	(none)
package_revision_list	id
group_list	all_fields
group_list_authz	(none)
group_list_available	(none)
group_revision_list	id
licence_list	(none)
tag_list	q, all_fields, limit, offset, return_objects, vocabulary_id
user_list	q, order_by
package_relationships_list	id, id2, rel
vocabulary_list	(none)
package_show	id
revision_show	id
group_show	id
related_show	id
related_list	id
tag_show	id
user_show	id
term_translation_show	"**terms**" A list of strings, the terms that you want to search for translations of, e.g. "russian", "
package_show_rest	id
group_show_rest	id
tag_show_rest	id
vocabulary_show	id
task_status_show	id
task_status_show	entity_id, task_type, key
resource_status_show	id
package_autocomplete	q
tag_autocomplete	q, fields, offset, limit, vocabulary_id
format_autocomplete	q, limit
user_autocomplete	q, limit
package_search	q, fields, facet_by, limit, offset
tag_search	q, fields, offset, limit, vocabulary_id
roles_show	domain_object, (user), (authorization_group)

create.py:

Logic Action	Parameter keys
package_create	(package keys)
package_create_validate	(package keys)
resource_create	(resource keys)
package_relationship_create	id, id2, rel, comment
group_create	(group keys)
rating_create	package, rating
related_create	(related keys)
user_create	(user keys)
package_create_rest	(package keys)
group_create_rest	(group keys)
vocabulary_create	(vocabulary keys)
tag_create	(tag keys)

update.py:

Logic Action	Parameter keys
make_latest_pending_pa	ckage_active
resource_update	(resource keys)
package_update	(package keys)
pack- age_update_validate	(package keys)
pack- age_relationship_update	id, id2, rel, comment
group_update	(group keys)
user_update	(user keys), reset_key
pack- age_update_rest	(package keys)
related_update	(related keys)
group_update_rest	(group keys)
user_role_update	user OR authorization_group, domain_object, roles
user_role_bulk_update	user_roles, domain_object
vocabulary_update	(vocabulary keys)
term_translation_update	"**term**" The term that you want to create (or update) a translation for, e.g. "russian", "romantic novel". "**term_translation**" the translation of the term, e.g. "Russisch", "Liebesroman". "**lang_code**" the language code for the translation, e.g. "fr", "de".
term_translation_update_	many A list of dictionaries with keys matching the parameter keys for term_translation_update

delete.py:

Logic Action	Parameter keys
package_delete	id
package_relationship_delete	id, id2, rel
group_delete	id
related_delete	id
vocabulary_delete	id
tag_delete	id, vocabulary_id

In case of doubt, refer to the code of the logic actions, which is found in the CKAN source in the ckan/logic/action directory.

Object dictionaries Package:

key	example value	notes
id	"fd788e57-dce4-481c-832d-497235bf9f78"	(Read-only) unique identifier
name	"uk-spending"	Unique identifier. Should be human readable
title	"UK Spending"	Human readable title of the dataset
url	"http://gov.uk/spend-downloads.html"	Home page for the data
version	"1.0"	Version associated with the data. String format.
author	"UK Treasury"	Name of person responsible for the data
author_email	"contact@treasury.gov.uk"	Email address for the person in the 'author' field
maintainer	null	Name of another person responsible for the data
maintainer_email	null	Email address for the person in the 'maintainer' field
notes	"### About\r\n\r\nUpdated 1997."	Other human readable info about the dataset. Markdown format.
license_id	"cc-by"	ID of the license this dataset is released under. You can then look up the license ID to get the title.
extras	[]	
tags	[{"name": "government-spending"}, {"name": "climate"}]	List of tags associated with this dataset.
groups	[{"name": "spending"}, {"name": "country-uk"}]	List of groups this dataset is a member of.
relationships_as_subject	[]	List of relationships. The 'type' of the relationship is described in terms of this package being the subject and the related package being the object.
state	active	May be `deleted` or other custom states like `pending`.
revision_id	"f645243a-7334-44e2-b87c-64231700a9a6"	(Read-only) ID of the last revision for the core package object was (doesn't include tags, groups, extra fields, relationships).
revision_timestamp	"2010-12-21T15:26:17.345502"	(Read-only) Time and date when the last revision for the core package object was (doesn't include tags, groups, extra fields, relationships). ISO format. UTC timezone assumed.

Package Extra:

key	example value	notes
id	"c10fb749-ad46-4ba2-839a-41e8e2560687"	(Read-only)
key	"number_of_links"	
value	"10000"	
package_id	"349259a8-cbff-4610-8089-2c80b34e27c5"	(Read-only) Edit package extras with package_update
state	"active"	(Read-only) Edit package extras with package_update
revision_timestamp	"2010-09-01T08:56:53.696551"	(Read-only)
revision_id	"233d0c19-fcdc-44b9-9afe-25e2aa9d0a5f"	(Read-only)

Resource:

key	example value	notes
id	"888d00e9-6ee5-49ca-9abb-6f216e646345"	(Read-only)
url	"http://gov.uk/spend-july-2009.csv"	Download URL of the data
description	""	
format	"XLS"	Format of the data
hash	null	Hash of the data e.g. SHA1
state	"active"	
position	0	(Read-only) This is set by the order of resources are given in the list when creating/updating the package.
resource_group_id	"49ddadb0-dd80-9eff-26e9-81c5a466cf6e"	(Read-only)
revision_id	"188ac88b-1573-48bf-9ea6-d3c503db5816"	(Read-only)
revision_timestamp	"2011-07-08T14:48:38.967741"	(Read-only)

Tag:

key	example value	notes
id	"b10871ea-b4ae-4e2e-bec9-a8d8ff357754"	(Read-only)
name	"country-uk"	(Read-only) Add/remove tags from a package or group using update_package or update_group
display_name	"country-uk"	(Read-only) display_name is the name of the tag that is displayed to user (as opposed to name which is used to identify the tag, e.g. in URLs). display_name is is usually the same as name but may be different, for example display_names may be translated by the ckanext-multilingual extension.
state	"active"	(Read-only) Add/remove tags from a package or group using update_package or update_group
revision_timestamp	"2009-08-08T12:46:40.920443"	(Read-only)
vocabulary_id	"Genre"	(Read-only) Vocabulary name or id. Optional.

user_roles:

key	example value	notes
user	"5ba3985d-447d-4919-867e-2ffe22281c40"	Provide exactly one out of "user" and "authorization_group" parameters.
authorization_group	"16f8f7ba-1a97-4d27-95ce-5e8827a0d75f"	
roles	['editor', 'admin']	

Related:

key	example value	notes
id	"b10871ea-b4ae-4e2e-bec9-a8d8ff357754"	(Read-only)
title	"A new visualization"	(Read-only)
type	"Visualization"	(Read-only)
description	"Describing the visualization"	(Read-only)
url	"http://invent.ge/HKjuyc"	(Read-only) Where the item can be found
image_url	"http://invent.ge/IR5r7W"	(Read-only) An optional image showing the item

Vocabulary:

key	example value	notes
id	“b10871ea-b4ae-4e2e-bec9-a8d8ff357754”	(Read-only)
name	“Genre”	
tags	[{“name”:”government-spending”}, {“name”: “climate”}]	List of tags belonging to this vocabulary.

Term Translation:

key	example value	notes
term	“russian”	The term that is being translated.
term_translation	“Russisch”	The translation of the term.
lang_code	“de”	The language of the translation, a language code string.

Parameters Requests must be a POST, including parameters in a JSON dictionary. If there are no parameters required, then an empty dictionary is still required (or you get a 400 error).

Examples:

```
curl http://test.ckan.net/api/action/package_list -d '{}'
curl http://test.ckan.net/api/action/package_show -d '{"id": "fd788e57-dce4-481c-832d-497235bf9f78"}'
```

Responses The response is wholly contained in the form of a JSON dictionary. Here is the basic format of a successful request:

```
{"help": "Creates a package", "success": true, "result": ...}
```

And here is one that incurred an error:

```
{"help": "Creates a package", "success": false, "error": {"message": "Access denied", "__type": "Auth
```

Where:

- `help` is the ‘doc string’ (or `null`)
- `success` is `true` or `false` depending on whether the request was successful. The response is always status 200, so it is important to check this value.
- `result` is the main payload that results from a successful request. This might be a list of the domain object names or a dictionary with the particular domain object.
- `error` is supplied if the request was not successful and provides a message and __type. See the section on errors.

Errors

The message types include:

- Authorization Error - an API key is required for this operation, and the corresponding user needs the correct credentials
- Validation Error - the object supplied does not meet with the standards described in the schema.
- (TBC) JSON Error - the request could not be parsed / decoded as JSON format, according to the Content-Type (default is `application/x-www-form-urlencoded;utf-8`).

Examples

```
$ curl http://ckan.net/api/action/package_show -d '{"id": "fd788e57-dce4-481c-832d-497235bf9f78"}'
{"help": null, "success": true, "result": {"maintainer": null, "name": "uk-quango-data", "relationshi
```

Search API Search resources are available at published locations. They are represented with a variety of data formats. Each resource location supports a number of methods.

The data formats of the requests and the responses are defined below.

Search Resources Here are the published resources of the Search API.

Search Resource	Location
Dataset Search	`/search/dataset`
Resource Search	`/search/resource`
Revision Search	`/search/revision`
Tag Counts	`/tag_counts`

See below for more information about dataset and revision search parameters.

Search Methods Here are the methods of the Search API.

Resource	Method	Request	Response
Dataset Search	POST	Dataset-Search-Params	Dataset-Search-Response
Resource Search	POST	Resource-Search-Params	Resource-Search-Response
Revision Search	POST	Revision-Search-Params	Revision-List
Tag Counts	GET		Tag-Count-List

It is also possible to supply the search parameters in the URL of a GET request, for example `/api/search/dataset?q=geodata&allfields=1`.

Search Formats Here are the data formats for the Search API.

Name	Format
Dataset-Search-Params Resource-Search-Params Revision-Search-Params	{ Param-Key: Param-Value, Param-Key: Param-Value, ... } See below for full details of search parameters across the various domain objects.
Dataset-Search-Response	{ count: Count-int, results: [Dataset, Dataset, ...] }
Resource-Search-Response	{ count: Count-int, results: [Resource, Resource, ...] }
Revision-List	[Revision-Id, Revision-Id, Revision-Id, ...] NB: Ordered with youngest revision first
Tag-Count-List	[[Name-String, Integer], [Name-String, Integer], ...]

Dataset Parameters

These parameters are all the standard SOLR syntax (in contrast to the syntax used in CKAN API versions 1 and 2). Here is a summary of the main features:

Param-Key	Param-Value	Examples	Notes
q	Search-String	q=geodata q=government%20sweden q=%22drug%20abuse%22 q=title:census q=tags:maps&tags:country-uk	Criteria to search the dataset fields for. URL-encoded search text. Search results must contain all the specified words. Use colon to specify which field to search in. (Extra fields are not currently supported.)
qjson	JSON encoded options	['q':'geodata']	All search parameters can be json-encoded and supplied to this parameter as a more flexible alternative in GET requests.
fl	list of fields	fl=name fl=name,title fl=*	Which fields to return. * is all.
sort	field name, asc / dec	sort=name asc sort=metadata_modified asc	Changes the sort order according to the field and direction given. default: score desc, name asc
start, rows	result-int (defaults: start=0, rows=20)	start=40&rows=20	Pagination options. Start is the number of the first result and rows is the number of results to return.
all_fields	0 (default) or 1	all_fields=1	Each matching search result is given as either a dataset name (0) or the full dataset record (1).

Resource Parameters

Param-Key	Param-Value	Example	Notes
url, format, description	Search-String	url=statistics.org format=xls description=Research+Institute	Criteria to search the dataset fields for. URL-encoded search text. This search string must be found somewhere within the field to match. Case insensitive.
qjson	JSON encoded options	['url':'www.statistics.org']	All search parameters can be json-encoded and supplied to this parameter as a more flexible alternative in GET requests.
hash	Search-String	hash=b0d7c260-35d4-42ab-9e3d-c1f4db9bc2f0	Searches for an match of the hash field. An exact match or match up to the length of the hash given.
all_fields	0 (default) or 1	all_fields=1	Each matching search result is given as either an ID (0) or the full resource record
offset, limit	result-int (defaults: offset=0, limit=20)	offset=40&limit=20	Pagination options. Offset is the number of the first result and limit is the number of results to return.

Revision Parameters

Param-Key	Param-Value	Example	Notes
since_time	Date-Time	since_time=2010-05-05T19:42:45.854533	The time can be less precisely stated (e.g 2010-05-05).
since_id	Uuid	since_id=6c9f32ef-1f93-4b2f-891b-fd01924ebe08	The stated id will not be included in the results.

Status Codes The Action API aims to return status `200 OK`, whether there are errors or not. The response body contains the *success* field indicating whether an error occurred or not. When `"success": false` then you will receive details of the error in the *error* field. For example requesting a dataset that doesn't exist:

```
curl http://test.ckan.net/api/action/package_show -d '{"id": "unknown_id"}'
```

gives:

```
{"help": null, "success": false, "error": {"message": "Not found", "__type": "Not Found Error"}}
```

Alternatively, requests to the Action API that have major formatting problems may result in a 409, 400, or 500 error (in order of increasing severity), but future CKAN releases aim to avoid these responses in favour of the previously described method of providing the error message.

The Search API returns standard HTTP status codes to signal method outcomes:

Code	Name
200	OK
201	OK and new object created (referred to in the Location header)
301	Moved Permanently (redirect)
400	Bad Request
403	Not Authorized - have you forgotton to specify your API Key?
404	Not Found
409	Conflict - error during processing of the request
500	Service Error - unhandled error - the system administrator has been notified

Util API

The Util API provides various utility APIs – e.g. auto-completion APIs used by front-end javascript.

Util API

The Util API provides various utility APIs – e.g. auto-completion APIs used by front-end javascript.

All Util APIs are read-only. The response format is JSON. Javascript calls may want to use the JSONP formatting.

Note: Some CKAN deployments have the API deployed at a different domain to the main CKAN website. To make sure that the AJAX calls in the Web UI work, you'll need to configue the ckan.api_url. e.g.:

```
ckan.api_url = http://api.example.com/
```

---

dataset autocomplete There an autocomplete API for package names which matches on name or title.

