Comprehensive Knowledge Archive Network (CKAN) Developer Documentation

Release 2.1.5

Open Knowledge Foundation

July 22, 2015

Contents

This documentation covers how to set up and manage CKAN. For high-level information on what CKAN is, see the CKAN website.

Structure & Audiences

These docs are ordered with the beginner documentation first, and the most advanced documentation last:

- Installing CKAN and Getting Started walk you through installing CKAN and setting up your own CKAN site with some basic customizations. These docs are for **sysadmins** who're new to CKAN and want go get started with it.
- The sections under Features cover setting up and using CKAN features, beyond those that just work out of the box. These are for **sysadmins** who want to learn how to manage and get more out of their CKAN site.
- Writing Extensions, Theming and The CKAN API are advanced docs for **developers** who want to develop an extension, theme or API app using CKAN.
- Contributing to CKAN and Testing CKAN are for **testers**, **translators** and **core developers** who want to contribute to CKAN.
- Finally, Config File Options and Changelog are **reference** docs covering CKAN's config file options and the differences between CKAN releases.

CHAPTER 1

Installing CKAN

Before you can use CKAN on your own computer, you need to install it. There are two ways to install CKAN, from package, or from source.

From package is the quickest and easiest way to install CKAN, but it requires Ubuntu 12.04 64-bit. **You should install CKAN from package if**:

- You want to install CKAN on an Ubuntu 12.04, 64-bit server, *and*
- You only want to run one CKAN website per server, *and*
- You want to run CKAN, Solr and PostgreSQL on the same server

See Installing CKAN from Package.

You should install CKAN from source if:

- You want to install CKAN on a 32-bit computer, *or*
- You want to install CKAN on a different version of Ubuntu, not 12.04, *or*
- You want to install CKAN on another operating system (eg. RedHat, CentOS, OS X), *or*
- You want to run multiple CKAN websites on the same server, *or*
- You want to run CKAN, Solr and PostgreSQL on different servers, *or*
- You want to install CKAN for development

See Installing CKAN from Source.

If you've already setup a CKAN website and want to upgrade it to a newer version of CKAN, see Upgrading CKAN.

1.1 Installing CKAN from Package

This section describes how to install CKAN from package. This is the quickest and easiest way to install CKAN, but it requires **Ubuntu 12.04 64-bit**. If you're not using Ubuntu 12.04 64-bit, or if you're installing CKAN for development, you should follow Installing CKAN from Source instead.

1.1.1 1. Install the CKAN Package

On your Ubuntu 12.04 system, open a terminal and run these commands to install CKAN:

1. Update Ubuntu's package index:

```
sudo apt-get update
```

2. Install the Ubuntu packages that CKAN requires:

```
sudo apt-get install -y nginx apache2 libapache2-mod-wsgi libpq5
```

3. Download the CKAN package:

```
wget http://packaging.ckan.org/python-ckan_2.0_amd64.deb
```

Note: If `wget` is not present, you can install it via:

```
sudo apt-get install wget
```

---

4. Install the CKAN package:

```
sudo dpkg -i python-ckan_2.0_amd64.deb
```

Note: If you get the following error it means that for some reason the Apache WSGI module was not enabled:

```
Syntax error on line 1 of /etc/apache2/sites-enabled/ckan_default:
Invalid command 'WSGISocketPrefix', perhaps misspelled or defined by a module not included in the ser
Action 'configtest' failed.
The Apache error log may have more information.
   ...fail!
```

You can enable it by running these commands in a terminal:

```
sudo a2enmod wsgi
sudo service apache2 restart
```

1.1.2 2. Install PostgreSQL and Solr

Tip: You can install PostgreSQL, Solr and CKAN on different servers. Just change the *sqlalchemy.url* and *solr_url* settings in your /etc/ckan/default/production.ini file to reference your PostgreSQL and Solr servers.

---

1. Install PostgreSQL and Solr, run this command in a terminal:

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

The install will whirr away, then towards the end you'll see this:

```
* Not starting jetty - edit /etc/default/jetty and change NO_START to be 0 (or comment it out).
```

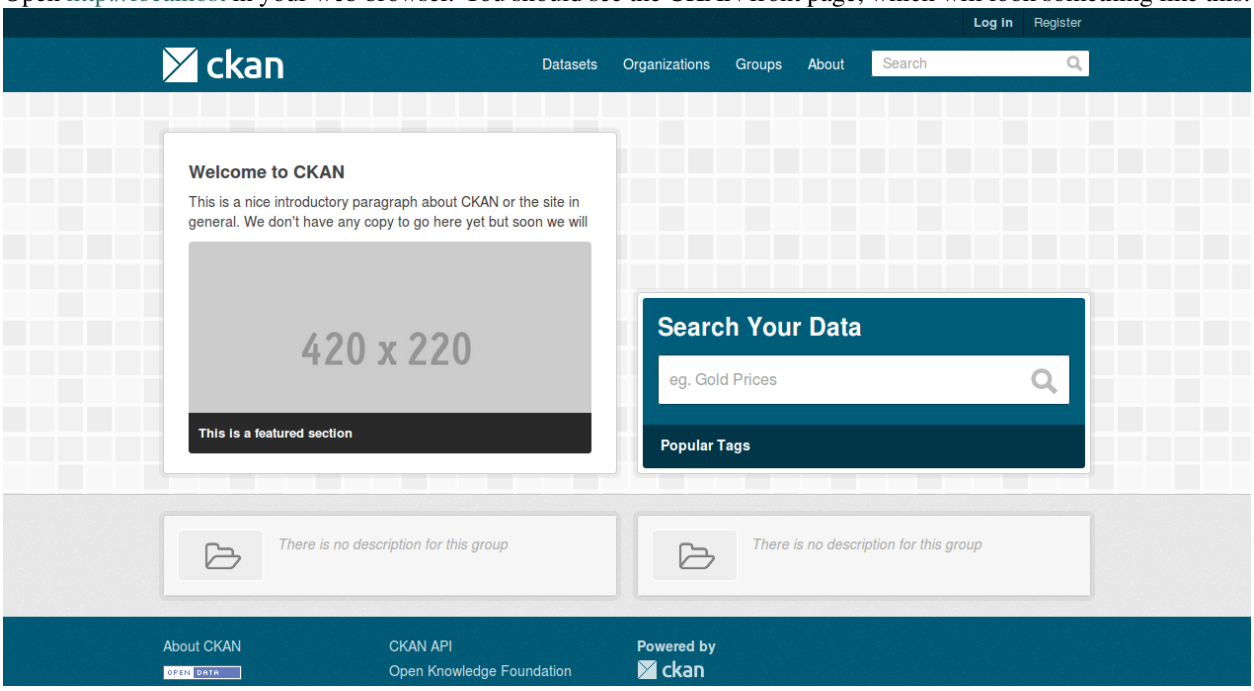
2. Follow the instructions in *Single Solr instance* or *Multiple Solr cores* to setup Solr.
3. Follow the instructions in *3. Setup a PostgreSQL database* to setup PostgreSQL, then edit the *sqlalchemy.url* option in your /etc/ckan/default/production.ini file and set the correct password, database and database user.
4. Initialize your CKAN database by running this command in a terminal:

```
sudo ckan db init
```

5. Optionally, setup the DataStore by following the instructions in DataStore Extension.
6. Also optionally, you can enable file uploads by following the instructions in FileStore and File Uploads.

1.1.3 3. You're done!

Open http://localhost in your web browser. You should see the CKAN front page, which will look something like this:



You can now move on to Getting Started to begin using and customizing your CKAN site.

1.2 Installing CKAN from Source

This section describes how to install CKAN from source. Although Installing CKAN from Package is simpler, it requires Ubuntu 12.04 64-bit. Installing CKAN from source works with other versions of Ubuntu and with other operating systems (e.g. RedHat, Fedora, CentOS, OS X). If you install CKAN from source on your own operating system, please share your experiences on our How to Install CKAN wiki page.

From source is also the right installation method for developers who want to work on CKAN.

1.2.1 1. Install the required packages

If you're using a Debian-based operating system (such as Ubuntu) install the required packages with this command:

```
sudo apt-get install python-dev postgresql libpq-dev python-pip python-virtualenv git-core solr-jetty
```

If you're not using a Debian-based operating system, find the best way to install the following packages on your operating system (see our How to Install CKAN wiki page for help):

Package	Description
Python	The Python programming language, v2.6 or 2.7
PostgreSQL	The PostgreSQL database system, v8.4 or newer
libpq	The C programmer's interface to PostgreSQL
pip	A tool for installing and managing Python packages
virtualenv	The virtual Python environment builder
Git	A distributed version control system
Apache Solr	A search platform
Jetty	An HTTP server (used for Solr)
OpenJDK 6 JDK	The Java Development Kit

1.2.2 2. Install CKAN into a Python virtual environment

Tip: If you're installing CKAN for development and want it to be installed in your home directory, you can symlink the directories used in this documentation to your home directory. This way, you can copy-paste the example commands from this documentation without having to modify them, and still have CKAN installed in your home directory:

```
mkdir -p ~/ckan/lib
sudo ln -s ~/ckan/lib /usr/lib/ckan
mkdir -p ~/ckan/etc
sudo ln -s ~/ckan/etc /etc/ckan
```

1. Create a Python virtual environment (virtualenv) to install CKAN into, and activate it:

   ```
   sudo mkdir -p /usr/lib/ckan/default
   sudo chown `whoami` /usr/lib/ckan/default
   virtualenv --no-site-packages /usr/lib/ckan/default
   . /usr/lib/ckan/default/bin/activate
   ```

Important: The final command above activates your virtualenv. The virtualenv has to remain active for the rest of the installation and deployment process, or commands will fail. You can tell when the virtualenv is active because its name appears in front of your shell prompt, something like this:

```
(default) $ _
```

For example, if you logout and login again, or if you close your terminal window and open it again, your virtualenv will no longer be activated. You can always reactivate the virtualenv with this command:

```
. /usr/lib/ckan/default/bin/activate
```

2. Install the CKAN source code into your virtualenv. To install the latest development version of CKAN (the most recent commit on the master branch of the CKAN git repository), run:

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

   Alternatively, to install a specific version such as CKAN 2.0 run:

   ```
   pip install -e `git+https://github.com/okfn/ckan.git@ckan-2.0#egg=ckan'
   ```

3. Install the Python modules that CKAN requires into your virtualenv:

   ```
   pip install -r /usr/lib/ckan/default/src/ckan/requirements.txt
   ```

4. Deactivate and reactivate your virtualenv, to make sure you're using the virtualenv's copies of commands like `paster` rather than any system-wide installed copies:

```
deactivate
. /usr/lib/ckan/default/bin/activate
```

1.2.3 3. 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. Create a new PostgreSQL database user called ckan_default, and enter a password for the user when prompted. You'll need this password later:

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

Create a new PostgreSQL database, called ckan_default, owned by the database user you just created:

```
sudo -u postgres createdb -O ckan_default ckan_default -E utf-8
```

1.2.4 4. Create a CKAN config file

Create a directory to contain the site's config files:

```
sudo mkdir -p /etc/ckan/default
sudo chown -R `whoami` /etc/ckan/
```

Change to the `ckan` directory and create a CKAN config file:

```
cd /usr/lib/ckan/default/src/ckan
paster make-config ckan /etc/ckan/default/development.ini
```

Edit the `development.ini` file in a text editor, changing the following options:

sqlalchemy.url This should refer to the database we created in *3. Setup a PostgreSQL database* above:

```
sqlalchemy.url = postgresql://ckan_default:pass@localhost/ckan_default
```

Replace `pass` with the password that you created in *3. Setup a PostgreSQL database* above.

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

```
sqlalchemy.url = postgresql://ckan_default:pass@<remotehost>/ckan_default?sslmode=disal
```

---

site_id Each CKAN site should have a unique `site_id`, for example:

```
ckan.site_id = default
```

1.2.5 5. Setup Solr

Follow the instructions in *Single Solr instance* or *Multiple Solr cores* to setup Solr, then change the `solr_url` option in your CKAN config file to point to your Solr server, for example:

```
solr_url=http://127.0.0.1:8983/solr
```

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 12.04 (Precise), the recommended 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.