This URL:

```
/api/2/util/dataset/autocomplete?incomplete=a%20novel
```

Returns:

```
{"ResultSet": {"Result": [{"match_field": "title", "match_displayed": "A Novel By Tolstoy (annakareni
```

tag autocomplete There is also an autocomplete API for tags which looks like this:

This URL:

```
/api/2/util/tag/autocomplete?incomplete=ru
```

Returns:

```
{"ResultSet": {"Result": [{"Name": "russian"}]}}
```

resource format autocomplete Similarly, there is an autocomplete API for the resource format field which is available at:

```
/api/2/util/resource/format_autocomplete?incomplete=cs
```

This returns:

```
{"ResultSet": {"Result": [{"Format": "csv"}]}}
```

markdown Takes a raw markdown string and returns a corresponding chunk of HTML. CKAN uses the basic Markdown format with some modifications (for security) and useful additions (e.g. auto links to datasets etc. e.g. `dataset:river-quality`).

Example:

```
/api/util/markdown?q=<http://ibm.com/>
```

Returns:

```
"<p><a href="http://ibm.com/" target="_blank" rel="nofollow">http://ibm.com/</a>\n</p>"
```

is slug valid Checks a name is valid for a new dataset (package) or group, with respect to it being used already.

Example:

```
/api/2/util/is_slug_valid?slug=river-quality&type=package
```

Response:

```
{"valid": true}
```

munge package name For taking an readable identifier and munging it to ensure it is a valid dataset id. Symbols and whitespeace are converted into dashes. Example:

```
/api/util/dataset/munge_name?name=police%20spending%20figures%202009
```

Returns:

```
"police-spending-figures-2009"
```

munge title to package name For taking a title of a package and munging it to a readable and valid dataset id. Symbols and whitespeace are converted into dashes, with multiple dashes collapsed. Ensures that long titles with a year at the end preserves the year should it need to be shortened. Example:

```
/api/util/dataset/munge_title_to_name?title=police:%20spending%20figures%202009
```

Returns:

```
"police-spending-figures-2009"
```

munge tag For taking a readable word/phrase and munging it to a valid tag (name). Symbols and whitespeace are converted into dashes. Example:

```
/api/util/tag/munge?tag=water%20quality
```

Returns:

```
"water-quality"
```

4.2 CKAN API: Quickstart Tutorial

A quickstart tutorial for the CKAN API that walks through some of the main features. For full details of the API see *CKAN API* and its *API: Reference* section.

The following examples use the demonstration instance at http://test.ckan.org/. You may obviously replace this with your own site.

Details of tools for accessing the API including client libraries can be found in the *main API section here*.

4.2.1 Example queries

Search

http://test.ckan.org/api/search/dataset?q=open+street+map

```
{"count": 4, "results": ["uk-naptan-osm", "osm-uk", "osm", "naptan"]}
```

Get dataset

http://test.ckan.org/api/rest/dataset/osm

```
{"id": "a3dd8f64-9078-4f04-845c-e3f047125028", "name": "osm", "title": "Open Street Map", ...
```

Create a dataset

Note: You'll need an *api key*.

```
curl http://test.ckan.org/api/rest/dataset -d '{"name":"test", "title":"Test dataset"}' -H "Authoriza
```

Update a dataset

(Using POST or PUT):

```
curl http://test.ckan.org/api/rest/dataset/test -d '{"name":"test", "title":"Changed Test dataset"}'
```

Viewing permissions

To view authorization roles on a dataset:

```
curl http://test.ckan.org/api/action/roles_show -d '{"domain_object": "freshwateratlasrivers"}'
```

```
{
  "help": "Returns the roles that users (and authorization groups) have on a\n    particular domain_
  "result": {
    "domain_object_id": "9da77628-2ac5-4965-af12-c7c51cc1d99a",
    "domain_object_type": "Package",
    "roles": [
      {
        "authorized_group_id": null,
        "context": "Package",
```

```
        "id": "481b6cd8-350b-4599-bd20-5e3c0ed0a8cb",
        "package_id": "9da77628-2ac5-4965-af12-c7c51cc1d99a",
        "role": "editor",
        "user_id": "4229c297-fe28-4597-a191-3ebbbee6c47a",
        "user_object_role_id": "481b6cd8-350b-4599-bd20-5e3c0ed0a8cb"
      },
      {
        "authorized_group_id": null,
        "context": "Package",
        "id": "aba38fa7-2fb4-4f84-98e1-02cb76c5d95a",
        "package_id": "9da77628-2ac5-4965-af12-c7c51cc1d99a",
        "role": "admin",
        "user_id": "e7f30c0d-944b-4a69-84c4-61b08bbf6b98",
        "user_object_role_id": "aba38fa7-2fb4-4f84-98e1-02cb76c5d95a"
      },
      {
        "authorized_group_id": null,
        "context": "Package",
        "id": "e06b1293-86ec-4417-8e28-b9499161348e",
        "package_id": "9da77628-2ac5-4965-af12-c7c51cc1d99a",
        "role": "reader",
        "user_id": "41cb1162-3d61-4b16-a3af-4cae27836ac5",
        "user_object_role_id": "e06b1293-86ec-4417-8e28-b9499161348e"
      }
    ]
  },
  "success": true
}
```


Looking at the list of "roles" we can see who has what permissions on this dataset. User with id="4229..." is an "editor", id="e7f3..." is an "admin", "41cb..." is a "reader". By using the user_show call to reveal the names of the users, we see that "visitor" (i.e. anyone who is not logged in) is the "reader" (no write permission), "logged-in" (any logged-in user) is the "editor" and the admin is "OKFN" who is the user who created this dataset in the first place and can therefore confer permissions to other users.

Adding permissions

To give user "dread" the "admin" authorization role on dataset "freshwateratlasrivers":

```
curl http://test.ckan.org/api/action/user_role_update -d '{"user": "dread", "domain_object": "freshwa
```

4.2.2 Javascript examples

See http://okfnlabs.org/ckanjs/ (demo search and count widgets)

4.3 Using the Data API

The following provides an introduction to using the CKAN *DataStore* Data API.

4.3.1 Introduction

Each 'table' in the DataStore is an ElasticSearch index type ('table'). As such the Data API for each CKAN resource is directly equivalent to a single index 'type' in ElasticSearch (we tend to refer to it as a 'table').

This means you can (usually) directly re-use ElasticSearch client libraries when connecting to a Data API endpoint. It also means that what follows is, in essence, a tutorial in using the ElasticSearch API.

The following short set of slides provide a brief overview and introduction to the DataStore and the Data API.

4.3.2 Quickstart

`{{endpoint}}` refers to the data API endpoint (or ElasticSearch index / table). For example, on the DataHub this gold prices data resource http://datahub.io/dataset/gold-prices/resource/b9aae52b-b082-4159-b46f-7bb9c158d013 would have its Data API endpoint at: http://datahub.io/api/data/b9aae52b-b082-4159-b46f-7bb9c158d013. If you were just using ElasticSearch standalone an example of an endpoint would be: http://localhost:9200/gold-prices/monthly-price-table.

Note: every resource on a CKAN instance for which a DataStore table is enabled provides links to its Data API endpoint via the Data API button at the top right of the resource page.

Key urls:

- Query: `{{endpoint}}/_search` (in ElasticSearch < 0.19 this will return an error if visited without a query parameter)
  - Query example: `{{endpoint}}/_search?size=5&pretty=true`
- Schema (Mapping): `{{endpoint}}/_mapping`

Examples

cURL (or Browser)

The following examples utilize the cURL command line utility. If you prefer, you you can just open the relevant urls in your browser:

```
// query for documents / rows with title field containing 'jones'
// added pretty=true to get the json results pretty printed
curl {{endpoint}}/_search?q=title:jones&size=5&pretty=true
```

Adding some data (requires an *API Key*):

```
// requires an API key
// Data (argument to -d) should be a JSON document
curl -X POST -H "Authorization: {{YOUR-API-KEY}}" {{endpoint}} -d '{
  "title": "jones",
  "amount": 5.7
}'
```

Javascript

A simple ajax (JSONP) request to the data API using jQuery:

```
var data = {
  size: 5 // get 5 results
  q: 'title:jones' // query on the title field for 'jones'
};
$.ajax({
  url: {{endpoint}}/_search,
  dataType: 'jsonp',
  success: function(data) {
    alert('Total results found: ' + data.hits.total)
  }
});
```

The Data API supports CORs so you can also write to it (this requires the json2 library for `JSON.stringify`):

```
var data = {
  title: 'jones',
  amount: 5.7
};
$.ajax({
  url: {{endpoint}},
  type: 'POST',
  data: JSON.stringify(data),
  success: function(data) {
    alert('Uploaded ok')
  }
});
```

Python

Note: You can also use the DataStore Python client library.

```
import urllib2
import json

# ==================================
# Store data in the DataStore table

url = '{{endpoint}}'
data = {
    'title': 'jones',
    'amount': 5.7
    }
# have to send the data as JSON
data = json.dumps(data)
# need to add your API key (and have authorization to write to this endpoint)
headers = {'Authorization': 'YOUR-API-KEY'}

req = urllib2.Request(url, data, headers)
out = urllib2.urlopen(req)
print out.read()

# ========================
# Query the DataStore table

url = '{{endpoint}}/_search?q=title:jones&size=5'
req = urllib2.Request(url)
```

```
out = urllib2.urlopen(req)
data = out.read()
print data
# returned data is JSON
data = json.loads(data)
# total number of results
print data['hits']['total']
```

4.3.3 Querying

Basic Queries Using Only the Query String

Basic queries can be done using only query string parameters in the URL. For example, the following searches for text 'hello' in any field in any document and returns at most 5 results:

```
{{endpoint}}/_search?q=hello&size=5
```

Basic queries like this have the advantage that they only involve accessing a URL and thus, for example, can be performed just using any web browser. However, this method is limited and does not give you access to most of the more powerful query features.

Basic queries use the *q* query string parameter which supports the Lucene query parser syntax and hence filters on specific fields (e.g. *fieldname:value*), wildcards (e.g. *abc**) and more.

There are a variety of other options (e.g. size, from etc) that you can also specify to customize the query and its results. Full details can be found in the ElasticSearch URI request docs.

Full Query API

More powerful and complex queries, including those that involve faceting and statistical operations, should use the full ElasticSearch query language and API.

In the query language queries are written as a JSON structure and is then sent to the query endpoint (details of the query langague below). There are two options for how a query is sent to the search endpoint:

1. Either as the value of a source query parameter e.g.:

   ```
   {{endpoint}}/_search?source={Query-as-JSON}
   ```

2. Or in the request body, e.g.:

   ```
   curl -XGET {{endpoint}}/_search -d 'Query-as-JSON'
   ```

   For example:

   ```
   curl -XGET {{endpoint}}/_search -d '{
       "query" : {
           "term" : { "user": "kimchy" }
       }
   }'
   ```

4.3.4 Query Language

Queries are JSON objects with the following structure (each of the main sections has more detail below):

```
{
    size: # number of results to return (defaults to 10)
    from: # offset into results (defaults to 0)
    fields: # list of document fields that should be returned - http://elasticsearch.org/guide/refere
    sort: # define sort order - see http://elasticsearch.org/guide/reference/api/search/sort.html

    query: {
        # "query" object following the Query DSL: http://elasticsearch.org/guide/reference/query-dsl/
        # details below
    },

    facets: {
        # facets specifications
        # Facets provide summary information about a particular field or fields in the data
    }

    # special case for situations where you want to apply filter/query to results but *not* to facets
    filter: {
        # filter objects
        # a filter is a simple "filter" (query) on a specific field.
        # Simple means e.g. checking against a specific value or range of values
    },
}
```

Query results look like:

```
{
    # some info about the query (which shards it used, how long it took etc)
    ...
    # the results
    hits: {
        total: # total number of matching documents
        hits: [
            # list of "hits" returned
            {
                _id: # id of document
                score: # the search index score
                _source: {
                    # document 'source' (i.e. the original JSON document you sent to the index
                }
            }
        ]
    }
    # facets if these were requested
    facets: {
        ...
    }
}
```

Query DSL: Overview

Query objects are built up of sub-components. These sub-components are either basic or compound. Compound sub-components may contains other sub-components while basic may not. Example:

```
{
    "query": {
        # compound component
```

```
        "bool": {
            # compound component
            "must": {
                # basic component
                "term": {
                    "user": "jones"
                }
            }
            # compound component
            "must_not": {
                # basic component
                "range" : {
                    "age" : {
                        "from" : 10,
                        "to" : 20
                    }
                }
            }
        }
    }
}
```

In addition, and somewhat confusingly, ElasticSearch distinguishes between sub-components that are "queries" and those that are "filters". Filters, are really special kind of queries that are: mostly basic (though boolean compounding is alllowed); limited to one field or operation and which, as such, are especially performant.

Examples, of filters are (full list on RHS at the bottom of the query-dsl page):

- term: filter on a value for a field
- range: filter for a field having a range of values (>=, <= etc)
- geo_bbox: geo bounding box
- geo_distance: geo distance

Rather than attempting to set out all the constraints and options of the query-dsl we now offer a variety of examples.

Examples

Match all / Find Everything

```
{
    "query": {
        "match_all": {}
    }
}
```

Classic Search-Box Style Full-Text Query

This will perform a full-text style query across all fields. The query string supports the Lucene query parser syntax and hence filters on specific fields (e.g. *fieldname:value*), wildcards (e.g. *abc**) as well as a variety of options. For full details see the query-string documentation.

```
{
    "query": {
        "query_string": {
```

```
            "query": {query string}
        }
    }
}
```

Filter on One Field

```
{
    "query": {
        "term": {
            {field-name}: {value}
        }
    }
}
```

High performance equivalent using filters:

```
{
    "query": {
        "constant_score": {
            "filter": {
                "term": {
                    # note that value should be *lower-cased*
                    {field-name}: {value}
                }
            }
        }
}
```

Find all documents with value in a range

This can be used both for text ranges (e.g. A to Z), numeric ranges (10-20) and for dates (ElasticSearch will converts dates to ISO 8601 format so you can search as 1900-01-01 to 1920-02-03).

```
{
    "query": {
        "constant_score": {
            "filter": {
                "range": {
                    {field-name}: {
                        "from": {lower-value}
                        "to": {upper-value}
                    }
                }
            }
        }
    }
}
```

For more details see range filters.

Full-Text Query plus Filter on a Field

```
{
    "query": {
        "query_string": {
            "query": {query string}
        },
        "term": {
            {field}: {value}
        }
    }
}
```

Filter on two fields

Note that you cannot, unfortunately, have a simple and query by adding two filters inside the query element. Instead you need an 'and' clause in a filter (which in turn requires nesting in 'filtered'). You could also achieve the same result here using a bool query.

```
{
    "query": {
        "filtered": {
            "query": {
                "match_all": {}
            },
            "filter": {
                "and": [
                    {
                        "range" : {
                            "b" : {
                                "from" : 4,
                                "to" : "8"
                            }
                        },
                    },
                    {
                        "term": {
                            "a": "john"
                        }
                    }
                ]
            }
        }
    }
}
```

Geospatial Query to find results near a given point

This uses the Geo Distance filter. It requires that indexed documents have a field of geo point type.

Source data (a point in San Francisco!):

```
# This should be in lat,lon order
{
  ...
```

```
  "Location": "37.7809035011582, -122.412119695795"
}
```

There are alternative formats to provide lon/lat locations e.g. (see ElasticSearch documentation for more):

```
# Note this must have lon,lat order (opposite of previous example!)
{
  "Location":[-122.414753390488, 37.7762147914147]
}

# or ...
{
  "Location": {
    "lon": -122.414753390488,
    "lat": 37.7762147914147
  }
}
```

We also need a mapping to specify that Location field is of type geo_point as this will not usually get guessed from the data (see below for more on mappings):

```
"properties": {
    "Location": {
        "type": "geo_point"
     }
     ...
}
```

Now the actual query:

```
{
    "query": {
        "filtered" : {
            "query" : {
                "match_all" : {}
            },
            "filter" : {
                "geo_distance" : {
                    "distance" : "20km",
                    "Location" : {
                        "lat" : 37.776,
                        "lon" : -122.41
                    }
                }
            }
        }
    }
}
```

Note that you can specify the query using specific lat, lon attributes even though original data did not have this structure (you can also use a query similar to the original structure if you wish - see Geo distance filter for more information).

Facets

Facets provide a way to get summary information about then data in an elasticsearch table, for example counts of distinct values.

ElasticSearch (and hence the Data API) provides rich faceting capabilities: http://www.elasticsearch.org/guide/reference/api/search/facets/

There are various kinds of facets available, for example (full list on the facets page):

- Terms - counts by distinct terms (values) in a field
- Range - counts for a given set of ranges in a field
- Histogram and Date Histogram - counts by constant interval ranges
- Statistical - statistical summary of a field (mean, sum etc)
- Terms Stats - statistical summary on one field (stats field) for distinct terms in another field. For example, spending stats per department or per region.
- Geo Distance: counts by distance ranges from a given point

Note that you can apply multiple facets per query.

4.3.5 Adding, Updating and Deleting Data

ElasticSeach, and hence the Data API, have a standard RESTful API. Thus:

```
POST       {{endpoint}}          : INSERT
PUT/POST   {{endpoint}}/{{id}}   : UPDATE (or INSERT)
DELETE     {{endpoint}}/{{id}}   : DELETE
```

For more on INSERT and UPDATE see the Index API documentation.

There is also support bulk insert and updates via the Bulk API.

Note: The DataStore Python client library has support for inserting, updating (in bulk) and deleting. There is also support for these operations in the ReclineJS javascript library.

4.3.6 Schema Mapping

As the ElasticSearch documentation states:

> Mapping is the process of defining how a document should be mapped to the Search Engine, including its searchable characteristics such as which fields are searchable and if/how they are tokenized. In ElasticSearch, an index may store documents of different "mapping types". ElasticSearch allows one to associate multiple mapping definitions for each mapping type.
>
> Explicit mapping is defined on an index/type level. By default, there isn't a need to define an explicit mapping, since one is automatically created and registered when a new type or new field is introduced (with no performance overhead) and have sensible defaults. Only when the defaults need to be overridden must a mapping definition be provided.

Relevant docs: http://elasticsearch.org/guide/reference/mapping/.

4.3.7 JSONP support

JSONP support is available on any request via a simple callback query string parameter:

```
?callback=my_callback_name
```

CHAPTER 5

General Administration

5.1 CKAN Administrative Dashboard

CKAN provides an administrative dashboard available to Sysadmin Administrators. The dashboard allows you to:

- Create and remove sysadmins
- Edit general system level authorization
- Manage the 'trash' bin (i.e. datasets or revisions that have been marked as deleted)

The dashboard is located, relative to your site root, at `/ckan-admin/`.

Note: To create your first sysadmin you cannot use Dashboard as you will not yet have access! Instead create a sysadmin using the command line `paster` by running the following command:

```
paster sysadmin -h
```

5.1.1 Setting System-Level Roles

Authorization interface is located at: `/ckan-admin/authz`

This page allows you to see and change the users and authorization groups who have 'roles' on the 'System Object'. In a standard installation, there are four 'roles' which a user can have on the System (or on any object):

- admin (administrator)
  - Having an admin role on the System objects means you are a System Administrator and may carry out **any** operation on any object.

    Warning: Once a person is an system administrator, they can carry out **any** operatoin on the system including **destructive** ones. Grant System Administrator access with care!
- reader (Read action allowed)
  - Without read access a site user or visitor will not be able to see anything except the login page, even the page which allows them to create an account, so they're locked out forever unless they already have a valid account.
- editor (Update action allowed)
- anon-editor

Note: these roles can be applied to users on your system as well as to 'pseudo-users' like 'visitor', which stands for anyone who accesses the site whether logged in or not (see *Set and Manage Permissions* for more on permissions and roles).

5.1.2 Make Someone a Sysadmin

Given the user the role 'admin'.

5.1.3 The Trash

When you delete datasets or revisions they go into the 'trash'. The contents of the trash can be viewed by System Administrators at: `/ckan-admin/trash`.

Contents of the trash can be removed permanently (and **irreversibly**) by going to the trash page and selecting the purge option.

5.2 Common CKAN Administrator Tasks

The majority of common CKAN administration tasks are carried out using the **paster** script.

Paster is run on the command line on the server running CKAN. This section covers:

- *Understanding Paster*. Understanding paster syntax and getting help.
- *Common Tasks Using Paster*. How to carry out common CKAN admin tasks using paster.

5.2.1 Understanding Paster

At its simplest, paster commands can be thought of like this:

```
paster <ckan commands>
```

But there are various extra elements to the commandline that usually need adding. We shall build them up:

5.2.2 Enabling CKAN commands

Paster is used for many things aside from CKAN. You usually need to tell paster that you want to enable the CKAN commands:

```
paster --plugin=ckan <ckan commands>
```

You know you need to do this if you get the error `Command 'user' not known` for a valid CKAN command.

(Alternatively, CKAN commands are enabled by default if your current directory is the CKAN source directory)

5.2.3 Pointing to your CKAN config

Paster needs to know where your CKAN config file is (so it knows which database and search index to deal with etc.):

```
paster --plugin=ckan <ckan commands> --config=<config file>
```

If you forget to specify `--config` then you will get error `AssertionError: Config filename '/home/okfn/development.ini' does not exist.`

(Paster defaults to looking for development.ini in the current directory.)

For example, to initialise a database:

```
paster --plugin=ckan db init --config=/etc/ckan/std/std.ini
```

5.2.4 Virtual environments

You often need to run paster within your CKAN virtual environment (pyenv). If CKAN was installed as 'source' then you can activate it as usual before running the paster command:

```
. ~/pyenv/bin/activate
paster --plugin=ckan db init --config=/etc/ckan/std/std.ini
```

The alternative, which also suits a CKAN 'package' install, is to simply give the full path to the paster in your pyenv:

```
/var/lib/ckan/std/pyenv/bin/paster --plugin=ckan db init --config=/etc/ckan/std/std.ini
```

5.2.5 Running Paster on a deployment

If CKAN is deployed with Apache on this machine, then you should run paster as the same user, which is usually `www-data`. This is because paster will write to the same CKAN logfile as the Apache process and file permissions need to match.

For example:

```
sudo -u www-data /var/lib/ckan/std/pyenv/bin/paster --plugin=ckan db init --config=/etc/ckan/std
```

Otherwise you will get an error such as: `IOError: [Errno 13] Permission denied: '/var/log/ckan/std/std.log'.`

Getting Help on Paster

To get a full list of paster commands (i.e. including CKAN commands):

```
paster --plugin=ckan --help
```

And to get more detailed help on each command (e.g. on `db`):

```
paster --plugin=ckan --help db
```

Paster executable

It is essential to run the correct paster. The program may be installed globally on a server, but in nearly all cases, the one installed in the CKAN python virtual environment (pyenv) is the one that should be used instead. This can be done by either:

1. Activating the virtual environment:

```
. pyenv/bin/activate
```

2. Giving the path to paster when you run it:

```
pyenv/bin/paster ...
```

Position of Paster Parameters

The position of paster parameters matters.

`--plugin` is a parameter to paster, so needs to come before the CKAN command. To do this, the first parameter to paster is normally `--plugin=ckan`.

Note: The default value for `--plugin` is `setup.py` in the current directory. If you are running paster from the directory where CKAN's `setup.py` file is located, you don't need to specify the plugin parameter..

Meanwhile, `--config` is a parameter to CKAN, so needs to come after the CKAN command. This specifies the CKAN config file for the instance you want to use, e.g. `--config=/etc/ckan/std/std.ini`

Note: The default value for `--config` is `development.ini` in the current directory. If you are running a package install of CKAN (as described in *Option 1: Package Installation*), you should explicitly specify `std.ini`.

The position of the CKAN command itself is less important, as longs as it follows `--plugin`. For example, both the following commands have the same effect::

```
paster --plugin=ckan db --config=development.ini init
paster --plugin=ckan db init --config=development.ini
```

Running a Paster Shell

If you want to run a "paster shell", which can be useful for development, then the plugin is pylons. e.g. `paster --plugin=pylons shell`.

Often you will want to run this as the same user as the web application, to ensure log files are written as the same user. And you'll also want to specify a config file (note that this is not specified using the `--config` parameter, but simply as the final argument). For example:

```
sudo -u www-data paster --plugin=pylons shell std.ini
```

5.2.6 Common Tasks Using Paster

The following tasks are supported by paster.

create-test-data	Create test data in the database.
db	Perform various tasks on the database.
ratings	Manage the ratings stored in the db
rights	Commands relating to per-object and system-wide access rights.
roles	Commands relating to roles and actions.
search-index	Creates a search index for all datasets
sysadmin	Gives sysadmin rights to a named user
user	Manage users

For the full list of tasks supported by paster, you can run:

```
paster --plugin=ckan --help
```

create-test-data: Create test data

As the name suggests, this command lets you load test data when first setting up CKAN. See *Load Test Data* for details.

db: Manage databases

Lets you initialise, upgrade, and dump the CKAN database.

Initialisation

Before you can run CKAN for the first time, you need to run "db init" to create the tables in the database and the default authorization settings:

```
paster --plugin=ckan db init --config=/etc/ckan/std/std.ini
```

If you forget to do this then CKAN won't serve requests and you will see errors such as this in the logs:

```
ProgrammingError: (ProgrammingError) relation "user" does not exist
```

Cleaning

You can delete everything in the CKAN database, including the tables, to start from scratch:

```
paster --plugin=ckan db clean --config=/etc/ckan/std/std.ini
```

The next logical step from this point is to do a "db init" step before starting CKAN again.

Dumping and Loading databases to/from a file

You can 'dump' (save) the exact state of the database to a file on disk and at a later point 'load' (restore) it again, or load it on another machine.

To write the dump:

```
paster --plugin=ckan db dump --config=/etc/ckan/std/std.ini std.pg_dump
```

To load it in again, you first have to clean the database of existing data (be careful not to wipe valuable data), followed by the load:

```
paster --plugin=ckan db clean --config=/etc/ckan/std/std.ini std.pg_dump
paster --plugin=ckan db load --config=/etc/ckan/std/std.ini std.pg_dump
```

The pg_dump file notes which PostgreSQL user 'owns' the data on the server. Because the PostgreSQL user (by default) is identified as the current Linux user, and this is setup to be `ckanINSTANCE` where `INSTANCE` is the name of the CKAN instance. This means if you want to restore the pg_dump as another CKAN instance name (often needed if you move it to another server) then you will need to change the database owner - see `howto-editing-database-ownership`.

Upgrade migration

When you upgrade CKAN software by any method *other* than the package update described in *Option 1: Package Installation*, before you restart it, you should run 'db upgrade', which will do any necessary migrations to the database tables:

```
paster --plugin=ckan db upgrade --config=/etc/ckan/std/std.ini
```

Creating dump files

For information on using `db` to create dumpfiles, see *Database Dumps*.

ratings: Manage dataset ratings

Manages the ratings stored in the database, and can be used to count ratings, remove all ratings, or remove only anonymous ratings.

For example, to remove anonymous ratings from the database:

```
paster --plugin=ckan ratings clean-anonymous --config=/etc/ckan/std/std.ini
```

rights: Set user permissions

Sets the authorization roles of a specific user on a given object within the system.

For example, to give the user named 'bar' the 'admin' role on the dataset 'foo':

```
paster --plugin=ckan rights make bar admin package:foo  --config=/etc/ckan/std/std.ini
```

To list all the rights currently specified:

```
paster --plugin=ckan rights list --config=/etc/ckan/std/std.ini
```

For more information and examples, see *Set and Manage Permissions*.

roles: Manage system-wide permissions

This important command gives you fine-grained control over CKAN permissions, by listing and modifying the assignment of actions to roles.

The `roles` command has its own section: see *Set and Manage Permissions*.

search-index: Rebuild search index

Rebuilds the search index. This is useful to prevent search indexes from getting out of sync with the main database.

For example:

```
paster --plugin=ckan search-index rebuild --config=/etc/ckan/std/std.ini
```

This default behaviour will clear the index and rebuild it with all datasets. If you want to rebuild it for only one dataset, you can provide a dataset name:

```
paster --plugin=ckan search-index rebuild test-dataset-name --config=/etc/ckan/std/std.ini
```

Alternatively, you can use the *-o* or *–only-missing* option to only reindex datasets which are not already indexed:

```
paster --plugin=ckan search-index rebuild -o --config=/etc/ckan/std/std.ini
```

If you don't want to rebuild the whole index, but just refresh it, use the *-r* or *–refresh* option. This won't clear the index before starting rebuilding it:

```
paster --plugin=ckan search-index rebuild -r --config=/etc/ckan/std/std.ini
```

There are other search related commands, mostly useful for debugging purposes:

```
search-index check                  - checks for datasets not indexed
search-index show {dataset-name}    - shows index of a dataset
search-index clear [dataset-name]   - clears the search index for the provided dataset or for the wh
```

sysadmin: Give sysadmin rights

Gives sysadmin rights to a named user. This means the user can perform any action on any object.