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 Installing CKAN from Source or Installing CKAN from Package 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` to point to your machine's JDK install location. For example:

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

or:

```
JAVA_HOME=/usr/lib/jvm/java-6-openjdk-i386/
```

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 /usr/lib/ckan/default/src/ckan/ckan/config/solr/schema-2.0.xml /etc/solr/conf/sc
```

Now restart jetty:

```
sudo service jetty restart
```

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

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/ckan-schema-2.0       # Used by CKAN 2.0
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-1.4" instanceDir="ckan-schema-1.4">
            <property name="dataDir" value="/var/lib/solr/data/ckan-schema-1.4" />
        </core>
        <core name="ckan-schema-2.0" instanceDir="ckan-schema-2.0">
            <property name="dataDir" value="/var/lib/solr/data/ckan-schema-2.0" />
        </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/ckan-schema-1.4
sudo -u jetty mkdir /var/lib/solr/data/ckan-schema-2.0
```

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/ckan-schema-1.4
sudo mv /etc/solr/conf /etc/solr/ckan-schema-1.4/

sudo mkdir /usr/share/solr/ckan-schema-1.4
sudo ln -s /etc/solr/ckan-schema-1.4/conf /usr/share/solr/ckan-schema-1.4/conf
```

Now configure the core to use the data directory you have created. Edit `/etc/solr/ckan-schema-1.4/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/ckan-schema-2.0
sudo cp -R /etc/solr/ckan-schema-1.4/conf /etc/solr/ckan-schema-2.0

sudo mkdir /usr/share/solr/ckan-schema-2.0
sudo ln -s /etc/solr/ckan-schema-2.0/conf /usr/share/solr/ckan-schema-2.0/conf
```

Remember to ensure that each core points to the correct CKAN schema. To link each schema to the relevant file on the CKAN source use the following:

```
sudo rm /etc/solr/ckan-schema-2.0/conf/schema.xml
sudo ln -s /usr/lib/ckan/default/src/ckan/ckan/config/solr/schema-2.0.xml /etc/solr/ckan-sc
```

Now restart jetty:

```
sudo service jetty restart
```

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

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>
```

- When running Solr it says `Unable to find a javac compiler; com.sun.tools.javac.Main is not on the classpath. Perhaps JAVA_HOME does not point to the JDK.`

  See the note above about `JAVA_HOME`. Alternatively you may not have installed the JDK. Check by seeing if javac is installed:

```
which javac
```

If it isn't do:

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

and restart Solr.

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
ckan/config/solr/schema-2.0.xml
```

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

```
<schema name="ckan" version="2.0">
```

- 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-1.4/
http://<solr-server>/solr/ckan-schema-2.0/
```

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.

1.2.6 6. Create database tables

Now that you have a configuration file that has the correct settings for your database, you can create the database tables:

```
cd /usr/lib/ckan/default/src/ckan
paster db init -c /etc/ckan/default/development.ini
```

You should see `Initialising DB: SUCCESS`.

Tip: If the command prompts for a password it is likely you haven't set up the `sqlalchemy.url` option in your CKAN configuration file properly. See *4. Create a CKAN config file*.

1.2.7 7. Set up the DataStore

Note: Setting up the DataStore is optional. However, if you do skip this step, the DataStore features will not be available and the DataStore tests will fail.

Follow the instructions in DataStore Extension to create the required databases and users, set the right permissions and set the appropriate values in your CKAN config file.

1.2.8 8. Link to `who.ini`

`who.ini` (the Repoze.who configuration file) needs to be accessible in the same directory as your CKAN config file, so create a symlink to it:

```
ln -s /usr/lib/ckan/default/src/ckan/who.ini /etc/ckan/default/who.ini
```

1.2.9 9. You're done!

You can now use the Paste development server to serve CKAN from the command-line. This is a simple and lightweight way to serve CKAN that is useful for development and testing:

```
cd /usr/lib/ckan/default/src/ckan
paster serve /etc/ckan/default/development.ini
```

Open http://127.0.0.1:5000/ in a web browser, and you should see the CKAN front page.

Now that you've installed CKAN, you should:

- Run CKAN's tests to make sure that everything's working, see Testing CKAN.
- If you want to use your CKAN site as a production site, not just for testing or development purposes, then deploy CKAN using a production web server such as Apache or Nginx. See Deploying a Source Install.
- Begin using and customizing your site, see Getting Started.

1.3 Deploying a Source Install

Once you've installed CKAN from source by following the instructions in Installing CKAN from Source, you can follow these instructions to deploy your CKAN site using a production web server (Apache), so that it's available to the Internet.

Note: If you installed CKAN from package you don't need to follow this section, your site is already deployed using Apache and modwsgi as described below.

Because 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

This guide explains how to deploy CKAN using Apache and modwsgi on an Ubuntu server. These instructions have been tested on Ubuntu 12.04.

If run into any problems following these instructions, see *Troubleshooting* below.

1.3.1 1. Create a `production.ini` File

Create your site's `production.ini` file, by copying the `development.ini` file you created in Installing CKAN from Source earlier:

```
cp /etc/ckan/default/development.ini /etc/ckan/default/production.ini
```

1.3.2 2. Install Apache and modwsgi

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

```
sudo apt-get install apache2 libapache2-mod-wsgi
```

1.3.3 3. Install an Email Server

If one isn't installed already, install an email server to enable CKAN's email features (such as sending traceback emails to sysadmins when crashes occur, or sending new activity email notifications to users). For example, to install the Postfix email server, do:

```
sudo apt-get install postfix
```

When asked to choose a Postfix configuration, choose *Internet Site* and press return.

1.3.4 4. Create the WSGI Script File

Create your site's WSGI script file /etc/ckan/default/apache.wsgi with the following contents:

```
import os
activate_this = os.path.join(`/usr/lib/ckan/default/bin/activate_this.py')
execfile(activate_this, dict(__file__=activate_this))

from paste.deploy import loadapp
config_filepath = os.path.join(os.path.dirname(os.path.abspath(__file__)), `production.ini
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).

1.3.5 5. Create the Apache Config File

Create your site's Apache config file at /etc/apache2/sites-available/ckan_default, with the following contents:

```
<VirtualHost 0.0.0.0:80>
    ServerName default.ckanhosted.com
    ServerAlias www.default.ckanhosted.com
    WSGIScriptAlias / /etc/ckan/default/apache.wsgi

    # Pass authorization info on (needed for rest api).
    WSGIPassAuthorization On

    # Deploy as a daemon (avoids conflicts between CKAN instances).
    WSGIDaemonProcess ckan_default display-name=ckan_default processes=2 threads=15

    WSGIProcessGroup ckan_default

    ErrorLog /var/log/apache2/ckan_default.error.log
    CustomLog /var/log/apache2/ckan_default.custom.log combined
</VirtualHost>
```

Replace `default.ckanhosted.com` and `www.default.ckanhosted.com` with the domain name for your site.

This tells the Apache modwsgi module to redirect any requests to the web server to the WSGI script that you created above. Your WSGI script in turn directs the requests to your CKAN instance.

1.3.6 6. Enable Your CKAN Site

Finally, enable your CKAN site in Apache:

```
sudo a2ensite ckan_default
sudo service apache2 reload
```

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

1.3.7 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 service apache2 reload
```

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 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. `ckan_default.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

CHAPTER 2

Upgrading CKAN

This document covers CKAN releases and how to upgrade a site to a newer version of CKAN:

- *CKAN releases* describes the different types of CKAN release
- *Release process* describes the process that the CKAN dev team follows, when we make a new CKAN release
- Finally, *Upgrading CKAN* will walk you through the steps for upgrading a CKAN site to a newer version of CKAN

For a list of CKAN releases and the changes introduced in each release, see the Changelog.

2.1 CKAN releases

CKAN follows a predictable release cycle so that users can depend on stable releases of CKAN, and can plan their upgrades to new releases. The Changelog documents the main changes in each release.

Each release has a version number of the form `M.m` (eg. 2.1) or `M.m.p` (eg. 1.8.2), where `M` is the **major version**, `m` is the **minor version** and `p` is the **patch version** number. There are three types of release:

Major Releases Major releases, such as CKAN 1.0 and CKAN 2.0, increment the major version number. These releases contain major changes in the CKAN code base, with significant refactorings and breaking changes, for instance in the API or the templates. These releases are very infrequent.

Minor Releases Minor releases, such as CKAN 1.8 and CKAN 2.1, increment the minor version number. These releases are not as disruptive as major releases, but backwards-incompatible changes *may* be introduced in minor releases. The Changelog will document any breaking changes. We aim to release a minor version of CKAN roughly every three months.

Patch Releases Patch releases, such as CKAN 1.8.1 or CKAN 2.0.1, increment the patch version number. These releases do not break backwards-compatibility, they include only bug fixes and non-breaking optimizations and features. Patch releases do not contain:

- Database schema changes or migrations
- Function interface changes
- Plugin interface changes
- New dependencies
- Big refactorings or new features in critical parts of the code

Users should always run the latest patch release for the minor release they are on, as they contain important bug fixes and security updates. As they don't include backwards incompatible changes, the upgrade process (as described in Upgrading a CKAN 2 package install to a new patch release) should be straightforward.

Outdated patch releases will no longer be supported after a newer patch release has been released. For example once CKAN 2.0.2 has been released, CKAN 2.0.1 will no longer be supported.

Releases are announced on the `ckan-announce` mailing list, a low-volume list that CKAN instance maintainers can subscribe to in order to be up to date with upcoming releases. You can sign up to the list here:

http://lists.okfn.org/mailman/listinfo/ckan-announce

2.2 Release process

When the development is ready to start the process of releasing a new version of CKAN, we will:

1. Create a new release branch from the master branch, named `release-v*` where `*` is the release's version number.
2. Deploy the release branch on beta.ckan.org for testing.
3. During the next two-three weeks, we'll allow changes on the release branch only to stabilize the code, update translations and documentation, etc. (new features are usually not added on the release branch).
4. During the final week before the release, we'll only allow critical bug fixes to be committed on the release branch.

At some point during the beta period a **strings freeze** will begin. That means that no changes to translatable strings are allowed on the release branch (no new strings, or changes to existing strings). This will give translators time to update the translations on Transifex. We'll publish a **call for translations** to the ckan-dev and ckan-discuss mailing lists, announcing that the new version is ready to be translated.

At some point before the final release, we'll announce an **end of translations** after which no new translations will be pulled into the release branch. At this point we'll deploy the translations to beta.ckan.org and we'll put out a request for people to test CKAN in their languages.

The upcoming releases are announced on the ckan-announce mailing list.

Release branches are not merged back into master. All changes on a release branch are cherry-picked from master (or merged from special branches based on the release branch).

To ensure that the release guidelines are enforced one of the CKAN core developers will act as **Release Manager**. They have the final say on what is merged into the release branches.

Detailed release process instructions for CKAN Developers can be found in the Doing a CKAN Release document:

2.2.1 Doing a CKAN Release

These are the steps followed by CKAN developers to do a release. To get an overview of CKAN releases, check Upgrading CKAN.

Doing a Beta Release

Beta releases are branched off a certain point in master and will eventually become stable releases.

1. Create a new release branch:

```
git checkout -b release-v1.8
```

   Update `ckan/__init__.py` to change the version number to the new version with a *b* after it, e.g. *1.8b*. Commit the change and push the new branch to GitHub:

```
git commit -am "Update version number"
git push origin release-v1.8
```

You will probably need to update the same file on master to increase the version number, in this case ending with an *a* (for alpha).

2. During the beta process, all changes to the release branch must be cherry-picked from master (or merged from special branches based on the release branch if the original branch was not compatible).

3. As in the master branch, if some commits involving CSS changes are cherry-picked from master, the less compiling command needs to be run on the release branch. This will update the `main.css` file:

```
./bin/less --production
git commit -am "Rebuild CSS"
git push
```

There will be a final front-end build before the actual release.

4. The beta staging site (http://beta.ckan.org, currently on s084) should be updated regularly to allow user testing.

5. Once the translation freeze is in place (ie no changes to the translatable strings are allowed), strings need to be extracted and uploaded to Transifex:

   (a) Install the Babel and Transifex libraries if necessary:

```
pip install --upgrade Babel
pip install transifex-client
```

   (b) Create a `~/.transifexrc` file if necessary with your login details (`token` should be left blank):

```
[https://www.transifex.com]
hostname = https://www.transifex.com
username = <username>
password = <password>
token =
```

   (c) Extract new strings from the CKAN source code into the `ckan.pot` file. The pot file is a text file that contains the original, untranslated strings extracted from the CKAN source code.:

```
python setup.py extract_messages
```

   (d) Pull new and updated translations from Transifex into the `ckan.po` files:

```
tx pull --all --force
```

The po files are text files, one for each language CKAN is translated to, that contain the translated strings next to the originals. Translators edit the po files (on Transifex) to update the translations. We never edit the po files locally.

`--force` tells Transifex to update all `ckan.po` files, regardless of the modification time.

   (e) Run our script that checks for mistakes in the ckan.po files:

```
pip install polib
paster check-po-files ckan/i18n/*/LC_MESSAGES/ckan.po
```

If the script finds any mistakes correct them on Transifex and then run the tx pull command again, don't edit the files directly. Repeat until the script finds no mistakes.

   (f) Edit `.tx/config`, on line 4 to set the Transifex 'resource' to the new major release name (if different), using dashes instead of dots. For instance v1.2, v1.2.1 and v1.2.2 all share: `[ckan.1-2]`.

(g) Update the `ckan.po` files with the new strings from the `ckan.pot` file:

```
python setup.py update_catalog --no-fuzzy-matching
```

Any new or updated strings from the CKAN source code will get into the po files, and any strings in the po files that no longer exist in the source code will be deleted (along with their translations).

We use the `--no-fuzzy-matching` option because fuzzy matching often causes problems with Babel and Transifex.

(h) Create a new resource in the CKAN project on Transifex by pushing the new pot and po files:

```
tx push --source --translations --force
```

Because it reads the new version number in the `.tx/config` file, tx will create a new resource on Transifex rather than updating an existing resource (updating an existing resource, especially with the `--force` option, can result in translations being deleted from Transifex).

(i) Update the `ckan.mo` files by compiling the po files:

```
python setup.py compile_catalog
```

The mo files are the files that CKAN actually reads when displaying strings to the user.

(j) Commit all the above changes to git and push them to GitHub:

```
git commit -am " Update strings files before CKAN X.Y call for translations"
git push
```

(k) Announce that strings for the new release are ready for translators. Send an email to the mailing lists, tweet or post it on the blog. Make sure to post a link to the correct Transifex resource (like this one) and tell users that they can register on Transifex to contribute.

(l) A week before the translations will be closed send a reminder email.

(m) Once the translations are closed, pull the updated strings from Transifex, check them, compile and push as described in the previous steps:

```
tx pull --all --force
paster check-po-files ckan/i18n/*/LC_MESSAGES/ckan.po
python setup.py compile_catalog
git commit -am " Update translations from Transifex"
git push
```

Doing a Proper Release

Once the release branch has been thoroughly tested and is stable we can do a release.

1. Run the most thorough tests:

```
nosetests ckan/tests --ckan --ckan-migration --with-pylons=test-core.ini
```

2. Do a final build of the front-end and commit the changes:

```
paster front-end-build
git commit -am "Rebuild front-end"
```

3. Update the CHANGELOG.txt with the new version changes:
   - Add the release date next to the version number

- Add the following notices at the top of the release, reflecting whether updates in requirements, database or Solr schema are required or not:

```
Note: This version requires a requirements upgrade on source installations
Note: This version requires a database upgrade
Note: This version does not require a Solr schema upgrade
```

- Check the issue numbers on the commit messages for information about the changes. These are some helpful git commands:

```
git branch -a --merged > merged-current.txt
git branch -a --merged ckan-1.8.1 > merged-previous.txt
diff merged-previous.txt merged-current.txt

git log --no-merges release-v1.8.1..release-v2.0
git shortlog --no-merges release-v1.8.1..release-v2.0
```

4. Check that the docs compile correctly:

```
rm build/sphinx -rf
python setup.py build_sphinx
```

5. Remove the beta letter in the version number in `ckan/__init__.py` (eg 1.1b -> 1.1) and commit the change:

```
git commit -am "Update version number for release X.Y"
```

6. Tag the repository with the version number, and make sure to push it to GitHub afterwards:

```
git tag -a -m '[release]: Release tag' ckan-X.Y
git push --tags
```

7. Upload the release to PyPI:

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

   You will need a PyPI account with admin permissions on the ckan package, and your credentials should be defined on a `~/.pypirc` file, as described here If you make a mistake, you can always remove the release file on PyPI and re-upload it.

8. Enable the new version of the docs on Read the Docs (you will need an admin account):

   (a) Go to the *versions page <http://readthedocs.org/dashboard/ckan/versions/>* and enable the relevant release (make sure to use the tag, ie ckan-X.Y, not the branch, ie release-vX.Y).

   (b) If it is the latest stable release, set it to be the Default Version and check it is displayed on http://docs.ckan.org.

9. Write a CKAN Blog post and send an email to the mailing list announcing the release, including the relevant bit of changelog.

10. Cherry-pick the i18n changes from the release branch onto master.

    Generally we don't merge or cherry-pick release branches into master, but the files in ckan/i18n are an exception. These files are only ever changed on release branches following the *Doing a Beta Release* instructions above, and after a release has been finalized the changes need to be cherry-picked onto master.

    To find out what i18n commits there are on the release-v* branch that are not on master, do:

```
git log master..release-v* ckan/i18n
```

    Then `checkout` the master branch, do a `git status` and a `git pull` to make sure you have the latest commits on master and no local changes. Then use `git cherry-pick` when on the master branch to cherry-pick these commits onto master. You should not get any merge conflicts. Run the `check-po-files` command

again just to be safe, it should not report any problems. Run CKAN's tests, again just to be safe. Then do `git push origin master`.

2.3 Upgrading CKAN

This section will walk you through the steps to upgrade your CKAN site to a newer version of CKAN.

Note: Before upgrading your version of CKAN you should check that any custom templates or extensions you're using work with the new version of CKAN. For example, you could install the new version of CKAN in a new virtual environment and use that to test your templates and extensions.

Note: You should also read the Changelog to see if there are any extra notes to be aware of when upgrading to the new version.

2.3.1 1. Backup your database

You should always backup your CKAN database before upgrading CKAN. If something goes wrong with the CKAN upgrade you can use the backup to restore the database to its pre-upgrade state.

1. Activate your virtualenv and switch to the ckan source directory, e.g.:

   ```
   . /usr/lib/ckan/default/bin/activate
   cd /usr/lib/ckan/default/src/ckan
   ```

2. Backup your CKAN database using the `db dump` command, for example:

   ```
   paster db dump --config=/etc/ckan/default/development.ini my_ckan_database.pg_dump
   ```

   This will create a file called `my_ckan_database.pg_dump`, you can use the the `db load` command to restore your database to the state recorded in this file. See *db: Manage databases* for details of the `db dump` and `db load` commands.

2.3.2 2. Upgrade CKAN

The process of upgrading CKAN differs depending on whether you have a package install or a source install of CKAN, and whether you're upgrading to a *major, minor or patch release* of CKAN. Follow the appropriate one of these documents:

Upgrading a CKAN 1 package install to CKAN 2.0

Note: If you want to upgrade a CKAN 1.x package install to a newer version of CKAN 1 (as opposed to upgrading to CKAN 2), see the documentation relevant to the old CKAN packaging system.

The CKAN 2.0 package requires Ubuntu 12.04 64-bit, whereas previous CKAN packages used Ubuntu 10.04. CKAN 2.0 also introduces many backwards-incompatible feature changes (see the changelog). So it's not possible to automatically upgrade to a CKAN 2.0 package install.

However, you can install CKAN 2.0 (either on the same server that contained your CKAN 1.x site, or on a different machine) and then manually migrate your database and any custom configuration, extensions or templates to your new CKAN 2.0 site. We will outline the main steps for migrating below.

1. Create a dump of your CKAN 1.x database:

```
sudo -u ckanstd /var/lib/ckan/std/pyenv/bin/paster --plugin=ckan db dump db-1.x.dump --config=/e
```

2. If you want to install CKAN 2.0 on the same server that your CKAN 1.x site was on, uninstall the CKAN 1.x package first:

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

3. Install CKAN 2.0, either from a package install if you have Ubuntu 12.04 64-bit, or from a source install otherwise.

4. Load your database dump from CKAN 1.x into CKAN 2.0. This will migrate all of your datasets, resources, groups, tags, user accounts, and other data to CKAN 2.0. Your database schema will be automatically upgraded, and your search index rebuilt.

   First, activate your CKAN virtual environment and change to the ckan dir:

   ```
   . /usr/lib/ckan/default/bin/activate
   cd /usr/lib/ckan/default/src/ckan
   ```

   Now, load your database. **This will delete any data already present in your new CKAN 2.0 database**. If you've installed CKAN 2.0 on a different machine from 1.x, first copy the database dump file to that machine. Then run these commands:

   ```
   paster db clean -c /etc/ckan/default/production.ini
   paster db load -c /etc/ckan/default/production.ini db-1.x.dump
   ```

5. If you had any custom config settings in your CKAN 1.x instance that you want to copy across to your CKAN 2.0 instance, then update your CKAN 2.0 /etc/ckan/default/production.ini file with these config settings. Note that not all CKAN 1.x config settings are still supported in CKAN 2.0, see Config File Options for details.

   In particular, CKAN 2.0 introduces an entirely new authorization system and any custom authorization settings you had in CKAN 1.x will have to be reconsidered for CKAN 2.0. See Authorization for details.

6. If you had any extensions installed in your CKAN 1.x instance that you also want to use with your CKAN 2.0 instance, install those extensions in CKAN 2.0. Not all CKAN 1.x extensions are compatible with CKAN 2.0. Check each extension's documentation for CKAN 2.0 compatibility and install instructions.

7. If you had any custom templates in your CKAN 1.x instance, these will need to be adapted before they can be used with CKAN 2.0. CKAN 2.0 introduces an entirely new template system based on Jinja2 rather than on Genshi. See Theming for details.

Upgrading a CKAN 2 package install to a new patch release

Note: Before upgrading CKAN you should check the compatibility of any custom themes or extensions you're using, check the changelog, and backup your database. See *Upgrading CKAN*.

Patch releases are distributed in the same package as the minor release they belong to, so for example CKAN `2.0`, `2.0.1`, `2.0.2`, etc. will all be installed using the CKAN `2.0` package (`python-ckan_2.0_amd64.deb`):

1. Download the CKAN package:

```
wget http://packaging.ckan.org/python-ckan_2.0_amd64.deb
```

   You can check the actual CKAN version from a package running the following command:

```
dpkg --info python-ckan_2.0_amd64.deb
```

   Look for the `Version` field in the output:

```
...
Package: python-ckan
Version: 2.0.1-3
...
```

2. Install the package with the following command:

```
sudo dpkg -i python-ckan_2.0_amd64.deb
```

Your CKAN instance should be upgraded straight away.

Note: If you have changed the Apache or Nginx configuration files, you will get a prompt like the following, asking whether to keep your local changes or replace the files. You generally would like to keep your local changes (option `N`, which is the default), but you can look at the differences between versions selecting option `D`:

```
Configuration file `/etc/apache2/sites-available/ckan_default'
 ==> File on system created by you or by a script.
 ==> File also in package provided by package maintainer.
   What would you like to do about it ?  Your options are:
    Y or I  : install the package maintainer's version
    N or O  : keep your currently-installed version
      D     : show the differences between the versions
      Z     : start a shell to examine the situation
 The default action is to keep your current version.
*** ckan_default (Y/I/N/O/D/Z) [default=N] ?
```

Your local CKAN configuration file in /etc/ckan/default will not be replaced.

Note: The install process will uninstall any existing CKAN extensions or other libraries located in the `src` directory of the CKAN virtualenv. To enable them again, the installation process will iterate all folders in the `src` directory, reinstall the requirements listed in `pip-requirements.txt` and `requirements.txt` files and run `python setup.py develop` for each. If you are using a custom extension which does not use this requirements file names or is located elsewhere, you will need to manually reenable it.

Note: When upgrading from 2.0 to 2.0.1 you may see some vdm related warnings when installing the package:

```
dpkg: warning: unable to delete old directory '/usr/lib/ckan/default/src/vdm': Directory not emp
```

These are due to vdm not longer being installed from source. You can ignore them and delete the folder manually if you want.

---

3. Finally, restart Apache:

```
sudo service apache2 restart
```

4. You're done!

You should now be able to visit your CKAN website in your web browser and see that it's running the new version of CKAN.

Upgrading a CKAN 2 package install to a new minor release

Note: Before upgrading CKAN you should check the compatibility of any custom themes or extensions you're using, check the changelog, and backup your database. See *Upgrading CKAN*.

Each *minor release* is distributed in its own package, so for example CKAN `2.0.X` and `2.1.X` will be installed using the `python-ckan_2.0_amd64.deb` and `python-ckan_2.1_amd64.deb` packages respectively.

1. Download the CKAN package for the new minor release you want to upgrade to (replace the version number with the relevant one):

```
wget http://packaging.ckan.org/python-ckan_2.1_amd64.deb
```

2. Install the package with the following command:

```
sudo dpkg -i python-ckan_2.1_amd64.deb
```

Note: If you have changed the Apache or Nginx configuration files, you will get a prompt like the following, asking whether to keep your local changes or replace the files. You generally would like to keep your local changes (option `N`, which is the default), but you can look at the differences between versions by selecting option `D`:

```
Configuration file `/etc/apache2/sites-available/ckan_default'
 ==> File on system created by you or by a script.
 ==> File also in package provided by package maintainer.
   What would you like to do about it ?  Your options are:
    Y or I  : install the package maintainer's version
    N or O  : keep your currently-installed version
      D     : show the differences between the versions
      Z     : start a shell to examine the situation
 The default action is to keep your current version.
*** ckan_default (Y/I/N/O/D/Z) [default=N] ?
```

Your local CKAN configuration file in /etc/ckan/default will not be replaced.

Note: The install process will uninstall any existing CKAN extensions or other libraries located in the `src` directory of the CKAN virtualenv. To enable them again, the installation process will iterate over all folders in the `src` directory, reinstall the requirements listed in `pip-requirements.txt` and `requirements.txt` files and run `python setup.py develop` for each. If you are using a custom extension which does not use this requirements file name or is located elsewhere, you will need to manually reinstall it.

---

3. If there have been changes in the database schema (check the Changelog to find out) you need to update your CKAN database's schema using the `db upgrade` command.

> **Warning:** To avoid problems during the database upgrade, comment out any plugins that you have enabled in your ini file. You can uncomment them again when the upgrade finishes.

For example:

```
paster db upgrade --config=/etc/ckan/default/development.ini
```

See *db: Manage databases* for details of the `db upgrade` command.

4. If there have been changes in the Solr schema (check the Changelog to find out) you need to update your Solr schema symlink.

When *Setting up Solr* you created a symlink `/etc/solr/conf/schema.xml` linking to a CKAN Solr schema file such as /usr/lib/ckan/default/src/ckan/ckan/config/solr/schema-2.0.xml. This symlink should be updated to point to the latest schema file in /usr/lib/ckan/default/src/ckan/ckan/config/solr/, if it doesn't already.

For example, to update the symlink:

```
sudo rm /etc/solr/conf/schema.xml
sudo ln -s /usr/lib/ckan/default/src/ckan/ckan/config/solr/schema-2.0.xml /etc/solr/co
```

You will need to restart Jetty for the changes to take effect:

```
sudo service jetty restart
```

5. If you have any CKAN extensions installed from source, you may need to checkout newer versions of the extensions that work with the new CKAN version. Refer to the documentation for each extension. We recommend disabling all extensions on your ini file and re-enable them one by one to make sure they are working fine.

6. Rebuild your search index by running the `ckan search-index rebuild` command:

```
sudo ckan search-index rebuild -r
```

   See *search-index: Rebuild search index* for details of the `ckan search-index rebuild` command.

7. Finally, restart Apache:

```
sudo service apache2 restart
```

Upgrading a source install

Note: Before upgrading CKAN you should check the compatibility of any custom themes or extensions you're using, check the changelog, and backup your database. See *Upgrading CKAN*.

The process for upgrading a source install is the same, no matter what type of CKAN release you're upgrading to:

1. Activate your virtualenv and switch to the ckan source directory, e.g.:

```
. /usr/lib/ckan/default/bin/activate
cd /usr/lib/ckan/default/src/ckan
```

2. Checkout the new CKAN version from git, for example:

```
git fetch
git checkout release-v2.0
```

   If you have any CKAN extensions installed from source, you may need to checkout newer versions of the extensions at this point as well. Refer to the documentation for each extension.

3. Update CKAN's dependencies:

```
pip install --upgrade -r pip-requirements.txt
```

4. Register any new or updated plugins:

```
python setup.py develop
```

5. If you are upgrading to a new *major release* you need to update your Solr schema symlink.

   When *Setting up Solr* you created a symlink `/etc/solr/conf/schema.xml` linking to a CKAN Solr schema file such as /usr/lib/ckan/default/src/ckan/ckan/config/solr/schema-2.0.xml. This symlink should be updated to point to the latest schema file in /usr/lib/ckan/default/src/ckan/ckan/config/solr/, if it doesn't already.

   For example, to update the symlink:

```
sudo rm /etc/solr/conf/schema.xml
sudo ln -s /usr/lib/ckan/default/src/ckan/ckan/config/solr/schema-2.0.xml /etc/solr/co
```

   You will need to restart Jetty for the changes to take effect:

```
sudo service jetty restart
```

6. If you are upgrading to a new *major release* update your CKAN database's schema using the `db upgrade` command.

   Warning: To avoid problems during the database upgrade, comment out any plugins that you have enabled in your ini file. You can uncomment them again when the upgrade finishes.

   For example:

   ```
   paster db upgrade --config=/etc/ckan/default/development.ini
   ```

   See *db: Manage databases* for details of the `db upgrade` command.

7. Rebuild your search index by running the `ckan search-index rebuild` command:

   ```
   paster search-index rebuild -r --config=/etc/ckan/default/development.ini
   ```

   See *search-index: Rebuild search index* for details of the `ckan search-index rebuild` command.

8. Finally, restart your web server. For example if you have deployed CKAN using the Apache web server on Ubuntu linux, run this command:

   ```
   sudo service apache2 reload
   ```

9. You're done!

You should now be able to visit your CKAN website in your web browser and see that it's running the new version of CKAN.

CHAPTER 3

Getting Started

Once you've finished installing CKAN, this section will walk you through getting started with your new CKAN website, including creating a CKAN sysadmin user and some test data, and the basics of administering and configuring your CKAN site.

3.1 Creating a Sysadmin User

You have to use CKAN's command line interface to create your first sysadmin user, and it can also be useful to create some test data from the command line. For full documentation of CKAN's command line interface (including troubleshooting) see Command Line Interface.

Note: CKAN commands are executed using the `paster` command on the server that CKAN is installed on. Before running the paster commands below, you need to make sure that your virtualenv is activated and that you're in your ckan source directory. For example:

```
. /usr/lib/ckan/default/bin/activate
cd /usr/lib/ckan/default/src/ckan
```

You have to create your first CKAN sysadmin user from the command line. For example, to create a user called `seanh` and make him a sysadmin:

```
paster sysadmin add seanh -c /etc/ckan/default/development.ini
```

If a user called `seanh` already exists he will be promoted to a sysadmin. If the user account doesn't exist yet you'll be prompted to enter a password and the account will be created.

For a list of other command line commands for managing sysadmins, run:

```
paster sysadmin --help
```

3.2 Creating Test Data

It can be handy to have some test data to start with, to quickly check that everything works. You can add a standard set of test data to your site from the command line with the `create-test-data` command:

```
paster create-test-data -c /etc/ckan/default/development.ini
```

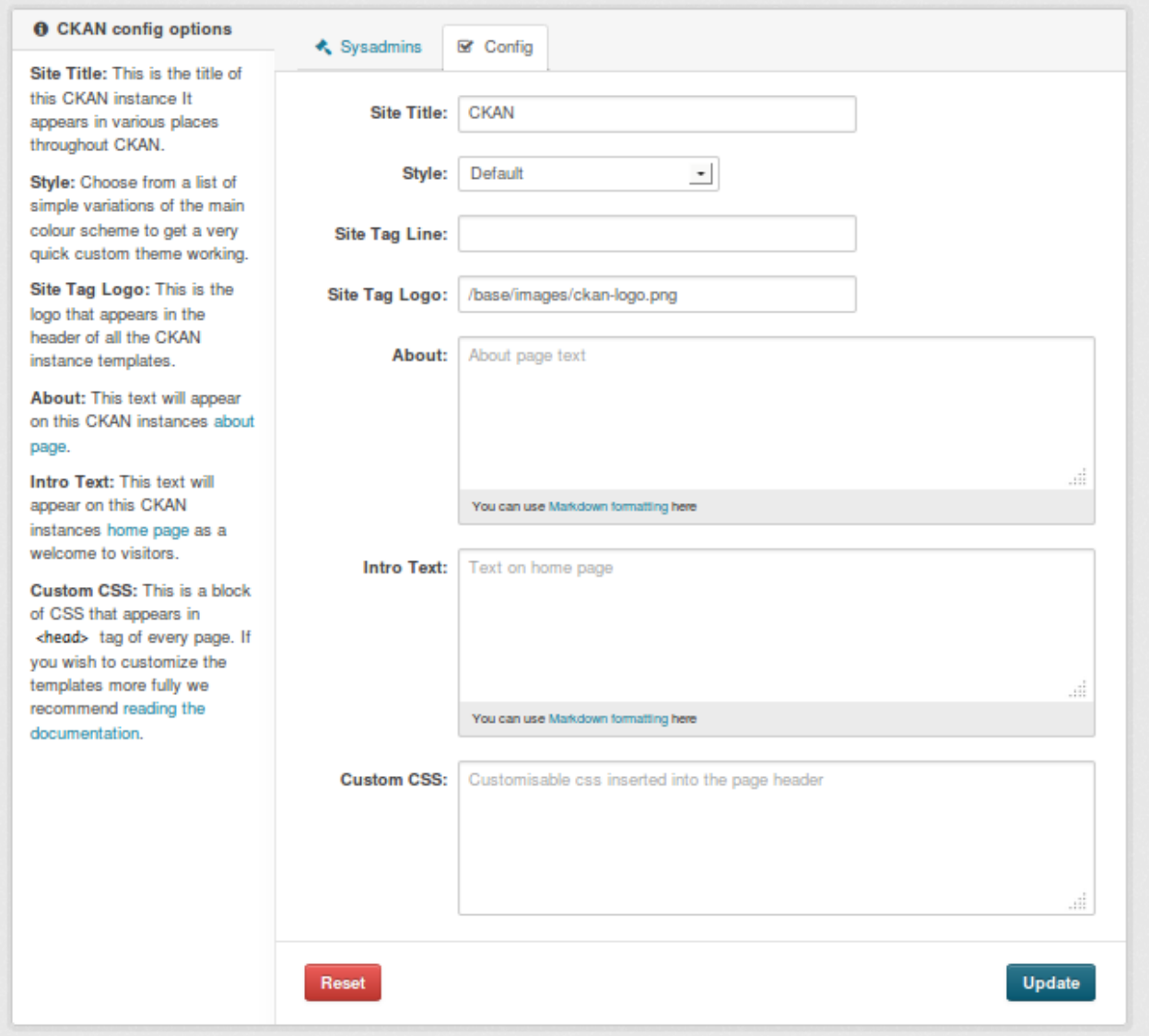
If you later want to delete this test data and start again with an empty database, you can use the `db clean` command, see *db: Manage databases*.

For a list of other command line commands for creating tests data, run:

```
paster create-test-data --help
```

3.3 The Admin Page

If you login with a sysadmin account (see *Creating a Sysadmin User*) you'll see a link in the top-right corner of the site to the administration page:



From the admin page you can set various CKAN options including the site title and logo, the about page text, the intro text on the front page, custom CSS, etc.

3.4 The Config File

All of the options that can be set in the admin page and many more can be set by editing CKAN's config file. By default, the config file is located at /etc/ckan/default/development.ini for development sites or /etc/ckan/default/production.ini for production sites. The config file can be edited in any text editor. For example, to change the title of your site you would find the `ckan.site_title` line in your config file and edit it:

```
ckan.site_title = Masaq Data Hub
```

Make sure the line is not commented-out (lines in the config file that begin with # are considered comments, so if there's a # at the start of a line you've edited, delete it), save the file, and then restart your web server for the changes to take effect. For example, if using Apache on Ubuntu:

```
sudo service apache2 reload
```

For full documentation of CKAN's config file and all the options you can set, see Config File Options.

Note: If the same option is set in both the config file and in the admin page, the admin page setting takes precedence. You can use the *Reset* button on the admin page to clear your settings, and allow settings from the config file to take effect.

CHAPTER 4

Features

The following sections describe features (both core features and built-in extensions) that come with CKAN, including how to enable, setup and configure each feature or extension.

Beyond these core features, further features can be added to CKAN by downloading and installing external extensions. A good place to find extensions is the list of extensions on the CKAN wiki.

4.1 Command Line Interface

Most common CKAN administration tasks can be carried out from the command line on the server that CKAN is installed on, using the `paster` command.

If you have trouble running paster commands, see *Troubleshooting Paster Commands* below.

Note: Before running a CKAN `paster` command, you have to activate your CKAN virtualenv and change to the `ckan` directory, for example:

```
. /usr/lib/ckan/default/bin/activate
cd /usr/lib/ckan/default/src/ckan
```

To run a paster command without activating the virtualenv first, you have to give the full path the paster script within the virtualenv, for example:

```
/usr/lib/ckan/default/bin/paster --plugin=ckan user list -c /etc/ckan/default/development.i
```

To run a paster command without changing to the ckan directory first, add the `--plugin=ckan` option to the command. For example:

```
paster --plugin=ckan user list -c /etc/ckan/default/development.ini
```

In the example commands below, we assume you're running the commands with your virtualenv activated and from your ckan directory.

The general form of a CKAN `paster` command is:

```
paster command --config=/etc/ckan/default/development.ini
```

The `--config` option tells CKAN where to find your config file, which it reads for example to know which database it should use. As you'll see in the examples below, this option can be given as `-c` for short.

`command` should be replaced with the name of the CKAN command that you wish to execute. Most commands have their own subcommands and options. For example, to print out a list of all of your CKAN site's users do:

```
paster user list -c /etc/ckan/default/development.ini
```

(Here `user` is the name of the CKAN command you're running, and `list` is a subcommand of `user`.)

For a list of all available commands, simply run `paster` on its own with no command, or see *Paster Commands Reference*. In this case we don't need the `-c` option, since we're only asking CKAN to print out information about commands, not to actually do anything with our CKAN site:

```
paster
```

Each command has its own help text, which tells you what subcommands and options it has (if any). To print out a command's help text, run the command with the `--help` option:

```
paster user --help
```

4.1.1 Troubleshooting Paster Commands

Virtualenv not activated, or not in ckan dir

Most errors with paster commands can be solved by remembering to **activate your virtual environment** and **change to the ckan directory** before running the command:

```
. /usr/lib/ckan/default/bin/activate
cd /usr/lib/ckan/default/src/ckan
```

Error messages such as the following are usually caused by forgetting to do this:

- **Command 'foo' not known** (where *foo* is the name of the command you tried to run)
- **The program 'paster' is currently not installed**
- **Command not found: paster**
- **ImportError: No module named fanstatic** (or other `ImportErrors`)

Running paster commands provided by extensions

If you're trying to run a CKAN command provided by an extension that you've installed and you're getting an error like **Command 'foo' not known** even though you've activated your virtualenv and changed to the ckan directory, this is because you need to run the extension's paster commands from the extension's source directory not CKAN's source directory. For example:

```
. /usr/lib/ckan/default/bin/activate
cd /usr/lib/ckan/default/src/ckanext-spatial
paster foo -c /etc/ckan/default/development.ini
```

This should not be necessary when using the pre-installed extensions that come with CKAN.

Alternatively, you can give the extension's name using the `--plugin` option, for example

```
paster --plugin=ckanext-foo foo -c /etc/ckan/default/development.ini
```

Todo

Running a paster shell with `paster --plugin=pylons shell -c ...`. Useful for development?

Wrong config file path

AssertionError: Config filename development.ini does not exist This means you forgot to give the `--config` or `-c` option to tell CKAN where to find your config file. (CKAN looks for a config file named `development.ini` in your current working directory by default.)

ConfigParser.MissingSectionHeaderError: File contains no section headers This happens if the config file that you gave with the `-c` or `--config` option is badly formatted, or if you gave the wrong filename.

IOError: [Errno 2] No such file or directory: '...' This means you gave the wrong path to the `--config` or `-c` option (you gave a path to a file that doesn't exist).

4.1.2 Paster Commands Reference

The following paster commands are supported by CKAN:

celeryd	Control celery daemon.
check-po-files	Check po files for common mistakes
color	Create or remove a color scheme.
create-test-data	Create test data in the database.
dataset	Manage datasets.
datastore	Perform commands to set up the datastore.
db	Perform various tasks on the database.
front-end-build	Creates and minifies css and JavaScript files
less	Compile all root less documents into their CSS counterparts
minify	Create minified versions of the given Javascript and CSS files.
notify	Send out modification notifications.
plugin-info	Provide info on installed plugins.
profile	Code speed profiler
ratings	Manage the ratings stored in the db
rdf-export	Export active datasets as RDF.
search-index	Creates a search index for all datasets
sysadmin	Gives sysadmin rights to a named user.
tracking	Update tracking statistics.
trans	Translation helper functions
user	Manage users.

celeryd: Control celery daemon

Usage:

```
celeryd <run>            - run the celery daemon
celeryd run concurrency  - run the celery daemon with
                           argument 'concurrency'
celeryd view             - view all tasks in the queue
celeryd clean            - delete all tasks in the queue
```

check-po-files: Check po files for common mistakes

Usage:

```
check-po-files [options] [FILE] ...
```

color: Create or remove a color scheme

After running this command, you'll need to regenerate the css files. See *less: Compile all root less documents into their CSS counterparts* for details.

Usage:

```
color                - creates a random color scheme
color clear          - clears any color scheme
color <'HEX'>        - uses as base color eg '#ff00ff' must be quoted.
color <VALUE>        - a float between 0.0 and 1.0 used as base hue
color <COLOR_NAME>   - html color name used for base color eg lightblue
```

create-test-data: Create test data

As the name suggests, this command lets you load test data when first setting up CKAN. See *Creating Test Data* for details.

dataset: Manage datasets

Usage:

```
dataset DATASET_NAME|ID            - shows dataset properties
dataset show DATASET_NAME|ID       - shows dataset properties
dataset list                       - lists datasets
dataset delete [DATASET_NAME|ID]   - changes dataset state to 'deleted'
dataset purge [DATASET_NAME|ID]    - removes dataset from db entirely
```

datastore: Perform commands to set up the datastore

Make sure that the datastore URLs are set properly before you run these commands.

Usage:

```
datastore set-permissions SQL_SUPER_USER

Where:
    SQL_SUPER_USER is the name of a postgres user with sufficient
                   permissions to create new tables, users, and grant
                   and revoke new permissions.  Typically, this would
                   be the "postgres" user.
```

db: Manage databases

Lets you initialise, upgrade, and dump the CKAN database.

Initialization

Before you can run CKAN for the first time, you need to run `db init` to initialize your database:

```
paster db init -c /etc/ckan/default/production.ini
```

If you forget to do this you'll see this error message in your web browser:

503 Service Unavailable: This site is currently off-line. Database is not initialised.

Cleaning

You can delete everything in the CKAN database, including the tables, to start from scratch:

> **Warning:** This will delete all data from your CKAN database!