For example, to make a user called 'admin' into a sysadmin:

```
paster --plugin=ckan sysadmin add admin --config=/etc/ckan/std/std.ini
```

user: Create and manage users

Lets you create, remove, list and manage users.

For example, to create a new user called 'admin':

```
paster --plugin=ckan user add admin --config=/etc/ckan/std/std.ini
```

To delete the 'admin' user:

```
paster --plugin=ckan user remove admin --config=/etc/ckan/std/std.ini
```

5.3 Database Dumps

It's often useful to allow users to download a complete CKAN database in a dumpfile.

In addition, a CKAN administrator would like to easily backup and restore a CKAN database.

5.3.1 Creating a Dump

We provide two `paster` methods to create dumpfiles.

- `db simple-dump-json` - A simple dumpfile, useful to create a public listing of the datasets with no user information. All datasets are dumped, including deleted datasets and ones with strict authorization. These may be in JSON or CSV format.
- `db dump` - A more complicated dumpfile, useful for backups. Replicates the database completely, including users, their personal info and API keys, and hence should be kept private. This is in the format of SQL commands.

For more information on paster, see *Common CKAN Administrator Tasks*.

Using db simple-dump-json

If you are using a Python environment, as part of a development installation, first enable the environment:

```
. /home/okfn/var/srvc/ckan.net/pyenv/bin/activate || exit 1
```

Then create and zip the dumpfile:

```
paster --plugin=ckan db simple-dump-json /var/srvc/ckan/dumps/ckan.net-daily.json --config=/etc/ckan/
gzip /var/srvc/ckan/dumps/ckan.net-daily.json
```

Change `simple-dump-json` to `simple-dump-csv` if you want CSV format instead of JSON.

Backing up - db dump

If you are using a Python environment, as part of a development installation, first enable the environment:

```
. /var/srvc/ckan/pyenv/bin/activate || exit 1
```

Then create and zip the dumpfile:

```
paster --plugin=ckan db dump /var/srvc/ckan/dumps/ckan.net-daily --config=/etc/ckan/std/std.ini
gzip /var/srvc/ckan/dumps/ckan.net-daily
```

Restoring a database - db load

To restore the dump to the database, us `db load`.

You either need a freshly created database (i.e. using createdb) or take the existing one and clean (wipe) it:

```
paster --plugin=ckan db clean --config=/etc/ckan/std/std.ini
```

Now you can 'db load' the dump file:

```
paster --plugin=ckan db load /var/srvc/ckan/dumps/ckan.net-daily --config=/etc/ckan/std/std.ini
```

5.3.2 Daily Dumps

You can set the dump(s) to be created daily with a cron job.

Edit your user's cron config:

```
$ crontab -e
```

Now add a line such as this:

```
0 21 * * * /home/okfn/var/srvc/ckan.net/dump.sh
```

Now create the dump.sh to contain the paster db dump command from above.

5.3.3 Serving the Files

Some simple additions to the Apache config can serve the files to users in a directory listing. This is ideal for the JSON/CSV simple dumps, but obviously not ideal for the SQL dump containing private user information.

To do this, add these lines to your virtual host config (e.g. `/etc/apache2/sites-enabled/ckan.net`):

```
Alias /dump/ /home/okfn/var/srvc/ckan.net/dumps/

# Disable the mod_python handler for static files
<Location /dump>
    SetHandler None
    Options +Indexes
</Location>
```

5.4 Reference: CKAN Configuration Options

You can change many important CKAN settings in the CKAN config file. This is the file called `std.ini` that you first encountered in *Create an Admin User*. It is usually located at `/etc/ckan/std/std.ini`.

The file is well-documented, but we recommend reading this section in full to learn about the CKAN config options available to you.

Note: After editing this file, you will need to restart Apache for the changes to take effect.

Note: The CKAN config file also includes general Pylons options. All CKAN-specific settings are in the *[app:main]* section.

5.4.1 Database Settings

sqlalchemy.url

Example:

```
sqlalchemy.url = postgres://tester:pass@localhost/ckantest3
```

This defines the database that CKAN is to use. The format is:

```
sqlalchemy.url = postgres://USERNAME:PASSWORD@HOST/DBNAME
```

5.4.2 Front-End Settings

site_description

Example:

```
ckan.site_description=
```

Default value: (none)

This is for a description, or tag line for the site, as displayed in the header of the CKAN web interface.

site_logo

Example:

```
ckan.site_logo = /images/ckan_logo_fullname_long.png
```

Default value: (none)

This sets the logo used in the title bar.

favicon

Example:

```
ckan.favicon = http://okfn.org/wp-content/themes/okfn-master-wordpress-theme/images/favicon.ico
```

Default value: `/images/icons/ckan.ico`

This sets the site's *favicon*. This icon is usually displayed by the browser in the tab heading and bookmark.

site_about

Example:

```
ckan.site_about=${g.site_title} is a community-driven catalogue of open data for the Greenfield area
```

Default value:

```
What was the <a href="http://thedatahub.org/dataset/house-prices-uk-from-1930">average price</a> of a

 <p i18n:msg="">${g.site_title} is a community-run catalogue of useful sets of data on the Internet.
```

This changes the text about the site on the 'About' page. i.e. replaces the text in the "About <site_name" section. The other sections of the About page are not affected.

Format tips:

- multiline strings can be used by indenting following lines
- the format is basically HTML, but with Genshi-format strings
- the about text will be automatically be placed with-in paragraph tags `<p>...</p>` but you can start new paragraphs within that by using `</p><p>`

Note: Whilst the default text is translated into many languages (switchable in the page footer), the text in this configuration option will not be translatable.

---

package_hide_extras

Example:

```
package_hide_extras = my_private_field other_field
```

Default value: (empty)

This sets a space-separated list of extra field key values which will not be shown on the dataset read page.

> **Warning:** While this is useful to e.g. create internal notes, it is not a security measure. The keys will still be available via the API and in revision diffs.

rdf_packages

Example:

```
rdf_packages = http://semantic.ckan.net/record/
```

Configure this if you have an RDF store of the same datasets as are in your CKAN instance. It will provide three sorts of links from each dataset page to the equivalent RDF URL given in *rdf_packages*:

1. 303 redirects for clients that content-negotiate rdf-xml or turtle. e.g. client GETs *http://ckan.net/dataset/pollution-2008* with accept header *application/rdf+xml* `curl -H "Accept: application/rdf+xml" http://ckan.net/dataset/pollution-2008`. CKAN's response is a 303 redirect to *http://semantic.ckan.net/dataset/pollution-2008* which can be obtained with: `curl -L -H "Accept: application/rdf+xml" http://ckan.net/dataset/pollution-2008`
2. Embedded links for browsers that are aware. e.g. *<link rel="alternate" type="application/rdf+xml" href="http://semantic.ckan.net/record/b410e678-8a96-40cf-8e46-e8bd4bf02684.rdf">*
3. A visible RDF link on the page. e.g. *<a href="http://semantic.ckan.net/record/b410e678-8a96-40cf-8e46-e8bd4bf02684.rdf">*

dumps_url & dumps_format

Example:

```
ckan.dumps_url = http://ckan.net/dump/
ckan.dumps_format = CSV/JSON
```

If there is a page which allows you to download a dump of the entire catalogue then specify the URL and the format here, so that it can be advertised in the web interface. `dumps_format` is just a string for display.

For more information on using dumpfiles, see *Database Dumps*.

recaptcha

Example:: ckan.recaptcha.publickey = 6Lc...-KLc ckan.recaptcha.privatekey = 6Lc...-jP

Setting both these options according to an established Recaptcha account adds captcha to the user registration form. This has been effective at preventing bots registering users and creating spam packages.

To get a Recaptcha account, sign up at: http://www.google.com/recaptcha

And there is an option for the default expiry time if not specified:

```
ckan.cache.default_expires = 600
```

datasets_per_page

Example:

```
ckan.datasets_per_page = 10
```

Default value: `20`

This controls the pagination of the dataset search results page. This is the maximum number of datasets viewed per page of results.

5.4.3 Authentication Settings

openid_enabled

Example:

```
openid_enabled = False
```

Default value: `True`

CKAN operates a delegated authentication model based on OpenID.

Setting this option to False turns off OpenID for login.

5.4.4 Internationalisation Settings

ckan.locale_default

Example:

```
ckan.locale_default=de
```

Default value: `en` (English)

Use this to specify the locale (language of the text) displayed in the CKAN Web UI. This requires a suitable *mo* file installed for the locale in the ckan/i18n. For more information on internationalization, see *Internationalize CKAN*. If you don't specify a default locale, then it will default to the first locale offered, which is by default English (alter that with *ckan.locales_offered* and *ckan.locales_filtered_out*.

ckan.locales_offered

Example:

```
ckan.locales_offered=en de fr
```

Default value: (none)

By default, all locales found in the ckan/i18n directory will be offered to the user. To only offer a subset of these, list them under this option. The ordering of the locales is preserved when offered to the user.

ckan.locales_filtered_out

Example:

```
ckan.locales_filtered_out=pl ru
```

Default value: (none)

If you want to not offer particular locales to the user, then list them here to have them removed from the options.

ckan.locale_order

Example:

```
ckan.locale_order=fr de
```

Default value: (none)

If you want to specify the ordering of all or some of the locales as they are offered to the user, then specify them here in the required order. Any locales that are available but not specified in this option, will still be offered at the end of the list.

5.4.5 Storage Settings

ckan.storage.bucket

Example:

```
ckan.storage.bucket = ckan
```

Default value: `None`

This setting will change the bucket name for the uploaded files.

ofs.storage_dir

Example:

```
ofs.storage_dir = /data/uploads/
```

Default value: `None`

Use this to specify where uploaded files should be stored, and also to turn on the handling of file storage. The folder should exist, and will automatically be turned into a valid pairtree repository if it is not already.

5.4.6 Theming Settings

extra_template_paths

Example:

```
extra_template_paths=/home/okfn/brazil_ckan_config/templates
```

To customise the display of CKAN you can supply replacements for the Genshi template files. Use this option to specify where CKAN should look for additional templates, before reverting to the `ckan/templates` folder. You can supply more than one folder, separating the paths with a comma (,).

For more information on theming, see *Theming and Customizing Appearance*.

extra_public_paths

Example:

```
extra_public_paths = /home/okfn/brazil_ckan_config/public
```

To customise the display of CKAN you can supply replacements for static files such as HTML, CSS, script and PNG files. Use this option to specify where CKAN should look for additional files, before reverting to the `ckan/public` folder. You can supply more than one folder, separating the paths with a comma (,).

For more information on theming, see *Theming and Customizing Appearance*.

template_head_end

HTML content to be inserted just before `</head>` tag (e.g. extra stylesheet)

Example:

```
ckan.template_head_end = <link rel="stylesheet" href="http://mysite.org/css/custom.css" type="text/cs
```

You can also have multiline strings. Just indent following lines. e.g.:

```
ckan.template_head_end =
 <link rel="stylesheet" href="/css/extra1.css" type="text/css">
 <link rel="stylesheet" href="/css/extra2.css" type="text/css">
```

template_footer_end

HTML content to be inserted just before `</body>` tag (e.g. Google Analytics code).

Note: you can have multiline strings (just indent following lines)

Example (showing insertion of Google Analytics code):

```
ckan.template_footer_end = <!-- Google Analytics -->
  <script src='http://www.google-analytics.com/ga.js' type='text/javascript'></script>
  <script type="text/javascript">
  try {
  var pageTracker = _gat._getTracker("XXXXXXXXX");
  pageTracker._setDomainName(".ckan.net");
  pageTracker._trackPageview();
  } catch(err) {}
  </script>
  <!-- /Google Analytics -->
```

5.4.7 Form Settings

package_form

Example:

```
package_form = ca
```

Default value: `standard`

This sets the name of the Formalchemy form to use when editing a dataset.

Note: This setting only applies to the deprecated Formalchemy forms. For enabling forms defined with a Navl schema, see *Customizing Forms*.

The value for this setting can be a Formalchemy form defined in the core CKAN code or in another setuputils-managed python module. The only requirement is that the `setup.py` file has an entry point for the form defined in the `ckan.forms` section.

For more information on forms, see *Customizing Forms*.

package_new_return_url & package_edit_return_url

Example:

```
package_new_return_url = http://datadotgc.ca/new_dataset_complete?name=<NAME>
package_edit_return_url = http://datadotgc.ca/dataset/<NAME>
```

If integrating the Edit Dataset and New Dataset forms into a third-party interface, setting these options allows you to set the return address. When the user has completed the form and presses 'commit', the user is redirected to the URL specified.

The `<NAME>` string is replaced with the name of the dataset edited. Full details of this process are given in *Form Integration*.

licenses_group_url

A url pointing to a JSON file containing a list of licence objects. This list determines the licences offered by the system to users, for example when creating or editing a dataset.

This is entirely optional - by default, the system will use an internal cached version of the CKAN list of licences available from the http://licenses.opendefinition.org/licenses/groups/ckan.json.

More details about the license objects - including the licence format and some example licence lists - can be found at the Open Licenses Service.

Examples:

```
licenses_group_url = file:///path/to/my/local/json-list-of-licenses.json
licenses_group_url = http://licenses.opendefinition.org/licenses/groups/od.json
```

5.4.8 Messaging Settings

carrot_messaging_library

Example:

```
carrot_messaging_library=pyamqplib
```

This is the messaging library backend to use. Options:

```
* ''pyamqplib'' - AMQP (e.g. for RabbitMQ)

* ''pika'' - alternative AMQP

* ''stomp'' - python-stomp

* ''queue'' - native Python Queue (default) - NB this doesn't work inter-process
```

See the Carrot documentation for details.

amqp_hostname, amqp_port, amqp_user_id, amqp_password

Example:

```
amqp_hostname=localhost
amqp_port=5672
amqp_user_id=guest
amqp_password=guest
```

These are the setup parameters for AMQP messaging. These only apply if the messaging library has been set to use AMQP (see carrot_messaging_library). The values given above are the default values.

5.4.9 Search Settings

ckan.site_id

Example:

```
ckan.site_id = my_ckan_instance
```

CKAN uses Solr to index and search packages. The search index is linked to the value of the `ckan.site_id`, so if you have more than one CKAN instance using the same solr_url, they will each have a separate search index as long as their `ckan.site_id` values are different. If you are only running a single CKAN instance then this can be ignored.

Note, if you change this value, you need to rebuild the search index.

solr_url

Example:

```
solr_url = http://solr.okfn.org:8983/solr/ckan-schema-1.3
```

Default value: `http://solr.okfn.org:8983/solr`

This configures the Solr server used for search. The Solr schema found at that URL must be one of the ones in `ckan/config/solr` (generally the most recent one). A check of the schema version number occurs when CKAN starts.

Optionally, `solr_user` and `solr_password` can also be configured to specify HTTP Basic authentication details for all Solr requests.

Note, if you change this value, you need to rebuild the search index.

simple_search

Example:

```
ckan.simple_search = true
```

Default value: `false`

Switching this on tells CKAN search functionality to just query the database, (rather than using Solr). In this setup, search is crude and limited, e.g. no full-text search, no faceting, etc. However, this might be very useful for getting up and running quickly with CKAN.

5.4.10 Site Settings

site_title

Example:

```
ckan.site_title=Open Data Scotland
```

Default value: `CKAN`

This sets the name of the site, as displayed in the CKAN web interface.

site_url

Example:

```
ckan.site_url=http://scotdata.ckan.net
```

Default value: (none)

The primary URL used by this site. Used in the API to provide datasets with links to themselves in the web UI.

api_url

Example:

```
ckan.api_url=http://scotdata.ckan.net/api
```

Default value: `/api`

The URL that resolves to the CKAN API part of the site. This is useful if the API is hosted on a different domain, for example when a third-party site uses the forms API.

apikey_header_name

Example:

```
apikey_header_name = API-KEY
```

Default value: `X-CKAN-API-Key` & `Authorization`

This allows another http header to be used to provide the CKAN API key. This is useful if network infrastructure block the Authorization header and `X-CKAN-API-Key` is not suitable.

5.4.11 Authorization Settings

default_roles

This allows you to set the default authorization roles (i.e. permissions) for new objects. Currently this extends to new datasets, groups, authorization groups and the `system` object. For full details of these, see *Set and Manage Permissions*.

The value is a strict JSON dictionary of user names `visitor` (any user who is not logged in) and `logged_in` (any user who is logged in) with lists of their roles.

Example:

```
ckan.default_roles.Package = {"visitor": ["editor"], "logged_in": ["editor"]}
ckan.default_roles.Group = {"visitor": ["reader"], "logged_in": ["reader"]}
```

With this example setting, visitors and logged-in users can only read datasets that get created.

Defaults: see in `ckan/model/authz.py` for: `default_default_user_roles`

5.4.12 Plugin Settings

plugins

Example:

```
ckan.plugins = disqus datapreview googleanalytics follower
```

Specify which CKAN extensions are to be enabled.

> **Warning:** If you specify an extension but have not installed the code, CKAN will not start.

Format as a space-separated list of the extension names. The extension name is the key in the [ckan.plugins] section of the extension's `setup.py`. For more information on extensions, see *Add Extensions*.

5.4.13 Directory Settings

log_dir

Example:

```
ckan.log_dir = /var/log/ckan/
```

This is the directory to which CKAN cron scripts (if there are any installed) should write log files.

Note: This setting is nothing to do with the main CKAN log file, whose filepath is set in the `[handler_file]` args.

---

dump_dir

Example:

```
ckan.dump_dir = /var/lib/ckan/dump/
```

This is the directory to which JSON or CSV dumps of the database are to be written, assuming a script has been installed to do this.

Note: It is usual to set up the Apache config to serve this directory.

---

backup_dir

Example:

```
ckan.backup_dir = /var/backups/ckan/
```

This is a directory where SQL database backups are to be written, assuming a script has been installed to do this.

5.4.14 Compatibility

restrict_template_vars

Example:

```
ckan.restrict_template_vars = true
```

Default value: `false`

This is used to limit the functions available via h in templates. It also forces correct usage of functions as some function signatures have changed. It's main purpose is to allow transition to a cleaner world.

5.5 Common error messages

Whether a developer runs CKAN using paster or going through CKAN test suite, there are a number of error messages seen that are the result of setup problems. As people experience them, please add them to the list here.

These instructions assume you have the python virtual environment enabled (`. pyenv/bin/activate`) and the current directory is the top of the ckan source, which is probably: `../pyenv/src/ckan/`.

5.5.1 nose.config.ConfigError: Error reading config file 'setup.cfg': no such option 'with-pylons'

This error can result when you run nosetests for two reasons:

1. Pylons nose plugin failed to run. If this is the case, then within a couple of lines of running *nosetests* you'll see this warning: *Unable to load plugin pylons* followed by an error message. Fix the error here first.
2. The Python module 'Pylons' is not installed into you Python environment. Confirm this with:

   ```
   python -c "import pylons"
   ```

5.5.2 OperationalError: (OperationalError) no such function: plainto_tsquery ...

This error usually results from running a test which involves search functionality, which requires using a PostgreSQL database, but another (such as SQLite) is configured. The particular test is either missing a *@search_related* decorator or there is a mixup with the test configuration files leading to the wrong database being used.

5.5.3 ImportError: No module named worker

The python entry point for the worker has not been generated. This occurs during the 'pip install' of the CKAN source, and needs to be done again if switching from older code that didn't have it. To recitify it:

```
python setup.py egg_info
```

5.5.4 ImportError: cannot import name get_backend

This can be caused by an out of date pyc file. Delete all your pyc files and start again:

```
find . -name "*.pyc" | xargs rm
```

5.5.5 ImportError: cannot import name UnicodeMultiDict

This is caused by using a version of WebOb that is too new (it has deprecated UnicodeMultiDict). Check the version like this (ensure you have activated your python environment first):

```
pip freeze | grep -i webob
```

Now install the version specified in requires/lucid_present.txt. e.g.:

```
pip install webob==1.0.8
```

5.5.6 nosetests: error: no such option: --ckan

Nose is either unable to find ckan/ckan_nose_plugin.py in the python environment it is running in, or there is an error loading it. If there is an error, this will surface it:

```
nosetests --version
```

There are a few things to try to remedy this:

Commonly this is because the nosetests isn't running in the python environment. You need to have nose actually installed in the python environment. To see which you are running, do this:

```
which nosetests
```

If you have activated the environment and this still reports `/usr/bin/nosetests` then you need to:

```
pip install --ignore-installed nose
```

If `nose --version` still fails, ensure that ckan is installed in your environment:

```
cd pyenv/src/ckan
python setup.py develop
```

One final check - the version of nose should be at least 1.0. Check with:

```
pip freeze | grep -i nose
```

5.5.7 AttributeError: 'unicode' object has no attribute 'items' (Cookie.py)

This can be caused by using repoze.who version 1.0.18 when 1.0.19 is required. Check what you have with:

```
pip freeze | grep -i repoze.who=
```

See what version you need with:

```
grep -f requires/*.txt |grep repoze\.who=
```

Then install the version you need (having activated the environment):

```
pip install repoze.who==1.0.19
```

5.5.8 AttributeError: 'module' object has no attribute 'BigInteger'

The sqlalchemy module version is too old.

5.5.9 ConfigParser.NoSectionError: No section: 'formatters'

This suggests that the config file specified with the paster `--config` parameter (e.g. `myconfig.ini`) is incorrectly formatted. This may be true, but this error is also printed if you specify an incorrect filename for the config file!

5.5.10 ImportError: No module named exceptions

This occurs when trying to `import migrate.exceptions` and is due to the version of sqlalchemy-migrate being used is too old - check the requires files for the version needed.

5.5.11 ckan.plugins.core.PluginNotFoundException: stats

After the CKAN 1.5.1 release, the Stats and Storage extensions were merged into the core CKAN code, and the ckanext namespace needs registering before the tests will run:

```
cd pyenv/src/ckan
python setup.py develop
```

Otherwise, this problem may be because of specifying an extension in the CKAN config but having not installed it. See: *Add Extensions*.

5.5.12 AssertionError: There is no script for 46 version

This sort of message may be seen if you swap between different branches of CKAN. The .pyc file for database migration 46 exists, but the .py file no longer exists by swapping to an earlier branch. The solution is to delete all pyc files (which is harmless):

```
find . -name "*.pyc" |xargs rm
```

5.5.13 AssertionError: Unexpected files/directories in pyenv/src/ckan

This occurs when installing CKAN source to a virtual environment when using an old version of pip. (e.g. pip 0.3.1 which comes with Ubuntu). Instead you should use pip 1.0.2 or higher, which will be found in your virtual environment: `pyenv/bin/pip`

5.5.14 sqlalchemy.exc.IntegrityError: (IntegrityError) could not create unique index "user_name_key

This occurs when upgrading to CKAN 1.5.1 with a database with duplicate user names. See *Upgrading a package install*

5.5.15 ERROR: must be member of role "okfn" & WARNING: no privileges could be revoked for "public"

These are seen when loading a CKAN database from another machine. It is the result of the database tables being owned by a user that doesn't exist on the new machine. The owner of the table is not important, so this error is harmless and can be ignored.

5.5.16 IOError: [Errno 13] Permission denied: '/var/log/ckan/colorado/color

This is usually seen when you run the paster command with one user, and CKAN is deployed on Apache (for example) which runs as another user. The usual remedy is to run the paster command as user `www-data`. i.e..:

```
sudo -u www-data paster ...
```

5.5.17 ImportError: No module named genshi.template

This is seen when running a paster command. The problem is paster is not recognising the python virtual environment where genshi (and other CKAN libraries) are installed. To resolve this, supply the path to the copy of paster in the virtual environment. e.g.:

```
pyenv/bin/paster ...
```

5.5.18 type "geometry" does not exist

(also `function public.multipolygonfromtext(text) does not exist` `permission denied for language c`)

This may occur when you are using psql or `paster db load`. It means that the database dump was taken from a Postgres database that was spatially enabled (PostGIS installed) and you are loading it into one that is not.

To make your Postgres cluster spatially enabled, see the instructions here: https://github.com/okfn/ckanext-spatial/blob/master/README.rst

CHAPTER 6

For CKAN Developers

6.1 CKAN Domain Model

6.1.1 Overview

- **Datasets**: the central entity in CKAN. Associated to Datasets are 'Resources' (files, APIs etc) and a large variety of metadata.
  - **Core metadata**: Datasets have a set of "core" metadata attributes
  - **Unlimited additional metadata**: Datasets may have an unlimited amount of arbitrary additional metadata in the form of "extra" key/value associations.
  - **Relationships**: relationships between datasets (such as depends on, child of, derived from etc)
- **Resources**: the actual data or APIs associated to a dataset are entered into Resources.

Additionally:

- **Revisions**: All metadata in the CKAN system is *revisioned* – i.e. all changes are recorded. This support reverting changes, viewing changes, and, perhaps most importantly going forward, the exchange of metadata changesets between different CKAN instances or CKAN and other catalogues. However, this is not a core part of the schema.
- **Task Status**: A key/value store used by CKAN Tasks (background processes).

6.1.2 Entity List

- *Dataset*
- *Resource* - a file, API or other resource
- Group
- Dataset Relationship - a relationship between Data Datasets.
- Tag - tags can be applied to packages.
- Vocabulary - tags can belong to vocabularies.

Part of the domain model but not central:

- Revision - changes to the domain model
- *Task Status* - key/value information stored by CKAN Tasks

6.2 Dataset

A Dataset (known as a (Data) Package in CKAN <=1.4) is the object representing datasets in CKAN and, as such, is the central domain object.

When you retrieve a Dataset in the CKAN API it will automatically include information from most related objects including Tags, Resources, Relationships, Ratings etc.

6.2.1 Schema

Mappings to dublin core are in brackets (dc:...).

- id: unique id
- name (slug): unique name that is used in urls and for identification
- title (dc:title): short title for dataset
- url (home page): home page for this dataset
- author (dc:creator): original creator of the dataset
- author_email:
- maintainer: current maintainer or publisher of the dataset
- maintainer_email:
- license (dc:rights): license under which the dataset is made available
- version: dataset version
- notes (description) (dc:description): description and other information about the dataset
- tags: arbitrary textual tags for the dataset
- state: state of dataset in CKAN system (active, deleted, pending)
- resources: list of [[Domain Model/Resource|Resources]]
- groups: list of [[Domain Model/Group|Groups]] this dataset is a member of
- "extras" - arbitrary, unlimited additional key/value fields

The schema in code (see default_package_schema): https://github.com/okfn/ckan/blob/master/ckan/logic/schema.py

6.2.2 Background

The CKAN Dataset was originally heavily based on the kind of packaging information provided for software but with some modifications. One of our aims is to keep things simple and as generic as possible as we have data from a lot of different domains.

Thus we've tried to keep the core metadata pretty restricted but allow for additional info either via tags or via "extra" arbitrary key/value fields.

6.3 Resource

A Resource corresponds to a file, API or other online data resource. A Resource is associated to a *Dataset* (which may have several Resources).

6.3.1 Attributes

Like Datasets, Resources can have arbitrary set of attributes. Thus, the attributes listed here are not exhaustive and may be extended by specific extensions.

Standard

These are standard set of attributes utilized by 'core' CKAN.

- url: the key attribute of a resource (and the only required attribute). The url points to the location online where the content of that resource can be found. For a file this would be the location online of that file (or more generally a url which yields the bitstream representing the contents of that file – for example some "files" are only generated on demand from a database). For an API this would be the endpoint for the api.
- name: a name for this resource (could be used in a ckan url)
- description: A brief description (one sentence) of the Resource. Longer descriptions can go in notes field of the associated Data Package.
- type: the type of the resource. One of: file | file.upload | api | visualization | code | documentation
  - file - a file (GET of this url should yield a bitstream)
  - file.upload - a file uploaded to the *FileStore and File Uploads*
  - api - an API
  - visualization - a visualization
  - code - code related to this dataset (for example a reference to a code repository containing processing scripts)
  - documentation - documentation for this dataset
- format: human created format string with possible nesting e.g. zip:csv. See below for details of the format field.
- mimetype: standard mimetype (e.g. for zipped csv would be application/zip)
- mimetype-inner: mimetype of innermost object (so for example would be text/csv)
- size: size of the resource (content length). Usually only relevant for resources of type file.
- last_modified: the date when this resource's data was last modified (NB: *not* the date when the metadata was modified).
- hash: md5 or sha-1 hash

Resource Quality Information

See http://wiki.ckan.org/Data_Quality. This information, while directly related resources, will not be stored on the resource table. It is currently undecided whether the Resource object will have this data directly available (e.g. via a 'quality' attribute).