```
paster db clean -c /etc/ckan/default/production.ini
```

After cleaning the db you must do a `db init` or `db load` before CKAN will work 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.

> **Tip:** You can also dump the database from one CKAN instance, and then load it into another CKAN instance on the same or another machine. This will even work if the CKAN instance you dumped the database from is an older version of CKAN than the one you load it into, the database will be automatically upgraded during the load command. (But you cannot load a database from a newer version of CKAN into an older version of CKAN.)

To export a dump of your CKAN database:

```
paster db dump -c /etc/ckan/default/production.ini my_database_dump.sql
```

To load it in again, you first have to clean the database (this will delete all data in the database!) and then load the file:

```
paster db clean -c /etc/ckan/default/production.ini
paster db load -c /etc/ckan/default/production.ini my_database_dump.sql
```

Exporting Datasets to JSON or CSV

You can export all of your CKAN site's datasets from your database to a JSON file using the `db simple-dump-json` command:

```
paster db simple-dump-json -c /etc/ckan/default/production.ini my_datasets.json
```

To export the datasets in CSV format instead, use `db simple-dump-csv`:

```
paster db simple-dump-csv -c /etc/ckan/default/production.ini my_datasets.csv
```

This is useful to create a simple public listing of the datasets, with no user information. Some simple additions to the Apache config can serve the dump files to users in a directory listing. To do this, add these lines to your virtual Apache config file (e.g. /etc/apache2/sites-available/ckan_default):

```
Alias /dump/ /home/okfn/var/srvc/ckan.net/dumps/

# Disable the mod_python handler for static files
<Location /dump>
    SetHandler None
    Options +Indexes
</Location>
```

> **Warning:** Don't serve an SQL dump of your database (created using the `paster db dump` command), as those contain private user information such as email addresses and API keys.

Exporting User Accounts to CSV

You can export all of your CKAN site's user accounts from your database to a CSV file using the `db user-dump-csv` command:

```
paster db user-dump-csv -c /etc/ckan/default/production.ini my_database_users.csv
```

front-end-build: Creates and minifies css and JavaScript files

Usage:

```
front-end-build
```

less: Compile all root less documents into their CSS counterparts

Usage:

```
less
```

minify: Create minified versions of the given Javascript and CSS files

Usage:

```
paster minify [--clean] PATH

For example:

paster minify ckan/public/base
paster minify ckan/public/base/css/*.css
paster minify ckan/public/base/css/red.css
```

If the –clean option is provided any minified files will be removed.

notify: Send out modification notifications

Usage:

```
notify replay    - send out modification signals. In "replay" mode,
                   an update signal is sent for each dataset in the database.
```

plugin-info: Provide info on installed plugins

As the name suggests, this commands shows you the installed plugins, their description, and which interfaces they implement

profile: Code speed profiler

Provide a ckan url and it will make the request and record how long each function call took in a file that can be read by runsnakerun.

Usage:

```
profile URL
```

The result is saved in profile.data.search. To view the profile in runsnakerun:

```
runsnakerun ckan.data.search.profile
```

You may need to install the cProfile python module.

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
```

rdf-export: Export datasets as RDF

This command dumps out all currently active datasets as RDF into the specified folder:

```
paster rdf-export /path/to/store/output
```

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 is also an option available which works like the refresh option but tries to use all processes on the computer to reindex faster:

```
paster --plugin=ckan search-index rebuild_fast --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
```

tracking: Update tracking statistics

Usage:

```
tracking update [start_date]       - update tracking stats
tracking export FILE [start_date]  - export tracking stats to a csv file
```

trans: Translation helper functions

Usage:

```
trans js      - generate the javascript translations
trans mangle  - mangle the zh_TW translations for testing
```

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
```

4.2 Authorization

Changed in version 2.0: Previous versions of CKAN used a different authorization system.

CKAN's authorization system controls which users are allowed to carry out which actions on the site. All actions that users can carry out on a CKAN site are controlled by the authorization system. For example, the authorization system controls who can register new user accounts, delete user accounts, or create, edit and delete datasets, groups and organizations.

Authorization in CKAN can be controlled in three ways:

1. Organizations
2. Configuration file options
3. Extensions

The following sections explain each of the three methods in turn.

Note: An **organization admin** in CKAN is an administrator of a particular organization within the site, with control over that organization and its members and datasets. A **sysadmin** is an administrator of the site itself. Sysadmins

can always do everything, including adding, editing and deleting datasets, organizations and groups, regardless of the organization roles and configuration options described below.

4.2.1 Organizations

Organizations are the primary way to control who can see, create and update datasets in CKAN. Each dataset can belong to a single organization, and each organization controls access to its datasets.

Datasets can be marked as public or private. Public datasets are visible to everyone. Private datasets can only be seen by logged-in users who are members of the dataset's organization. Private datasets are not shown in general dataset searches but are shown in dataset searches within the organization.

When a user joins an organization, an organization admin gives them one of three roles: member, editor or admin.

An organization **admin** can:

- View the organization's private datasets
- Add new datasets to the organization
- Edit or delete any of the organization's datasets
- Make datasets public or private.
- Add users to the organization, and choose whether to make the new user a member, editor or admin
- Change the role of any user in the organization, including other admin users
- Remove members, editors or other admins from the organization
- Edit the organization itself (for example: change the organization's title, description or image)
- Delete the organization

An **editor** can:

- View the organization's private datasets
- Add new datasets to the organization
- Edit or delete any of the organization's datasets

A **member** can:

- View the organization's private datasets.

When a user creates a new organization, they automatically become the first admin of that organization.

4.2.2 Configuration File Options

The following configuration file options can be used to customize CKAN's authorization behavior:

ckan.auth.anon_create_dataset

Example:

```
ckan.auth.anon_create_dataset = False
```

Default value: `False`

Allow users to create datasets without registering and logging in.

ckan.auth.create_unowned_dataset

Example:

```
ckan.auth.create_unowned_dataset = False
```

Default value: `True`

Allow the creation of datasets not owned by any organization.

ckan.auth.create_dataset_if_not_in_organization

Example:

```
ckan.auth.create_dataset_if_not_in_organization = False
```

Default value: `True`

Allow users who are not members of any organization to create datasets, default: true. `create_unowned_dataset` must also be True, otherwise setting `create_dataset_if_not_in_organization` to True is meaningless.

ckan.auth.user_create_groups

Example:

```
ckan.auth.user_create_groups = False
```

Default value: `True`

Allow users to create groups.

ckan.auth.user_create_organizations

Example:

```
ckan.auth.user_create_organizations = False
```

Default value: `True`

Allow users to create organizations.

ckan.auth.user_delete_groups

Example:

```
ckan.auth.user_delete_groups = False
```

Default value: `True`

Allow users to delete groups.

ckan.auth.user_delete_organizations

Example:

```
ckan.auth.user_delete_organizations = False
```

Default value: `True`

Allow users to delete organizations.

ckan.auth.create_user_via_api

Example:

```
ckan.auth.create_user_via_api = False
```

Default value: `False`

Allow new user accounts to be created via the API.

4.2.3 Extensions

CKAN allows extensions to change the authorization rules used. Please see individual extensions for details.

Todo

Insert cross-reference to `IAuthFunctions` docs.

4.3 Data Viewer

The CKAN resource page can contain a preview of the resource's data. This works by either:

1. Embedding the data into the page, either directly or by loading the data in an iframe.
2. Using a custom widget (such as Recline) to view the data.

Generally, the decision as to which action to take is determined by the type of resource being viewed. In general, images will be directly embedded, unstructured or plain text files will be loaded in an iframe, and more complex data types will need to use a custom widget.

The data preview functionality that is provided by CKAN is described in the following sections:

- *Viewing images and text files*
- *Viewing structured data: the Data Explorer*
- *Viewing highlighted XML, JSON and plain text data*
- *Viewing PDF documents*
- *Viewing remote resources: the resource proxy*
- *Embedding Previews In Other Web Pages*

These sections list the resource formats that each extension can preview and provide instructions for how to enable each extension. It is also possible for developers to create new extensions that can preview different types of resources. For more information on this topic see Writing Extensions.

4.3.1 Viewing images and text files

Configuration required: None. Images and text files (that match one of the file types given below) will be previewed automatically by default.

Resource formats: images, plain text (details below).

By default, the following types of resources will be embedded directly into the resource read page:

- `png`
- `jpg`
- `jpeg`
- `gif`

The types of resources that are embedded directly can be specified in the CKAN config file. See *ckan.preview.direct* for more information.

The following types of resources will be loaded in an iframe if there is no extension that can preview these types:

- `plain`
- `txt`
- `html`
- `htm`
- `xml`
- `rdf+xml`
- `owl+xml`
- `n3`
- `n-triples`
- `turtle`
- `atom`
- `rss`

The types of resources that are loaded in an iframe can be specified in the CKAN config file. See *ckan.preview.loadable* for more information.

Note that these documents will be directly linked by the browser, so the way in which they are shown may vary. If you want to ensure for instance that XML based documents are correctly previewed, have a look at *Viewing highlighted XML, JSON and plain text data*.

4.3.2 Viewing structured data: the Data Explorer

New in version 2.0: the `recline_preview` extension is new in CKAN 2.0.

Configuration required: The `recline_preview` extension must be added to `ckan.plugins` in your CKAN configuration file. This extension is part of CKAN and so does not need to be installed separately.

Resource formats: DataStore, `csv`, `xls`.

Structured data can be previewed using the Recline Data Explorer. The Data Explorer provides a rich, queryable view of the data, and allows the data to be filtered, graphed and mapped.

To be viewed, the data must either be:

1. In the CKAN DataStore. This is the recommended way to preview structured data.

Or:

2. In `csv` or `xls` format. In this case, CKAN will first have to try to convert the file into a more structured format by using the Dataproxy. This is an automatic process that does not require any additional configuration. However, as the resource must be downloaded by the Dataproxy service and then analysed before it is viewed, this option is generally slower and less reliable than viewing data that is in the DataStore.

4.3.3 Viewing highlighted XML, JSON and plain text data

Configuration required: The `text_preview` extension must be added to `ckan.plugins` in your CKAN configuration file. This extension is part of CKAN and does not need to be installed separately.

Resource formats:

- **`json`, `gjson`, `geojson`** (can be configured by setting `ckan.preview.json_formats`)
- **`jsonp`** (can be configured by setting `ckan.preview.jsonp_formats`)
- **`xml`, `rdf`, `rdf+xml`, `owl+xml`, `atom`, `rss`** (can be configured by setting `ckan.preview.xml_formats`)
- **`text/plain`, `txt`, `plain`** (can be configured by setting `ckan.preview.text_formats`)

The `text_preview` extension provides previews of many file types that have been added to a CKAN instance. To view the data the resource format must be set to one of the resource formats from above (case insensitive).

See also:

The resourceproxy extension

> If you want to preview linked-to text files (and not only files that have been uploaded to CKAN) you need to enable the `resource_proxy` extension as well.

4.3.4 Viewing PDF documents

Configuration required: The `pdf_preview` extension must be added to `ckan.plugins` in your CKAN configuration file. This extension is part of CKAN and does not need to be installed separately.

Resource formats: `pdf`, `x-pdf`, `acrobat`, `vnd.pdf`.

The `pdf_preview` extension provides previews of any `pdf` documents that have been added to a CKAN instance. This extension uses Mozilla's pdf.js library.

See also:

The resourceproxy extension

> If you want to preview linked-to PDF files (and not only files that have been uploaded to CKAN) you need to enable the `resource_proxy` extension as well.

4.3.5 Viewing remote resources: the resource proxy

Configuration required: The `resource_proxy` extension must be added to `ckan.plugins` in your CKAN configuration file. This extension is part of CKAN and so does not need to be installed separately.

This extension must be enabled if you wish to preview resources that are on a different domain. That means if this extension is not enabled, e.g. PDF, or JSON files that are on `www.example.com` while CKAN is on `www.ckan.org` cannot be previewed by any extension.

Previewing is prevented by the same origin policy which prevents files from different domains (different *origins*) from being loaded into browsers. This extension gets around the same origin policy by pretending that all files are served from the same domain (same *origin*) that CKAN is on (e.g. `www.ckan.org`).

If you are writing a custom preview extension that requires resources to be proxied, you need to replace the URL that is used to load the file. This can be done using the function `ckanext.resourceproxy.plugin.get_proxified_resource_url()`. To find out whether the resource proxy is enabled, check `ckan.resource_proxy_enabled` from the config. You can find a complete example in the CKAN source.

4.3.6 Embedding Previews In Other Web Pages

Changed in version 2.0: The URL that is used to obtain the contents of the resource preview has changed from `/dataset/{name}/resource/{resource_id}/embed` to `/dataset/{name}/resource/{resource_id}/preview`.

For each resource, the preview content can be viewed at `/dataset/{dataset id}/resource/{resource id}/preview`. The preview content can therefore be embedded in other web pages by loading the contents of this URL in an iframe.

4.4 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.

4.4.1 Setup the FileStore with Local File Storage

To setup CKAN's FileStore with local file storage:

1. Create the directory where CKAN will store uploaded files:

   ```
   sudo mkdir -p /var/lib/ckan/default
   ```

2. Add the following lines to your CKAN config file, after the `[app:main]` line:

   ```
   ofs.impl = pairtree
   ofs.storage_dir = /var/lib/ckan/default
   ```

3. Set the permissions of the `storage_dir`. For example if you're running CKAN with Apache, then Apache's user (`www-data` on Ubuntu) must have read, write and execute permissions for the `storage_dir`:

   ```
   sudo chown www-data /var/lib/ckan/default
   sudo chmod u+rwx /var/lib/ckan/default
   ```

4. Make sure you've set *ckan.site_url* in your config file.

5. Restart your web server, for example to restart Apache:

   ```
   sudo service apache2 reload
   ```

4.4.2 Setup the FileStore with 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 = ....
```

4.4.3 FileStore Web Interface

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

4.4.4 FileStore API

CKAN's FileStore API lets you upload files to CKAN's FileStore. If you're looking for an example, ckanclient contains Python code for uploading a file to CKAN using the FileStore API.

FileStore 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).

FileStore Form Authentication API

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.
```

FileStore 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.
```

4.4.5 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.

4.5 DataStore Extension

Todo

What features does the datastore actually provide that users care about? Why would they want to use it?

- API for reading, writing data without downloading, uploading entire file

- Enables Recline previews
- API for searching data, including search across resources

The CKAN DataStore provides a database for structured storage of data together with a powerful Web-accessible Data API, all seamlessly integrated into the CKAN interface and authorization system. At the same time, we kept the layer between the underlying database and the user as thin as possible.

4.5.1 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 file 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.

4.5.2 Setting up the DataStore

Note: The DataStore requires PostgreSQL 9.0 or later. It is possible to use the DataStore on versions prior to 9.0 (for example 8.4). However, the `datastore_search_sql()` will not be available and the set-up is slightly different. Make sure, you read *Legacy mode: use the DataStore with old PostgreSQL versions* for more details.

Warning: The DataStore does not support hiding resources in a private dataset.

1. Enable the plugin

Add the `datastore` plugin to your CKAN config file:

```
ckan.plugins = datastore
```

2. Set-up the database

Warning: Make sure that you follow the steps in *Set Permissions* below correctly. Wrong settings could lead to serious security issues.

The DataStore requires a separate PostgreSQL database to save the resources to.

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 datastore setup.

Create users and databases

Tip: If your CKAN database and DataStore databases are on different servers, then you need to create a new database user on the server where the DataStore database will be created. As in Installing CKAN from Source we'll name the database user ckan_default:

```
sudo -u postgres createuser -S -D -R -P -l ckan_default
```

Create a database_user called datastore_default. This user will be given read-only access to your DataStore database in the *Set Permissions* step below:

```
sudo -u postgres createuser -S -D -R -P -l datastore_default
```

Create the database (owned by ckan_default), which we'll call datastore_default:

```
sudo -u postgres createdb -O ckan_default datastore_default -E utf-8
```

Set URLs

Now, uncomment the *ckan.datastore.write_url* and *ckan.datastore.read_url* lines in your CKAN config file and edit them if necessary, for example:

```
ckan.datastore.write_url = postgresql://ckan_default:pass@localhost/datastore_default
ckan.datastore.read_url = postgresql://datastore_default:pass@localhost/datastore_default
```

Replace `pass` with the passwords you created for your ckan_default and datastore_default database users.

Set Permissions

Tip: See *Legacy mode: use the DataStore with old PostgreSQL versions* if these steps continue to fail or seem too complicated for your set-up. However, keep in mind that the legacy mode is limited in its capabilities.

Once the DataStore database and the users are created, the permissions on the DataStore and CKAN database have to be set. Since there are different set-ups, there are different ways of setting the permissions. Only **one** of the options should be used.

Option 1: Paster command This option is preferred if CKAN and PostgreSQL are on the same server.

To set the permissions, use this paster command after you've set the database URLs (make sure to have your virtualenv activated):

```
paster datastore set-permissions postgres -c /etc/ckan/default/development.ini
```

The `postgres` in this command should be the name of a postgres user with permission to create new tables and users, grant permissions, etc. Typically this user is called "postgres". See `paster datastore set-permissions -h`.

Option 2: Command line tool This option should be used if the CKAN server is different from the database server.

Copy the content from the `datastore/bin/` directory to the database server. Then run the command line tool `datastore_setup.py` to set the permissions on the database. To see all available options, run:

```
python datastore_setup.py -h
```

Once you are confident that you know the right names, set the permissions (assuming that the CKAN database is called ckan_default and the CKAN PostgreSQL user is called ckan_default):

```
python datastore_setup.py ckan_default datastore_default ckan_default ckan_default datasto
```

Option 3: SQL script This option is for more complex set-ups and requires understanding of SQL and PostgreSQL.

Copy the `set_permissions.sql` file to the server that the database runs on. Make sure you set all variables in the file correctly and comment out the parts that are not needed for you set-up. Then, run the script:

```
sudo -u postgres psql postgres -f set_permissions.sql
```

3. Test the set-up

The DataStore is now set-up. To test the set-up, (re)start CKAN and run the following command to list all resources that are in the DataStore:

```
curl -X GET "http://127.0.0.1:5000/api/3/action/datastore_search?resource_id=_table_metadata"
```

This should return a JSON page without errors.

To test the whether the set-up allows writing, you can create a new resource in the DataStore. To do so, run the following command:

```
curl -X POST http://127.0.0.1:5000/api/3/action/datastore_create -H "Authorization: {YOUR-API-KEY}" -
```

Replace `{YOUR-API-KEY}` with a valid API key and `{RESOURCE-ID}` with a resource id of an existing CKAN resource.

A table named after the resource id should have been created on your DataStore database. Visiting this URL should return a response from the DataStore with the records inserted above:

```
http://127.0.0.1:5000/api/3/action/datastore_search?resource_id={RESOURCE_ID}
```

You can now delete the DataStore table with:

```
curl -X POST http://127.0.0.1:5000/api/3/action/datastore_delete -H "Authorization: {YOUR-API-KEY}" -
```

To find out more about the DataStore API, see *The DataStore API*.

Legacy mode: use the DataStore with old PostgreSQL versions

Tip: The legacy mode can also be used to simplify the set-up since it does not require you to set the permissions or create a separate user.

The DataStore can be used with a PostgreSQL version prior to 9.0 in *legacy mode*. Due to the lack of some functionality, the `datastore_search_sql()` and consequently the *HTSQL Support* cannot be used. To enable the legacy mode, remove the declaration of the `ckan.datastore.read_url`.

The set-up for legacy mode is analogous to the normal set-up as described above with a few changes and consists of the following steps:

1. Enable the plugin
2. The legacy mode is enabled by **not** setting the `ckan.datastore.read_url`
3. Set-Up the database
   (a) Create a separate database
   (b) Create a write user on the DataStore database (optional since the CKAN user can be used)
4. Test the set-up

There is no need for a read-only user or special permissions. Therefore the legacy mode can be used for simple set-ups as well.

4.5.3 The DataStore API

The DataStore API allows tabular data to be stored inside CKAN quickly and easily. Each resource in a CKAN instance can have an associated DataStore table. The API for using the DataStore is outlined below.

Making a DataStore API Request

Making a DataStore API request is the same as making an Action API request: you post a JSON dictionary in an HTTP POST request to an API URL, and the API also returns its response in a JSON dictionary. See the The CKAN API for details.

API Reference

Note: Lists can always be expressed in different ways. It is possible to use lists, comma separated strings or single items. These are valid lists: `['foo', 'bar']`, `'foo, bar'`, `"foo", "bar"` and `'foo'`. Additionally, there are several ways to define a boolean value. `True`, `on` and `1` are all vaid boolean values.

Note: The table structure of the DataStore is explained in *Internal structure of the database*.

Download resource as CSV

A DataStore resource can be downloaded in the CSV file format from `{CKAN-URL}/datastore/dump/{RESOURCE-ID}`.

Fields

Fields define the column names and the type of the data in a column. A field is defined as follows:

```
{
    "id":    # a string which defines the column name
    "type":  # the data type for the column
}
```

Field **types are optional** and will be guessed by the DataStore from the provided data. However, setting the types ensures that future inserts will not fail because of wrong types. See *Field types* for details on which types are valid.

Example:

```
[
    {
        "id": "foo",
        "type": "int4"
    },
    {
        "id": "bar"
        # type is optional
    }
]
```

Records

A record is the data to be inserted in a table and is defined as follows:

```
{
    "<id>":  # data to be set
    # .. more data
}
```

Example:

```
[
    {
        "foo": 100,
        "bar": "Here's some text"
    },
    {
        "foo": 42
    }
]
```

Field types

The DataStore supports all types supported by PostgreSQL as well as a few additions. A list of the PostgreSQL types can be found in the type section of the documentation. Below you can find a list of the most common data types. The `json` type has been added as a storage for nested data.

In addition to the listed types below, you can also use array types. They are defines by prepending a _ or appending `[]` or `[n]` where n denotes the length of the array. An arbitrarily long array of integers would be defined as `int[]`.

text Arbitrary text data, e.g. `Here's some text`.

json Arbitrary nested json data, e.g `{"foo": 42, "bar": [1, 2, 3]}`. Please note that this type is a custom type that is wrapped by the DataStore.

date Date without time, e.g `2012-5-25`.

time Time without date, e.g `12:42`.

timestamp Date and time, e.g `2012-10-01T02:43Z`.

int Integer numbers, e.g `42`, `7`.

float Floats, e.g. `1.61803`.

bool Boolean values, e.g. `true`, `0`

You can find more information about the formatting of dates in the date/time types section of the PostgreSQL documentation.

Resource aliases

A resource in the DataStore can have multiple aliases that are easier to remember than the resource id. Aliases can be created and edited with the `datastore_create()` API endpoint. All aliases can be found in a special view called `_table_metadata`. See *Internal structure of the database* for full reference.

HTSQL Support

The ckanext-htsql extension adds an API action that allows a user to search data in a resource using the HTSQL query expression language. Please refer to the extension documentation to know more.

Comparison of different querying methods

The DataStore supports querying with multiple API endpoints. They are similar but support different features. The following list gives an overview of the different methods.

	`datastore_search()`	`datastore_search_sql()`	*HTSQL*
Ease of use	Easy	Complex	Medium
Flexibility	Low	High	Medium
Query language	Custom (JSON)	SQL	HTSQL
Join resources	No	Yes	No

Internal structure of the database

The DataStore is a thin layer on top of a PostgreSQL database. Each DataStore resource belongs to a CKAN resource. The name of a table in the DataStore is always the resource id of the CKAN resource for the data.

As explained in *Resource aliases*, a resource can have mnemonic aliases which are stored as views in the database.

All aliases (views) and resources (tables respectively relations) of the DataStore can be found in a special view called `_table_metadata`. To access the list, open `http://{YOUR-CKAN-INSTALLATION}/api/3/action/datastore_search?resource_id=_table_metadata`.

`_table_metadata` has the following fields:

_id Unique key of the relation in `_table_metadata`.

alias_of Name of a relation that this alias point to. This field is `null` iff the name is not an alias.

name Contains the name of the alias if alias_of is not null. Otherwise, this is the resource id of the CKAN resource for the DataStore resource.

oid The PostgreSQL object ID of the table that belongs to name.

4.5.4 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

4.6 Apps & Ideas

Since 1.7 CKAN has a feature called Apps & Ideas which allows users to provide information on apps, ideas, visualizations, articles etc that are related to a specific dataset. Once created these items will be shown against the dataset but also shown on the apps dashboard which will allow users to filter the results based on popularity, or type, or the data when the items were created.

This feature is enabled by default but can be disabled using the *ckan.dataset.show_apps_ideas* setting to hide the tab on the dataset pages.

4.7 Tag Vocabularies

New in version 1.7.

CKAN sites can have *tag vocabularies*, which are a way of grouping related tags together into custom fields.

For example, if you were making a site for music datasets. you might use a tag vocabulary to add two fields *Genre* and *Composer* to your site's datasets, where each dataset can have one of the values *Avant-Garde*, *Country* or *Jazz* in its genre field, and one of the values *Beethoven*, *Wagner*, or *Tchaikovsky* in its composer field. In this example, genre and composer would be vocabularies and the values would be tags:

- Vocabulary: Genre
  - Tag: Avant-Garde
  - Tag: Country
  - Tag: Jazz
- Vocabulary: Composer
  - Tag: Beethoven
  - Tag: Wagner
  - Tag: Tchaikovsky

Ofcourse, you could just add Avant-Garde, Beethoven, etc. to datasets as normal CKAN tags, but using tag vocabularies lets you define Avant-Garde, Country and Jazz as genres and Beethoven, Wagner and Tchaikovsky as composers, and lets you enforce restrictions such as that each dataset must have a genre and a composer, and that no dataset can have two genres or two composers, etc.