Attributes for FileStore Archiving and DataStore Usage

Resource data may have been archived into the FileStore or stored into the DataStore. In these cases the following additional attributes are used:

- cache_url: url for cache of object in *FileStore and File Uploads*
  - Note could be same as resource url if resource directly stored in storage

- cache_last_updated: timestamp when cached version was created
- webstore_url: set to non-empty value if data is in the doc:*datastore* (note unusual naming is a holdover from previous usage)
- webstore_last_updated: timestamp when webstore was last updated

6.3.2 Resource Format Strings

Conventions on format strings:

- file: mime-type or file extensions (for common tile types)
  - Examples: csv (text/csv), xls (application/vnd.ms-excel or application/xls – there are about 6 alternatives!), html (text/html), pdf (application/pdf) etc
- api: {spec-type}+{mime-type-of-standard-response}
  - Examples: sparql+rdf/xml, rest+json
- service: service/{service-identifier}/{type-of-object-or-file-format}
  - Examples: service/gdocs/spreadsheet (google docs spreadsheet)

Nested Formats

It is very common for files to be provided in compressed form (e.g. zip, targz, tar). Strictly the format of this object is the format for the compression (e.g. zip). However, for users it is the underlying format that will matter. To solve this one provides the formats in nested order separate by a colon. Formats should be provide in outermost first (i.e. start with format of last layer and work inwards). Examples:

- zip:json - a zipped json file e.g. myfile.json.zip
- targz:xml - an xml file that has been tar'd and gzipped e.g. myfile.xml.tgz
- torrent:zip:csv - a csv file that has been zipped and then provided as a torrent

Multiple Formats for Same Resources

It is common for a given API to provide data in multiple formats, for example xml and json. In this case use the '||' term. Examples:

- api/xml||json - an API providing both xml and json

Formats for resources that are listings or index pages

It is common, at present, to find projects where the data is in lots of files with these files listed on an index page. Rather than attempt to create a resource entry for each file we have adopted the convention of creating a resource for the relevant index page with a special format string beginning "index", e.g.:

- index/html (an index page in html format)
- index/ftp (an index page for a ftp site)

6.4 Task Status

The Task Status domain object is essentially a key/value store that is used by CKAN Tasks to store the results of each processing task.

6.4.1 Schema

Each task status entry consists of the following required fields:

- **id** [UnicodeText]: A unique ID for each status object. Automatically generated if not provided.
- **entity_id** [UnicodeText]: Each task_status entry is assumed to be information about a task that performs some operation on another CKAN domain object (usually either a dataset/package or a resource). This refers to the ID of that object.
- **entity_type** [UnicodeText]: The type of CKAN domain object that the task operates on (eg: resource).
- **task_type** [UnicodeText]: The type of CKAN Task (eg: qa, webstorer, archiver, etc).
- **key** [UnicodeText]: Key descriptor for data being stored.
- **value** [UnicodeText]: Actual data being stored.

Note: each task status entry must be unique on *(entity_id, task_type, key)*.

They also contain a number of optional fields:

- **state** [UnicodeText]: The current (or final) state of the task.
- **error** [UnicodeText]: Information about any error that occurred during processing.
- **last_updated** [DateTime]: The time at which this entry was last updated. Defaults to the current time.

6.5 Internationalize CKAN

CKAN is used in many countries, and adding a new language to the web interface is a simple process.

CKAN uses the url to determine which language is used. An example would be `/fr/dataset` would be shown in french. If CKAN is running under a directory then an example would be `/root/fr/dataset`. For custom paths the `ckan.root_path` config option can be used and is of the form `/path/from/root/with/{{LANG}}/substitution`.

6.5.1 Supported Languages

CKAN already supports numerous languages. To check whether your language is supported, look in the source at `ckan/i18n` for translation files. Languages are named using two-letter ISO language codes (e.g. `es`, `de`).

If your language is present, you can switch the default language simply by setting the `ckan.locale_default` option in your CKAN config file, as described in *Internationalisation Settings*. For example, to switch to German:

```
ckan.locale_default=de
```

If your language is not supported yet, the remainder of this section section provides instructions on how to prepare a translation file and add it to CKAN.

6.5.2 Adding a New Language

If you want to add an entirely new language to CKAN, you have two options.

- *Transifex Setup*. Creating translation files using Transifex, the open source translation software.
- *Manual Setup*. Creating translation files manually.

Transifex Setup

Transifex, the open translation platform, provides a simple web interface for writing translations and is widely used for CKAN internationalization.

Using Transifex makes it easier to handle collaboration, with an online editor that makes the process more accessible.

Existing CKAN translation projects can be found at: https://www.transifex.net/projects/p/ckan/teams/

When leading up to a CKAN release, the strings are loaded onto Transifex and ckan-discuss list is emailed to encourage translation work. When the release is done, the latest translations on Transifex are checked back into CKAN.

Transifex Administration

The Transifex workflow is as follows:

- Install transifex command-line utilities
- `tx init` in CKAN to connect to Transifex
- Run `python setup.py extract_messages` on the CKAN source
- Upload the local .pot file via command-line `tx push`
- Get people to complete translations on Transifex
- Pull locale .po files via `tx pull`
- `python setup.py compile_catalog`
- Git Commit and push po and mo files

Manual Setup

If you prefer not to use Transifex, you can create translation files manually.

Note: Please keep the CKAN core developers aware of new languages created in this way.

All the English strings in CKAN are extracted into the `ckan.pot` file, which can be found in `ckan/i18n`.

Note: For information, the pot file was created with the `babel` command `python setup.py extract_messages`.

---

0. Install Babel

You need Python's `babel` library (Debian package `python-pybabel`). Install it as follows with pip:

```
pip -E pyenv install babel
```

1. Create a 'po' File for Your Language

First, grab the CKAN i18n repository:

```
hg clone http://bitbucket.org/bboissin/ckan-i18n/
```

Then create a translation file for your language (a po file) using the pot file:

```
python setup.py init_catalog --locale YOUR_LANGUAGE
```

Replace `YOUR_LANGUAGE` with the two-letter ISO language code (e.g. `es`, `de`).

In future, when the pot file is updated, you can update the strings in your po file, while preserving your po edits, by doing:

```
python setup.py update_catalog --locale YOUR-LANGUAGE
```

2. Do the Translation

Edit the po file and translate the strings. For more information on how to do this, see the Pylons book.

We recommend using a translation tool, such as poedit, to check the syntax is correct. There are also extensions for editors such as emacs.

3. Commit the Translation

When the po is complete, commit it to the CKAN i18n repo:

```
git add ckan/i18n/YOUR_LANGUAGE/LC_MESSAGES/ckan.po
git commit -m '[i18n]: New language po added: YOUR_LANGUAGE' ckan/i18n/YOUR_LANGUAGE/LC_MESSAGES/ckan
git push
```

Note: You need to be given credentials to do this - to request these, contact the ckan-discuss list.

4. Compile a Translation

Once you have created a translation (either with Transifex or manually) you can build the po file into a `mo` file, ready for deployment.

With either method of creating the po file, it should be found in the CKAN i18n repository: `ckan/i18n/YOUR_LANGUAGE/LC_MESSAGES/ckan.po`

In this repo, compile the po file like this:

```
python setup.py compile_catalog --locale YOUR_LANGUAGE
```

As before, replace `YOUR_LANGUAGE` with your language short code, e.g. `es`, `de`.

This will result in a binary 'mo' file of your translation at `ckan/i18n/YOUR_LANGUAGE/LC_MESSAGES/ckan.mo`.

5. (optional) Deploy the Translation

This section explains how to deploy your translation automatically to your host, if you are using a remote host.

It assumes a standard layout on the server (you may want to check before you upload!) and that you are deploying to `hu.ckan.net` for language `hu`.

Once you have a compiled translation file, for automated deployment to your host do:

```
fab config_0:hu.ckan.net upload_i18n:hu
```

See the `config_0` options if more configuration is needed e.g. of host or location.

Alternatively, if you do not want to use fab, you can just scp:

```
scp ckan.mo /home/okfn/var/srvc/hu.ckan.net/pyenv/src/ckan/ckan/i18n/hu/LC_MESSAGES/ckan.mo
```

6. Configure the Language

Finally, once the mo file is in place, you can switch between the installed languages using the `ckan.locale` option in the CKAN config file, as described in *Internationalisation Settings*.

6.6 Testing for Developers

If you are installing CKAN from source, or developing extensions, then you need to know how to run CKAN tests.

This section describes testing topics for developers, including basic tests, migration testing and testing against PostgreSQL.

6.6.1 Basic Tests

After completing your source installation of CKAN, you should check that tests pass. You should also check this before checking in changes to CKAN code.

Make sure you've created a config file at `pyenv/ckan/development.ini`. Then activate the Python environment:

```
. pyenv/bin/activate
```

Install nose and other test-specific dependencies into your virtual environment:

```
pip install --ignore-installed -r pyenv/src/ckan/pip-requirements-test.txt
```

At this point you will need to deactivate and then re-activate your virtual environment to ensure that all the scripts point to the correct locations:

```
deactivate
. pyenv/bin/activate
```

Then run the quick development tests:

```
cd pyenv/src/ckan
nosetests ckan/tests --ckan
```

You *must* run the tests from the CKAN directory as shown above, otherwise the `--ckan` plugin won't work correctly.

> **Warning:** By default, the test run is 'quick and dirty' - only good enough as an initial check.

6.6.2 Testing against PostgreSQL

The default way to run tests is defined in `test.ini` (which is the default config file for nose - change it with option `--with-pylons`). This specifies using SQLite and sets `faster_db_test_hacks`, which are compromises.

```
cd pyenv/src/ckan
nosetests ckan/tests --ckan
```

Although SQLite is useful for testing a large proportion of CKAN, actually in deployment, CKAN must run with PostgreSQL.

Running the tests against PostgreSQL is slower but more thorough for two reasons:

1. You test subtleties of PostgreSQL
2. CKAN's default search relies on PostgreSQL's custom full-text search, so these (100 or so) tests are skipped when running against SQLite.

So when making changes to anything involved with search or closely related to the database, it is wise to test against PostgreSQL.

To test against PostgreSQL:

1. Edit your local `development.ini` to specify a PostgreSQL database with the `sqlalchemy.url` parameter.
2. Tell nose to use `test-core.ini` (which imports settings from `development.ini`)

```
nosetests ckan/tests --ckan --with-pylons=test-core.ini
```

The test suite takes a long time to run against standard PostgreSQL (approx. 15 minutes, or close to an hour on Ubuntu/10.04 Lucid).

This can be improved to between 5 and 15 minutes by running PostgreSQL in memory and turning off durability, as described in the PostgreSQL documentation.

6.6.3 Migration Testing

If your changes require a model change, you'll need to write a migration script. To ensure this is tested as well, you should instead run the tests this way:

```
nosetests ckan/tests --ckan --ckan-migration --with-pylons=test-core.ini
```

By default, tests are run using the model defined in `ckan/model`, but by using the `--ckan-migration` option the tests will run using a database that has been created using the migration scripts, which is the way the database is created and upgraded in production. These tests are the most thorough and will take around 20 minutes.

Caution: Ordinarily, you should set `development.ini` to specify a PostgreSQL database so these also get used when running `test-core.ini`, since `test-core.ini` inherits from `development.ini`. If you were to change the `sqlalchemy.url` option in your `development.ini` file to use SQLite, the command above would actually test SQLite rather than PostgreSQL, so always check the setting in `development.ini` to ensure you are running the full tests.

Warning: A common error when wanting to run tests against a particular database is to change `sqlalchemy.url` in `test.ini` or `test-core.ini`. The problem is that these are versioned files and people have checked in these by mistake, creating problems for other developers and the CKAN buildbot. This is easily avoided by only changing `sqlalchemy.url` in your local `development.ini` and testing `--with-pylons=test-core.ini`.

6.6.4 Testing Core Extensions

Some extensions are in the CKAN core codebase and have their own suite of tests. For example:

```
nosetests --ckan ckanext/stats/tests
```

6.6.5 Common error messages

Often errors are due to set-up errors. Always refer to the CKAN buildbot as the canonical build.

Consult *Common error messages* for solutions to a range of setup problems.

6.7 Install Buildbot

This section provides information for CKAN core developers setting up buildbot on an Ubuntu Lucid machine.

If you simply want to check the status of the latest CKAN builds, visit http://buildbot.okfn.org/.

6.7.1 Apt Installs

Install CKAN core dependencies from Lucid distribution:

```
sudo apt-get install build-essential libxml2-dev libxslt-dev
sudo apt-get install wget mercurial postgresql libpq-dev git-core
sudo apt-get install python-dev python-psycopg2 python-virtualenv
sudo apt-get install subversion
```

Maybe need this too:

```
sudo apt-get install python-include
```

Buildbot software:

```
sudo apt-get install buildbot
```

Deb building software:

```
sudo apt-get install -y dh-make devscripts fakeroot cdbs
```

Fabric:

```
sudo apt-get install -y fabric
```

If you get errors with postgres and locales you might need to do these:

```
sudo apt-get install language-pack-en-base
sudo dpkg-reconfigure locales
```

6.7.2 Postgres Setup

If installation before failed to create a cluster, do this after fixing errors:

```
sudo pg_createcluster 8.4 main --start
```

Create users and databases:

```
sudo -u postgres createuser -S -D -R -P buildslave
# set this password (matches buildbot scripts): biomaik15
sudo -u postgres createdb -O buildslave ckan1
sudo -u postgres createdb -O buildslave ckanext
```

6.7.3 Buildslave Setup

Rough commands:

```
sudo useradd -m -s /bin/bash buildslave
sudo chown buildslave:buildslave /home/buildslave
sudo su buildslave
cd ~
git clone https://github.com/okfn/buildbot-scripts.git
ssh-keygen -t rsa
cp /home/buildslave/.ssh/id_rsa.pub  ~/.ssh/authorized_keys
mkdir -p ckan/build
cd ckan/build
python ~/ckan-default.py
buildbot create-slave ~ localhost:9989 okfn <buildbot_password>
vim ~/info/admin
vim ~/info/host
mkdir /home/buildslave/pip_cache
virtualenv pyenv-tools
pip -E pyenv-tools install buildkit
```

6.7.4 Buildmaster Setup

Rough commands:

```
mkdir ~/buildmaster
buildbot create-master ~/buildmaster
ln -s /home/buildslave/master/master.cfg ~/buildmaster/master.cfg
cd ~/buildmaster
buildbot checkconfig
```

6.7.5 Startup

Setup the daemons for master and slave:

```
sudo vim /etc/default/buildbot
```

This file should be edited to be like this:

```
BB_NUMBER[0]=0                  # index for the other values; negative disables the bot
BB_NAME[0]="okfn"               # short name printed on startup / stop
BB_USER[0]="okfn"               # user to run as
BB_BASEDIR[0]="/home/okfn/buildmaster"          # basedir argument to buildbot (absolute path)
BB_OPTIONS[0]=""                # buildbot options
BB_PREFIXCMD[0]=""              # prefix command, i.e. nice, linux32, dchroot

BB_NUMBER[1]=1                  # index for the other values; negative disables the bot
BB_NAME[1]="okfn"               # short name printed on startup / stop
BB_USER[1]="buildslave"               # user to run as
```

```
BB_BASEDIR[1]="/home/buildslave"          # basedir argument to buildbot (absolute path)
BB_OPTIONS[1]=""                # buildbot options
BB_PREFIXCMD[1]=""              # prefix command, i.e. nice, linux32, dchroot
```

Start master and slave (according to /etc/default/buildbot):

```
sudo /etc/init.d/buildbot start
```

Now check you can view buildbot at http://localhost:8010/

6.7.6 Connect Ports

It's preferable to view the buildbot site at port 80 rather than 8010.

If there is no other web service on this machine, you might connect up the addresses using `iptables`:

```
sudo iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT --to-port 8010
```

Otherwise it is best to set up a reverse proxy. Using Apache, edit this file:

```
sudo vim /etc/apache2/sites-available/buildbot.okfn.org
```

to look like this:

```
<VirtualHost *:80>
   ServerName buildbot.okfn.org

   ProxyPassReverse ts Off
     <Proxy *>
             Order deny,allow
             Allow from all
     </Proxy>
     ProxyPass         / http://127.0.0.1:8010/
     ProxyPassReverse  / http://127.0.0.1:8010/
     ProxyPreserveHost On
</VirtualHost>
```

or the old one had:

```
<VirtualHost *:80>
    ServerAdmin sysadmin@okfn.org
    ServerName buildbot.okfn.org
    DocumentRoot /var/www/
    <Location />
        Order allow,deny
        allow from all
    </Location>
    RewriteEngine On
    RewriteRule /(.*) http://localhost:8010/$1 [P,L]
</VirtualHost>
```

Then:

```
sudo apt-get install libapache2-mod-proxy-html
sudo a2enmod proxy_http
sudo a2ensite buildbot.okfn.org
sudo /etc/init.d/apache2 reload
```

CHAPTER 7

Other material

7.1 Contrib and Tools

This is a a place for code, scripts and applications that extend CKAN in some way (but which aren't extensions).

Note: Have something to add to this page? Please either email `info [at] ckan [dot] org` or just edit this page in the CKAN repo.

7.1.1 Google Docs Integration

Integration of Google docs spreadsheets with CKAN. Google app script script which supports:

- Publish a Google docs spreadsheet as a dataset on CKAN
- Push and pull metadata from a CKAN instance to a Google docs spreadsheet

Install: via the google docs Script Gallery in the tools menu (or use the source).

Repo: https://github.com/okfn/ckan-google-docs

7.1.2 Google Refine Extension for CKAN

This extension allows data of Google Refine projects to be uploaded to CKAN Storage and connected to a dataset on a running CKAN instance (for example http://thedatahub.org).

- http://lab.linkeddata.deri.ie/2011/grefine-ckan/
- Intro: http://ckan.org/2011/07/05/google-refine-extension-for-ckan/
- Author: Fadi Maali (DERI)

Repo: https://github.com/fadmaa/grefine-ckan-storage-extension

7.1.3 Embeddable Dataset Count Widget

Simple group count widget in pure javascript for embedding in other websites.

http://okfnlabs.org/ckanjs/widgets/count/

7.1.4 Embeddable Dataset Listing Widget

http://okfnlabs.org/ckanjs/widgets/search/

7.1.5 SPARQL Endpoint Status Checker

Status of SPARQL endpoints (in the DataHub).

http://labs.mondeca.com/sparqlEndpointsStatus/index.html

- Author: Mondeca

7.1.6 PublicData.eu Data map

http://publicdata.eu/map

7.1.7 VoID metadata generator

Useful to automatically correct/maintain/enrich dataset metadata

- voidGen : http://www.hpi.uni-potsdam.de/naumann/projects/btc
- ve2 : http://lab.linkeddata.deri.ie/ve2/
- (POC by Richard) : https://github.com/cygri/make-void
- (POC by Pierre-Yves) : http://labs.mondeca.com/vocabuse/

7.2 CKAN CHANGELOG

7.2.1 v1.7.4 2013-08-13

Bug fixes: * Refactor for user update logic * Tweak resources visibility query

7.2.2 v1.7.3 2013-05-10

Bug fixes: * Fixed possible XSS vulnerability on html input (#703)

7.2.3 v1.7.2 2012-10-19

Minor: * Documentation enhancements regarding file uploads

Bug fixes: * Fixes for lincences i18n * Remove sensitive data from user dict (#2784) * Fix bug in feeds controller (#2869) * Show dataset author and maintainer names even if they have no emails * Fix URLs for some Amazon buckets * Other minor fixes

7.2.4 v1.7.1 2012-06-20

Minor: * Documentation enhancements regarding install and extensions (#2505) * Home page and search results speed improvements (#2402,#2403) * I18n: Added Greek translation and updated other ones (#2506)

Bug fixes: * UI fixes (#2507) * Fixes for i18n login and logout issues (#2497) * Date on add/edit resource breaks if offset is specified (#2383) * Fix in organizations read page (#2509) * Add synchronous_search plugin to deployment.ini template (#2521) * Inconsistent language on license dropdown (#2575) * Fix bug in translating lists in multilingual plugin * Group autocomplete doesn't work with multiple words (#2373) * Other minor fixes

7.2.5 v1.7 2012-05-09

Major:

- Updated SOLR schema (#2327). Note: This will require and update of the SOLR schema file and a reindex.
- Support for Organization based workflow, with membership determinig access permissions to datasets (#1669,#2255)
- Related items such as visualizations, applications or ideas can now be added to datasets (#2204)
- Restricted vocabularies for tags, allowing grouping related tags together (#1698)
- Internal analytics that track number of views and downloads for datasets and resources (#2251)
- Consolidated multilingual features in an included extension (#1821,#1820)
- Atom feeds for publishers, tags and search results (#1593,#2277)
- RDF dump paster command (#2303)
- Better integration with the DataStore, based on ElasticSearch, with nice helper docs (#1797)
- Updated the Recline data viewer with new features such as better graphs and a map view (#2236,#2283)
- Improved and redesigned documentation (#2226,#2245,#2248)

Minor:

- Groups can have an image associated (#2275)
- Basic resource validation (#1711)
- Ability to search without accents for accented words (#906)
- Weight queries so that title is more important than rest of body (#1826)
- Enhancements in the dataset and resource forms (#1506)
- OpenID can now be disabled (#1830)
- API and forms use same validation (#1792)
- More robust bulk search indexing, with options to ignore exceptions and just refresh (#1616i,#2232)
- Modify where the language code is placed in URLs (#2261)
- Simplified licenses list (#1359)
- Add extension point for dataset view (#1741)

Bug fixes:

- Catch exceptions on the QA archiver (#1809)

- Error when changing language when CKAN is mounted in URL (#1804)
- Naming of a new package/group can clash with a route (#1742)
- Can't delete all of a package's resources over REST API (#2266)
- Group edit form didn't allow adding multiple datasets at once (#2292)
- Fix layout bugs in IE 7 (#1788)
- Bug with Portugese translation and Javascript (#2318)
- Fix broken parse_rfc_2822 helper function (#2314)

7.2.6 v1.6 2012-02-24

Major:

- Resources now have their own pages, as well as showing in the Dataset (#1445, #1449)
- Group pages enhanced, including in-group search (#1521)
- User pages enhanced with lists of datasets (#1396) and recent activity (#1515)
- Dataset view page decluttered (#1450)
- Tags not restricted to just letters and dashes (#1453)
- Stats Extension and Storage Extension moved into core CKAN (#1576, #1608)
- Ability to mounting CKAN at a sub-URL (#1401, #1659)
- 5 Stars of Openness ratings show by resources, if ckanext-qa is installed (#1583)
- Recline Data Explorer (for previewing and plotting data) improved and v2 moved into core CKAN (#1602, #1630)

Minor:

- 'About' page rewritten and easily customisable in the config (#1626)
- Gravatar picture displayed next to My Account link (#1528)
- 'Delete' button for datasets (#1425)
- Relationships API more RESTful, validated and documented (#1695)
- User name displayed when logged in (#1529)
- Database dumps now exclude deleted packages (#1623)
- Dataset/Tag name length now limited to 100 characters in API (#1473)
- 'Status' API call now includes installed extensions (#1488)
- Command-line interface for list/read/deleting datasets (#1499)
- Slug API calls tidied up and documented (#1500)
- Users nagged to add email address if missing from their account (#1413)
- Model and API for Users to become Members of a Group in a certain Capacity (#1531, #1477)
- Extension interface to adjust search queries, indexing and results (#1547, #1738)
- API for changing permissions (#1688)

Bug fixes:

- Group deletion didn't work (#1536)
- metadata_created used to return an entirely wrong date (#1546)
- Unicode characters in field-specific API search queries caused exception (since CKAN 1.5) (#1798)
- Sometimes task_status errors weren't being recorded (#1483)
- Registering or Logging in failed silently when already logged in (#1799)
- Deleted packages were browseable by administrators and appeared in dumps (#1283, #1623)
- Facicon was a broken link unless corrected in config file (#1627)
- Dataset search showed last result of each page out of order (#1683)
- 'Simple search' mode showed 0 packages on home page (#1709)
- Occasionally, 'My Account' shows when user is not logged in (#1513)
- Could not change language when on a tag page that had accented characters or dataset creation page (#1783, #1791)
- Editing package via API deleted its relationships (#1786)

7.2.7 v1.5.1 2012-01-04

Major:

- Background tasks (#1363, #1371, #1408)
- Fix for security issue affecting CKAN v1.5 (#1585)

Minor:

- Language support now excellent for European languages: en de fr it es no sv pl ru pt cs sr ca
- **Web UI improvements:**
  - Resource editing refreshed
  - Group editing refreshed
  - Indication that group creation requires logging-in (#1004)
  - Users' pictures displayed using Gravatar (#1409)
  - 'Welcome' banner shown to new users (#1378)
  - Group package list now ordered alphabetically (#1502)
- Allow managing a dataset's groups also via package entity API (#1381)
- Dataset listings in API standardised (#1490)
- Search ordering by modification and creation date (#191)
- Account creation disallowed with Open ID (create account in CKAN first) (#1386)
- User name can be modified (#1386)
- Email address required for registration (for password reset) (#1319)
- Atom feeds hidden for now
- New config options to ease CSS insertion into the template (#1380)
- Removed ETag browser cache headers (#1422)

- CKAN version number and admin contact in new 'status_show' API (#1087)
- Upgrade SQLAlchemy to 0.7.3 (compatible with Postgres up to 9.1) (#1433)
- SOLR schema is now versioned (#1498)

Bug fixes:

- Group ordering on main page was alphabetical but should be by size (since 1.5) (#1487)
- Package could get added multiple times to same Group, distorting Group size (#1484)
- Search index corruption when multiple CKAN instances on a server all storing the same object (#1430)
- Dataset property metadata_created had wrong value (since v1.3) (#1546)
- Tag browsing showed tags for deleted datasets (#920)
- User name change field validation error (#1470)
- You couldn't edit a user with a unicode email address (#1479)
- Package search API results missed the extra fields (#1455)
- OpenID registration disablement explained better (#1532)
- Data upload (with ckanext-storage) failed if spaces in the filename (#1518)
- Resource download count fixed (integration with ckanext-googleanalytics) (#1451)
- Multiple CKANs with same dataset IDs on the same SOLR core would conflict (#1462)

7.2.8 v1.5 2011-11-07

Deprecated due to security issue #1585

Major:

- **New visual theme (#1108)**
  - Package & Resource edit overhaul (#1294/#1348/#1351/#1368/#1296)
  - JS and CSS reorganisation (#1282, #1349, #1380)
- Apache Solr used for search in core instead of Postgres (#1275, #1361, #1365)
- Authorization system now embedded in the logic layer (#1253)
- Captcha added for user registration (#1307, #1431)
- UI language translations refreshed (#1292, #1350, #1418)
- Action API improved with docs now (#1315, #1302, #1371)

Minor:

- Cross-Origin Resource Sharing (CORS) support (#1271)
- Strings to translate into other languages tidied up (#1249)
- Resource format autocomplete (#816)
- Database disconnection gives better error message (#1290)
- Log-in cookie is preserved between sessions (#78)
- Extensions can access formalchemy forms (#1301)
- 'Dataset' is the new name for 'Package' (#1293)

- Resource standard fields added: type, format, size (#1324)
- Listing users speeded up (#1268)
- Basic data preview functionality moved to core from QA extension (#1357)
- Admin Extension merged into core CKAN (#1264)
- URLs in the Notes field are automatically linked (#1320)
- Disallow OpenID for account creation (but can be linked to accounts) (#1386)
- Tag name now validated for max length (#1418)

Bug fixes:

- Purging of revisions didn't work (since 1.4.3) (#1258)
- Search indexing wasn't working for SOLR (since 1.4.3) (#1256)
- Configuration errors were being ignored (since always) (#1172)
- Flash messages were temporarily held-back when using proxy cache (since 1.3.2) (#1321)
- On login, user told 'welcome back' even if he's just registered (#1194)
- Various minor exceptions cropped up (mostly since 1.4.3) (#1334, #1346)
- Extra field couldn't be set to original value when key deleted (#1356)
- JSONP callback parameter didn't work for the Action API (since 1.4.3) (#1437)
- The same tag could be added to a package multiple times (#1331)