Another example use-case for tag vocabularies would be to add a *Country Code* field to datasets defining the geographical coverage of the dataset, where each dataset is assigned a country code such as *en*, *fr*, *de*, etc. See `ckanext/example_idatasetform` for a working example implementation of country codes as a tag vocabulary.

4.7.1 Properties of Tag Vocabularies

- A CKAN website can have any number of vocabularies.
- Each vocabulary has an ID and name.

- Each tag either belongs to a vocabulary, or can be a *free tag* that doesn't belong to any vocabulary (i.e. a normal CKAN tag).
- A dataset can have more than one tag from the same vocabulary, and can have tags from more than one vocabulary.

4.7.2 Using Vocabularies

To add a tag vocabulary to a site, a CKAN sysadmin must:

1. Call the `vocabulary_create()` action of the CKAN API to create the vocabulary and tags. See The CKAN API.
2. Implement an `IDatasetForm` plugin to add a new field for the tag vocabulary to the dataset schema. See Writing Extensions.
3. Provide custom dataset templates to display the new field to users when adding, updating or viewing datasets in the CKAN web interface. See Theming.

See `ckanext/example_idatasetform` for a working example of these steps.

4.8 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.

4.8.1 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*.

(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>

...
```

4.9 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.

4.9.1 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.

4.9.2 Accessing Linked Data

To access linked data versions just access the 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
```

4.9.3 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=c
      </dcat:Distribution>
  </dcat:distribution>
  <dcat:distribution>
      <dcat:Distribution>
          <dcat:accessURL rdf:resource="https://api.scraperwiki.com/api/1.0/datastore/sqlite?format=c
      </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>
```

4.10 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)

4.10.1 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.

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
```

4.10.2 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)

4.11 Email Notifications

CKAN can send email notifications to users, for example when a user has new activities on her dashboard. Once email notifications have been enabled by a site admin, each user of a CKAN site can turn email notifications on or off for herself by logging in and editing her user preferences. To enable email notifications for a CKAN site, a sysadmin must:

1. Setup a cron job or other scheduled job on a server to call CKAN's `send_email_notifications` API action at regular intervals (e.g. hourly) and send any pending email notifications to users.

   On most UNIX systems you can setup a cron job by running `crontab -e` in a shell to edit your crontab file, and adding a line to the file to specify the new job. For more information run `man crontab` in a shell.

   CKAN API actions can be called via the `paster post` command, which simulates an HTTP-request. For example, here is a crontab line to send out CKAN email notifications hourly:

```
    @hourly echo '{}' | /usr/lib/ckan/bin/paster --plugin=ckan post -c /etc/ckan/production.ini /api
```

   The `@hourly` can be replaced with `@daily`, `@weekly` or `@monthly`.

   Warning: CKAN will not send email notifications for events older than the time period specified by the `ckan.email_notifications_since` config setting (default: 2 days), so your cron job should run more frequently than this. `@hourly` and `@daily` are good choices.

Note: Since `send_email_notifications` is an API action, it can be called from a machine other than the server on which CKAN is running, simply by POSTing an HTTP request to the CKAN API (you must be a sysadmin to call this particular API action). See The CKAN API.

2. CKAN will not send out any email notifications, nor show the email notifications preference to users, unless the *ckan.activity_streams_email_notifications* option is set to `True`, so put this line in the `[app:main]` section of your CKAN config file:

```
ckan.activity_streams_email_notifications = True
```

3. Make sure that *ckan.site_url* is set correctly in the `[app:main]` section of your CKAN configuration file. This is used to generate links in the bodies of the notification emails. For example:

```
ckan.site_url = http://publicdata.eu
```

4. Make sure that *smtp.mail_from* is set correctly in the `[app:main]` section of your CKAN configuration file. This is the email address that CKAN's email notifications will appear to come from. For example:

```
smtp.mail_from = mailman@publicdata.eu
```

   This is combined with your *ckan.site_title* to form the `From:` header of the email that are sent, for example:

```
From: PublicData.eu <mailmain@publicdata.eu>
```

5. If you do not have an SMTP server running locally on the machine that hosts your CKAN instance, you can change the *Email Settings* to send email via an external SMTP server. For example, these settings in the `[app:main]` section of your configuration file will send emails using a gmail account (not recommended for production websites!):

```
smtp.server = smtp.gmail.com:587
smtp.starttls = True
smtp.user = your_username@gmail.com
smtp.password = your_gmail_password
smtp.mail_from = your_username@gmail.com
```

6. For the new configuration to take effect you need to restart the web server. For example if your are using Apache on Ubuntu, run this command in a shell:

```
sudo service apache2 reload
```

4.12 Page View Tracking

CKAN can track visits to pages of your site and use this tracking data to:

- Sort datasets by popularity
- Highlight popular datasets and resources
- Show view counts next to datasets and resources
- Show a list of the most popular datasets
- Export page-view data to a CSV file

See also:

ckanext-googleanalytics A CKAN extension that integrates Google Analytics into CKAN.

4.12.1 Enabling Page View Tracking

To enable page view tracking:

1. Set *ckan.tracking_enabled* to true in the `[app:main]` section of your CKAN configuration file (e.g. `development.ini` or `production.ini`):

```
[app:main]
ckan.tracking_enabled = true
```

   Save the file and restart your web server. CKAN will now record raw page view tracking data in your CKAN database as pages are viewed.

2. Setup a cron job to update the tracking summary data.

   For operations based on the tracking data CKAN uses a summarised version of the data, not the raw tracking data that is recorded "live" as page views happen. The `paster tracking update` and `paster search-index rebuild` commands need to be run periodicially to update this tracking summary data.

   You can setup a cron job to run these commands. On most UNIX systems you can setup a cron job by running `crontab -e` in a shell to edit your crontab file, and adding a line to the file to specify the new job. For more information run `man crontab` in a shell. For example, here is a crontab line to update the tracking data and rebuild the search index hourly:

```
@hourly /usr/lib/ckan/bin/paster --plugin=ckan tracking update -c /etc/ckan/production.ini && /u
```

   Replace `/usr/lib/ckan/bin/` with the path to the `bin` directory of the virtualenv that you've installed CKAN into, and replace `/etc/ckan/production.ini` with the path to your CKAN configuration file.

   The `@hourly` can be replaced with `@daily`, `@weekly` or `@monthly`.

4.12.2 Retrieving Tracking Data

Tracking summary data for datasets and resources is available in the dataset and resource dictionaries returned by, for example, the `package_show()` API:

```
"tracking_summary": {
    "recent": 5,
    "total": 15
},
```

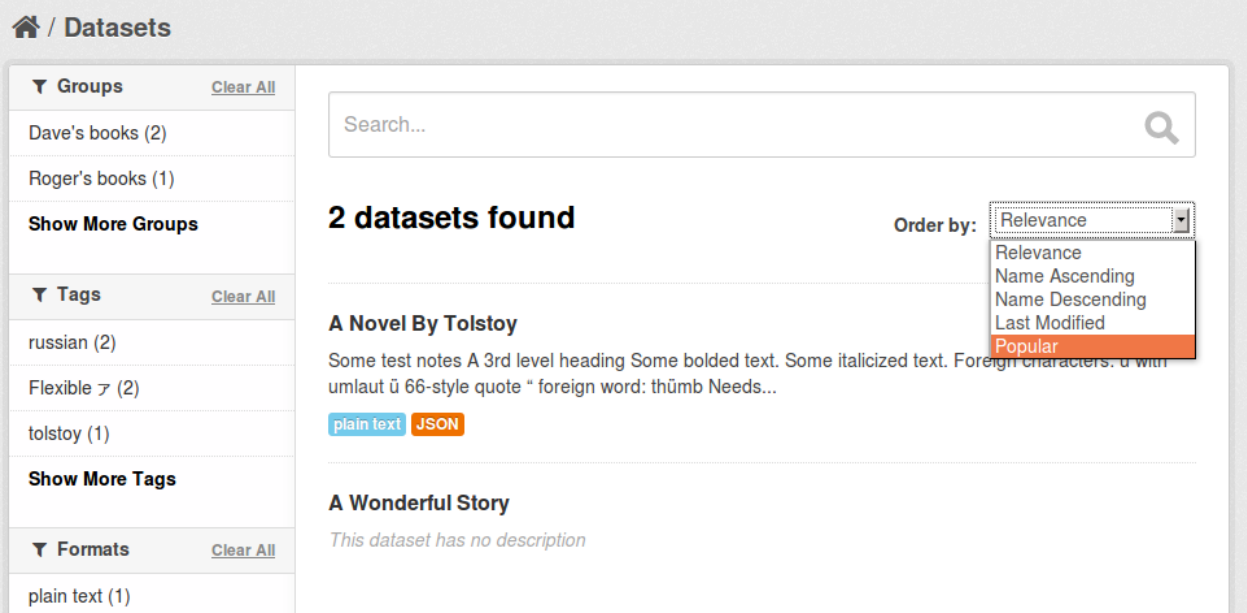
This can be used, for example, by custom templates to show the number of views next to datasets and resources. A dataset or resource's `recent` count is its number of views in the last 14 days, the `total` count is all of its tracked views (including recent ones).

You can also export tracking data for all datasets to a CSV file using the `paster tracking export` command. For details, run `paster tracking -h`.

Note: Repeatedly visiting the same page will not increase the page's view count! Page view counting is limited to one view per user per page per day.

4.12.3 Sorting Datasets by Popularity

Once you've enabled page view tracking on your CKAN site, you can view datasets most-popular-first by selecting `Popular` from the `Order by:` dropdown on the dataset search page:



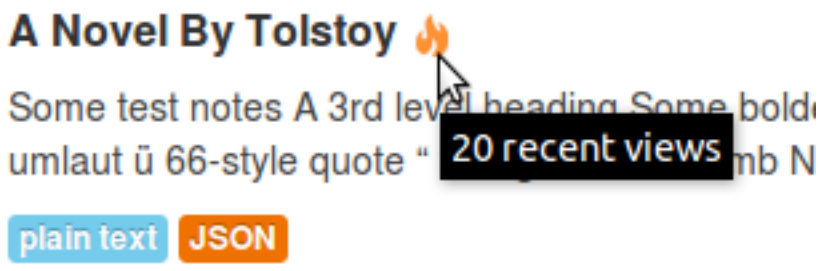
The datasets are sorted by their number of recent views.

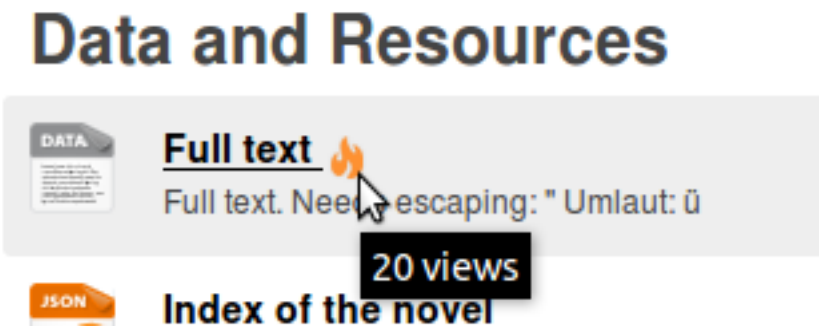
You can retrieve datasets most-popular-first from the CKAN API by passing `'sort': 'views_recent desc'` to the `package_search()` action. This could be used, for example, by a custom template to show a list of the most popular datasets on the site's front page.

Tip: You can also sort datasets by total views rather than recent views. Pass `'sort': 'views_total desc'` to the `package_search()` API, or use the URL `/dataset?q=&sort=views_total+desc` in the web interface.

4.12.4 Highlighting Popular Datasets and Resources

Once you've enabled page view tracking on your CKAN site, popular datasets and resources (those with more than 10 views) will be highlighted with a "popular" badge and a tooltip showing the number of views:





4.13 Multilingual Extension

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.

4.13.1 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 The CKAN API for more details.

4.13.2 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 Command Line Interface for more details.

4.13.3 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 *Installing Additional Dependencies* for more information.

4.14 Stats Extension

CKAN's stats extension analyzes your CKAN database and displays several tables and graphs with statistics about your site, including:

- Total number of datasets
- Dataset revisions per week
- Top-rated datasets
- Most-edited Datasets
- Largest groups
- Top tags
- Users owning most datasets

See also:

CKAN's *built-in page view tracking feature*, which tracks visits to pages.

See also:

ckanext-googleanalytics A CKAN extension that integrates Google Analytics into CKAN.

4.14.1 Enabling the Stats Extension

To enable the stats extensions add `stats` to the *ckan.plugins* option in your CKAN config file, for example:

```
ckan.plugins = stats
```

If you also set the *ckanext.stats.cache_enabled* option to `true`, CKAN will cache the stats for one day instead of calculating them each time a user visits the stats page.

4.14.2 Viewing the Statistics

To view the statistics reported by the stats extension, visit the `/stats` page, for example: http://demo.ckan.org/stats

CHAPTER 5

Writing Extensions

Note: A CKAN **extension** is a Python package that contains one or more **plugins**. A plugin is a class that implements one or more of CKAN's **plugin interfaces** to customize CKAN or add new features.

5.1 Plugins: An Overview

Plugins are created as classes inheriting from either the *Plugin* or *SingletonPlugin* base classes. Most Extensions use the *SingletonPlugin* base class and we advise you to use this if possible.

Having created your class you need to inherit from one or more plugin interfaces to allow CKAN to interact with your extension. When specifying the interfaces that will be implemented you must remember to either (a) define all methods required by the interface or (b) use the *inherits=True* parameter which will use the interfaces default methods for any that you have not defined.

Note: When writing extensions it is important to keep your code separate from CKAN by not importing ckan modules, so that internal CKAN changes do not break your code between releases. You can however import ckan.plugins without this risk.