7.2.9 v1.4.3.1 2011-09-30

Minor:

- Added files to allow debian packaging of CKAN
- Added Catalan translation

Bug fixes:

- Incorrect Group creation form parameter caused exception (#1347)
- Incorrect AuthGroup creation form parameter caused exception (#1346)

7.2.10 v1.4.3 2011-09-13

Major:

- Action API (API v3) (beta version) provides powerful RPC-style API to CKAN data (#1335)
- Documentation overhaul (#1142, #1192)

Minor:

- Viewing of a package at a given date (as well as revision) with improved UI (#1236)
- Extensions can now add functions to the logic layer (#1211)
- Refactor all remaining database code out of the controllers and into the logic layer (#1229)
- Any OpenID log-in errors that occur are now displayed (#1228)
- 'url' field added to search index (e9214)

- Speed up tag reading (98d72)
- Cope with new WebOb version 1 (#1267)
- Avoid exceptions caused by bots hitting error page directly (#1176)
- Too minor to mention: #1234,

Bug fixes:

- Re-adding tags to a package failed (since 1.4.1 in Web UI, 1.4 in API) (#1239)
- Modified revisions retrieved over API caused exception (since 1.4.2) (#1310)
- Whichever language you changed to, it announced "Language set to: English" (since 1.3.1) (#1082)
- Incompatibilities with Python 2.5 (since 1.3.4.1 and maybe earlier) (#1325)
- You could create an authorization group without a name, causing exceptions displaying it (#1323)
- Revision list wasn't showing deleted packages (b21f4)
- User editing error conditions handled badly (#1265)

7.2.11 v1.4.2 2011-08-05

Major:

- Packages revisions can be marked as 'moderated' (#1141, #1147)
- Password reset facility (#1186/#1198)

Minor:

- Viewing of a package at any revision (#1236)
- API POSTs can be of Content-Type "application/json" as alternative to existing "application/x-www-form-urlencoded" (#1206)
- Caching of static files (#1223)

Bug fixes:

- When you removed last row of resource table, you could't add it again - since 1.0 (#1215)
- Adding a tag to package that had it previously didn't work - since 1.4.1 in UI and 1.4.0 in API (#1239)
- Search index was not updated if you added a package to a group - since 1.1 (#1140)
- Exception if you had any Groups and migrated between CKAN v1.0.2 to v1.2 (migration 29) - since v1.0.2 (#1205)
- API Package edit requests returned the Package in a different format to usual - since 1.4 (#1214)
- API error responses were not all JSON format and didn't have correct Content-Type (#1214)
- API package delete doesn't require a Content-Length header (#1214)

7.2.12 v1.4.1 2011-06-27

Major:

- Refactor Web interface to use logic layer rather than model objects directly. Forms now defined in navl schema and designed in HTML template. Forms use of Formalchemy is deprecated. (#1078)

Minor:

- Links in user-supplied text made less attractive to spammers (nofollow) #1181
- Package change notifications - remove duplicates (#1149)
- Metadata dump linked to (#1169)
- Refactor authorization code to be common across Package, Group and Authorization Group (#1074)

Bug fixes

- Duplicate authorization roles were difficult to delete (#1083)

7.2.13 v1.4 2011-05-19

Major:

- Authorization forms now in grid format (#1074)
- Links to RDF, N3 and Turtle metadata formats provided by semantic.ckan.net (#1088)
- Refactor internal logic to all use packages in one format - a dictionary (#1046)
- A new button for administrators to change revisions to/from a deleted state (#1076)

Minor:

- Etags caching can now be disabled in config (#840)
- Command-line tool to check search index covers all packages (#1073)
- Command-line tool to load/dump postgres database (#1067)

Bug fixes:

- Visitor can't create packages on new CKAN install - since v1.3.3 (#1090)
- OpenID user pages couldn't be accessed - since v1.3.2 (#1056)
- Default site_url configured to ckan.net, so pages obtains CSS from ckan.net- since v1.3 (#1085)

7.2.14 v1.3.3 2011-04-08

Major:

- Authorization checks added to editing Groups and PackageRelationships (#1052)
- API: Added package revision history (#1012, #1071)

Minor:

- API can take auth credentials from cookie (#1001)
- Theming: Ability to set custom favicon (#1051)
- Importer code moved out into ckanext-importlib repo (#1042)
- API: Group can be referred to by ID (in addition to name) (#1045)
- Command line tool: rights listing can now be filtered (#1072)

Bug fixes:

- SITE_READ role setting couldn't be overridden by sysadmins (#1044)
- Default 'reader' role too permissive (#1066)
- Resource ordering went wrong when editing and adding at same time (#1054)

- GET followed by PUTting a package stored an incorrect license value (#662)
- Sibling package relationships were shown for deleted packages (#664)
- Tags were displayed when they only apply to deleted packages (#920)
- API: 'Last modified' time was localised - now UTC (#1068)

7.2.15 v1.3.2 2011-03-15

Major:

- User list in the Web interface (#1010)
- CKAN packaged as .deb for install on Ubuntu
- Resources can have extra fields (although not in web interface yet) (#826)
- CSW Harvesting - numerous of fixes & improvements. Ready for deployment. (#738 etc)
- Language switcher (82002)

Minor:

- Wordpress integration refactored as a Middleware plugin (#1013)
- Unauthorized actions lead to a flash message (#366)
- Resources Groups to group Resources in Packages (#956)
- Plugin interface for authorization (#1011)
- Database migrations tested better and corrected (#805, #998)
- Government form moved out into ckanext-dgu repo (#1018)
- Command-line user authorization tools extended (#1038, #1026)
- Default user roles read from config file (#1039)

Bug fixes:

- Mounting of filesystem (affected versions since 1.0.1) (#1040)
- Resubmitting a package via the API (affected versions since 0.6?) (#662)
- Open redirect (affected v1.3) (#1026)

7.2.16 v1.3 2011-02-18

http://ckan.org/milestone/ckan-v1.3

Highlights of changes:

- **Package edit form improved:**
  - field instructions (#679)
  - name autofilled from title (#778)
- Group-based access control - Authorization Groups (#647)
- Metadata harvest job management (#739, #884, #771)
- CSW harvesting now uses owslib (#885)
- Package creation authorization is configurable (#648)

- Read-only maintenance mode (#777)
- Stats page (#832) and importer (#950) moved out into CKAN extensions

Minor:

- site_title and site_description config variables (#974)
- Package creation/edit timestamps (#806)
- Caching configuration centralised (#828)
- Command-line tools - sysadmin management (#782)
- Group now versioned (#231)

7.2.17 v1.2 2010-11-25

http://ckan.org/milestone/ckan-v1.2

Highlights of changes:

- Package edit form: attach package to groups (#652) & revealable help
- Form API - Package/Harvester Create/New (#545)
- Authorization extended: user groups (#647) and creation of packages (#648)
- Plug-in interface classes (#741)
- WordPress twentyten compatible theming (#797)
- Caching support (ETag) (#693)
- Harvesting GEMINI2 metadata records from OGC CSW servers (#566)

Minor:

- New API key header (#466)
- Group metadata now revisioned (#231)

7.2.18 v1.1 2010-08-10

http://ckan.org/milestone/v1.1

Highlights of changes:

- Changes to the database cause notifications via AMQP for clients (#325)
- Pluggable search engines (#317), including SOLR (#353)
- API is versioned and packages & groups can be referred to by invariant ID (#313)
- Resource search in API (#336)
- Visual theming of CKAN now easy (#340, #320)
- Greater integration with external Web UIs (#335, #347, #348)
- Plug-ins can be configured to handle web requests from specified URIs and insert HTML into pages.

Minor:

- Search engine optimisations e.g. alphabetical browsing (#350)
- CSV and JSON dumps improved (#315)

7.2.19 v1.0.2 2010-08-27

- Bugfix: API returns error when creating package (#432)

7.2.20 v1.0.1 2010-06-23

Functionality:

- API: Revision search 'since id' and revision model in API
- API: Basic API versioning - packages specified by ID (#313)
- Pluggable search - initial hooks
- Customisable templates (#340) and external UI hooks (#335)

Bugfixes:

- Revision primary key lost in migrating data (#311)
- Local authority license correction in migration (#319)
- I18n formatting issues
- Web i/f searches foreign characters (#319)
- Data importer timezone issue (#330)

7.2.21 v1.0 2010-05-11

CKAN comes of age, having been used successfully in several deployments around the world. 56 tickets covered in this release. See: http://ckan.org/milestone/v1.0

Highlights of changes:

- Package edit form: new pluggable architecture for custom forms (#281, #286)
- Package revisions: diffs now include tag, license and resource changes (#303)
- Web interface: visual overhaul (#182, #206, #214-#227, #260) including a tag cloud (#89)
- i18n: completion in Web UI - now covers package edit form (#248)
- API extended: revisions (#251, #265), feeds per package (#266)
- Developer documentation expanded (#289, #290)
- Performance improved and CKAN stress-tested (#201)
- Package relationships (Read-Write in API, Read-Only in Web UI) (#253-257)
- Statistics page (#184)
- Group edit: add multiple packages at once (#295)
- Package view: RDF and JSON formatted metadata linked to from package page (#247)

Bugfixes:

- Resources were losing their history (#292)
- Extra fields now work with spaces in the name (#278, #280) and international characters (#288)
- Updating resources in the REST API (#293)

Infrastructural:

- Licenses: now uses external License Service ('licenses' Python module)
- Changesets introduced to support distributed revisioning of CKAN data - see doc/distributed.rst for more information.

7.2.22 v0.11 2010-01-25

Our biggest release so far (55 tickets) with lots of new features and improvements. This release also saw a major new production deployment with the CKAN software powering http://data.gov.uk/ which had its public launch on Jan 21st!

For a full listing of tickets see: <http://ckan.org/milestone/v0.11>. Main highlights:

- Package Resource object (multiple download urls per package): each package can have multiple 'resources' (urls) with each resource having additional metadata such as format, description and hash (#88, #89, #229)
- "Full-text" searching of packages (#187)
- Semantic web integration: RDFization of all data plus integration with an online RDF store (e.g. for http://www.ckan.net/ at http://semantic.ckan.net/ or Talis store) (#90 #163)
- Package ratings (#77 #194)
- i18n: we now have translations into German and French with deployments at http://de.ckan.net/ and http://fr.ckan.net/ (#202)
- Package diffs available in package history (#173)
- Minor:
  - Package undelete (#21, #126)
  - Automated CKAN deployment via Fabric (#213)
  - Listings are sorted alphabetically (#195)
  - Add extras to rest api and to ckanclient (#158 #166)
- Infrastructural:
  - Change to UUIDs for revisions and all domain objects
  - Improved search performance and better pagination
  - Significantly improved performance in API and WUI via judicious caching

7.2.23 v0.10 2009-09-30

- Switch to repoze.who for authentication (#64)
- Explicit User object and improved user account UI with recent edits etc (#111, #66, #67)
- Generic Attributes for Packages (#43)
- Use sqlalchemy-migrate to handle db/model upgrades (#94)
- "Groups" of packages (#105, #110, #130, #121, #123, #131)
- Package search in the REST API (#108)
- Full role-based access control for Packages and Groups (#93, #116, #114, #115, #117, #122, #120)
- New CKAN logo (#72)
- Infrastructural:

  - Upgrade to Pylons 0.9.7 (#71)
  - Convert to use formalchemy for all forms (#76)
  - Use paginate in webhelpers (#118)
- Minor:
  - Add author and maintainer attributes to package (#91)
  - Change package state in the WUI (delete and undelete) (#126)
  - Ensure non-active packages don't show up (#119)
  - Change tags to contain any character (other than space) (#62)
  - Add Is It Open links to package pages (#74)

7.2.24 v0.9 2009-07-31

- (DM!) Add version attribute for package
- Fix purge to use new version of vdm (0.4)
- Link to changed packages when listing revision
- Show most recently registered or updated packages on front page
- Bookmarklet to enable easy package registration on CKAN
- Usability improvements (package search and creation on front page)
- Use external list of licenses from license repository
- Convert from py.test to nosetests

7.2.25 v0.8 2009-04-10

- View information about package history (ticket:53)
- Basic datapkg integration (ticket:57)
- Show information about package openness using icons (ticket:56)
- One-stage package create/registration (r437)
- Reinstate package attribute validation (r437)
- Upgrade to vdm 0.4

7.2.26 v0.7 2008-10-31

- Convert to use SQLAlchemy and vdm v0.3 (v. major)
- Atom/RSS feed for Recent Changes
- Package search via name and title
- Tag lists show number of associated packages

7.2.27 v0.6 2008-07-08

- Autocompletion (+ suggestion) of tags when adding tags to a package.
- Paginated lists for packages, tags, and revisions.
- RESTful machine API for package access, update, listing and creation.
- API Keys for users who wish to modify information via the REST API.
- Update to vdm v0.2 (SQLObject) which fixes ordering of lists.
- Better immunity to SQL injection attacks.

7.2.28 v0.5 2008-01-22

- Purging of a Revision and associated changes from cli and wui (ticket:37)
- Make data available in machine-usable form via sql dump (ticket:38)
- Upgrade to Pylons 0.9.6.* and deploy (ticket:41)
- List and search tags (ticket:33)
- (bugfix) Manage reserved html characters in urls (ticket:40)
- New spam management utilities including (partial) blacklist support

7.2.29 v0.4 2007-07-04

- Preview support when editing a package (ticket:36).
- Correctly list IP address of of not logged in users (ticket:35).
- Improve read action for revision to list details of changed items (r179).
- Sort out deployment using modpython.

7.2.30 v0.3 2007-04-12

- System now in a suitable state for production deployment as a beta
- Domain model versioning via the vdm package (currently released separately)
- Basic Recent Changes listing log messages
- User authentication (login/logout) via open ID
- License page
- Myriad of small fixes and improvements

7.2.31 v0.2 2007-02

- Complete rewrite of ckan to use pylons web framework
- Support for full CRUD on packages and tags
- No support for users (authentication)
- No versioning of domain model objects

7.2.32 v0.1 2006-05

NB: not an official release

- Almost functional system with support for persons, packages
- Tag support only half-functional (tags are per package not global)
- Limited release and file support

CHAPTER 8

Indices and tables

- *genindex*
- *modindex*
- *search*