5.2 Libraries Available To Extensions

As well as using the variables made available to them by implementing various plugin hooks, extensions will likely want to be able to use parts of the CKAN core library. To allow this, CKAN provides a stable set of modules that extensions can use safe in the knowledge the interface will remain stable, backward-compatible and with clear deprecation guidelines as development of CKAN core progresses. This interface is available in Plugins Toolkit.

5.2.1 Plugins Toolkit

To allow a safe way for extensions to interact with ckan a toolkit is provided. We aim to keep this toolkit stable across ckan versions so that extensions will work across different versions of ckan.

Note: It is advised that when writing extensions that all interaction with ckan is done via the toolkit so that they do not break when new versions of ckan are released.

Over time we will be expanding the functionality available via this toolkit.

Example extension that registers a new helper function available to templates via h.example_helper()

```
import ckan.plugins as p


class ExampleExtension(p.SingletonPlugin):

    p.implements(p.IConfigurer)
    p.implements(p.ITemplateHelpers)

    def update_config(self, config):
        # add template directory that contains our snippet
        p.toolkit.add_template_directory(config, 'templates')

    @classmethod
    def example_helper(cls, data=None):
        # render our custom snippet
        return p.toolkit.render_snippet('custom_snippet.html', data)


    def get_helpers(self):
        # register our helper function
        return {'example_helper': self.example_helper}
```

The following functions, classes and exceptions are provided by the toolkit.

***class* BAseController** Base class for building controllers.

***class* CkanCommand** Base class for building paster functions.

***class* DefaultDatasetForm** base class for IDatasetForm.

***exception* CkanVersionException** Exception raised if required ckan version is not available.

***exception* NotAuthorized** Exception raised when an action is not permitted by a user.

***exception* ObjectNotFound** Exception raised when an object cannot be found.

***exception* ValidationError** Exception raised when supplied data is invalid. it contains details of the error that occurred.

***exception* UnknownConverter** Exception raised when a converter cannot be found.

***exception* UnknownValidator** Exception raised when a validator cannot be found.

_ (*value*) Mark a string for translation. Returns the localized unicode string of value.

Mark a string to be localized as follows:

```
    _('This should be in lots of languages')
```

add_public_directory (*config, relative_path*) Function to aid adding extra public paths to the config. The path is relative to the file calling this function.

add_template_directory (*config, relative_path*) Function to aid adding extra template paths to the config. The path is relative to the file calling this function.

asbool (*obj*) part of paste.deploy.converters: convert strings like yes, no, true, false, 0, 1 to boolean

asint (*obj*) part of paste.deploy.converters: convert stings to integers

aslist (*obj, sep=None, strip=True*) part of paste.deploy.converters: convert string objects to a list

check_access (*action, context, data_dict=None*) check that the named action with the included context and optional data dict is allowed raises NotAuthorized if the action is not permitted or True.

check_ckan_version (*min_version=None, max_version=None*) Check that the ckan version is correct for the plugin.

get_action (*action*) Get the requested action function.

get_converter (*coverter*) Get the requested converter function.

get_validator (*validator*) Get the requested validator function.

literal (*html*) Treat the html as a literal that will not get escaped.

render (*template_name, data=None*) Main template render function.

render_snippet (*template_name, data=None*) helper for the render_snippet function similar to the render function.

render_text (*template_name, data=None*) Render genshi text template.

requires_ckan_version (*min_version, max_version=None*) Check that the ckan version is correct for the plugin.

request object This is the http request and contains the environ, cookies etc.

response object This is the http response.

abort (*error_code, error_message*) Aborts the current request.

redirect This causes a http redirect to be returned to the client.

url_for This function can be used to create urls.

side_effect_free decorator This marks action functions as accessible via the action api.

auth_sysadmins_check decorator This marks auth functions as needing to be run for sys admins. Usually sysadmins are automatically allowed to run actions etc.

get_or_bust (*data_dict, keys*) Try and get values from dictionary and if they are not there raise a ValidationError.

5.3 Example Extension

```
# Example Extension
# This extension adds a new template helper function `hello_world` when
# enabled templates can `{{ h.hello_world() }}` to add this html snippet.

import ckan.plugins as p

class HelloWorldPlugin(p.SingletonPlugin):

    p.implements(p.ITemplateHelpers)

    @staticmethod
    def hello_world():
        # This is our simple helper function.
        html = '<span>Hello World</span>'
        return p.toolkit.literal(html)

    def get_helpers(self):
        # This method is defined in the ITemplateHelpers interface and
        # is used to return a dict of named helper functions.
        return {'hello_world': hello_world}
```

5.4 Guidelines for writing extensions

- Use the plugins Plugins Toolkit.
- Extensions should use actions via `get_action()`. This function is available in the toolkit.
- No foreign key constraints into core as these cause problems.

Todo

Anything else?

5.5 Creating CKAN Extensions

All CKAN extensions must start with the name `ckanext-`. You can create your own CKAN extension like this (you must be in your CKAN pyenv):

```
(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.

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

```
(pyenv)$ cd ckanext-myextension
(pyenv)$ python setup.py develop
```

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`.

5.6 Testing

5.6.1 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/stats/tests --ckan
nosetests ckanext/stats/tests --ckan --with-pylons=test-core.ini
```

5.6.2 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 pylons import config
import paste.fixture
from ckan.config.middleware import make_app

class TestMyRoutesPlugin(TestCase):

    @classmethod
    def setup_class(cls):
        cls._original_config = config.copy()
        config['ckan.plugins'] = 'my_routes_plugin'
        wsgiapp = make_app(config['global_conf'], **config.local_conf)
        cls.app = paste.fixture.TestApp(wsgiapp)

    @classmethod
    def teardown_class(cls):
        config.clear()
        config.update(cls._original_config)
```

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

5.7 Plugin API Documentation

5.7.1 Core Plugin Reference

5.7.2 CKAN Interface Reference

CHAPTER 6

Theming

If you want more control over your CKAN site's layout and appearance than the options described in Getting Started give, you can further customize CKAN's appearance by developing a theme. CKAN's templates, HTML and CSS are all completely customizable by themes. This document will walk you through the process of developing a CKAN theme.

6.1 Create Custom Extension

This method is best for you want to customize the HTML templates of you CKAN instance. It's also more extensible and means you can make sure you keep your custom theme as seperate from CKAN core as possible.

Here follows the main topics you'll need in order to understand how to write a custom extension in order to customize your CKAN instance.

6.1.1 Customizing the HTML

The main templates within CKAN use the templating language Jinja2. Jinja2 has template inheritance which means that you don't have to re-write a whole template in order to change small elements within templates.

For more information on how to exactly change the HTML of your CKAN instance: please read the Templating > Templating within extensions documentation.

6.1.2 Including custom Stylesheets, JavaScript and images

Within CKAN we use a resource manager to handle the static resources that are required by any given template. In order to include a stylesheet or a JavaScript document you should tell the resource manager of its existence and then include it within your template.

For more information on how resources work within CKAN and how to add custom resources to your extension: please read the Resources > Resources within extensions documentation.

Note: The main CKAN theme is a heavily customized version of Bootstrap. However the core of Bootstrap is no different in CKAN and therefore people familiar with Bootstrap should feel right at home writing custom HTML and CSS for CKAN.

6.1.3 Customizing the JavaScript

Within CKAN core we have a concept of JavaScript modules which allow you to simply attach JavaScript to DOM elements via HTML5 data attributes.

For more information on what a JavaScript module is and how to build one: please read the Building a JavaScript Module documentation.

6.1.4 Customizing the CSS

To customize your CSS all you really need to know is how to add a stylesheet as a resource. Beyond that it's purely writing your own CSS and making sure it's included on the correct pages.

For more information on how CSS works in CKAN core: please read the Front End Documentation > Stylesheets documentation.

Note: In CKAN core we use LESS to pre-process our main CSS document. We do this to make the core CSS more maintainable (as well as to offer different basic colour styles on our default theme). It's not necessary that you do the same, but we'd recommend using something like it if you plan on customizing your CKAN instance heavily.

CHAPTER 7

The CKAN API

CKAN's **Action API** is a powerful, RPC-style API that exposes all of CKAN's core features to API clients. All of a CKAN website's core functionality (everything you can do with the web interface and more) can be used by external code that calls the CKAN API. For example, using the CKAN API your app can:

- Get JSON-formatted lists of a site's datasets, groups or other CKAN objects:

  http://demo.ckan.org/api/3/action/package_list

  http://demo.ckan.org/api/3/action/group_list

  http://demo.ckan.org/api/3/action/tag_list

- Get a full JSON representation of a dataset, resource or other object:

  http://demo.ckan.org/api/3/action/package_show?id=adur_district_spending

  http://demo.ckan.org/api/3/action/tag_show?id=gold

  http://demo.ckan.org/api/3/action/group_show?id=data-explorer

- Search for packages or resources matching a query:

  http://demo.ckan.org/api/3/action/package_search?q=spending

  http://demo.ckan.org/api/3/action/resource_search?query=name:District%20Names

- Create, update and delete datasets, resources and other objects
- Get an activity stream of recently changed datasets on a site:

  http://demo.ckan.org/api/3/action/recently_changed_packages_activity_list

Note: CKAN's FileStore and DataStore have their own APIs, see:

- FileStore and File Uploads
- DataStore Extension

Note: For documentation of CKAN's legacy API's, see Legacy APIs.

7.1 Legacy APIs

Warning: The legacy APIs documented in this section are provided for backwards-compatibility, but support for new CKAN features will not be added to these APIs.

7.1.1 API Versions

There are two versions of the legacy APIs. When the API returns a reference to an object, version 1 of the API will return the name of the object (e.g. `"river-pollution"`), whereas version 2 will return the ID of the object (e.g. `"a3dd8f64-9078-4f04-845c-e3f047125028"`). Tag objects are an exception, tag names are immutable so tags are always referred to with their name.

You can specify which version of the API to use in the URL. For example, opening this URL in your web browser will list demo.ckan.org's datasets using API version 1:

http://demo.ckan.org/api/1/rest/dataset

Opening this URL calls the same function using API version 2:

http://demo.ckan.org/api/2/rest/dataset

If no version number is given in the URL then the API defaults to version 1, so this URL will list the site's datasets using API version 1:

http://demo.ckan.org/api/rest/dataset

Dataset names can change, so to reliably refer to the same dataset over time, you will want to use the dataset's ID and therefore use API v2. Alternatively, many people prefer to deal with Names, so API v1 suits them.

When posting parameters with your API requests, you can refer to objects by either their name or ID, interchangeably.

7.1.2 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
meta-data_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)

7.1.3 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.

Note: Only public datasets can be accessed via the legacy search API, regardless of the provided authorization. If you need to access private datasets via the API you will need to use the *package_search* method of the The CKAN

API.

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.

7.1.4 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 (annakarenina)", "name": "a-novel-by-tolstoy-annakarenina"}]}}
```

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"
```

7.1.5 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

7.2 Making an API Request

To call the CKAN API, post a JSON dictionary in an HTTP POST request to one of CKAN's API URLs. The parameters for the API function should be given in the JSON dictionary. CKAN will also return its response in a JSON dictionary.

One way to post a JSON dictionary to a URL is using the command-line HTTP client HTTPie. For example, to get a list of the names of all the datasets in the `data-explorer` group on demo.ckan.org, install HTTPie and then call the `group_list` API function by running this command in a terminal:

```
http http://demo.ckan.org/api/3/action/group_list id=data-explorer
```

The response from CKAN will look like this:

```
{
    "help": "...",
    "result": [
        "data-explorer",
        "department-of-ricky",
        "geo-examples",
        "geothermal-data",
        "reykjavik",
        "skeenawild-conservation-trust"
    ],
    "success": true
}
```

The response is a JSON dictionary with three keys:

1. `"sucess"`: `true` or `false`.

   The API aims to always return `200 OK` as the status code of its HTTP response, whether there were errors with the request or not, so it's important to always check the value of the `"success"` key in the response dictionary and (if success is `False`) check the value of the `"error"` key.

Note: If there are major formatting problems with a request to the API, CKAN may still return an HTTP response with a `409`, `400` or `500` status code (in increasing order of severity). In future CKAN versions we intend to remove these responses, and instead send a `200 OK` response and use the `"success"` and `"error"` items.

---

2. `"result"`: the returned result from the function you called. The type and value of the result depend on which function you called. In the case of the `group_list` function it's a list of strings, the names of all the datasets that belong to the group.

   If there was an error responding to your request, the dictionary will contain an `"error"` key with details of the error instead of the `"result"` key. A response dictionary containing an error will look like this:

```
    {
        "help": "Creates a package",
        "success": false,
        "error": {
            "message": "Access denied",
            "__type": "Authorization Error"
            }
     }
```

3. `"help"`: the documentation string for the function you called.

The same HTTP request can be made using Python's standard `urllib2` module, with this Python code:

```
#!/usr/bin/env python
import urllib2
import urllib
import json
import pprint

# Use the json module to dump a dictionary to a string for posting.
data_string = urllib.quote(json.dumps({'id': 'data-explorer'}))

# Make the HTTP request.
response = urllib2.urlopen('http://demo.ckan.org/api/3/action/group_list',
        data_string)
assert response.code == 200

# Use the json module to load CKAN's response into a dictionary.
response_dict = json.loads(response.read())

# Check the contents of the response.
assert response_dict['success'] is True
result = response_dict['result']
pprint.pprint(result)
```

7.3 Example: Importing Datasets with the CKAN API

You can add datasets using CKAN's web interface, but when importing many datasets it's usually more efficient to automate the process in some way. In this example, we'll show you how to use the CKAN API to write a Python script to import datasets into CKAN.

Todo

Make this script more interesting (eg. read data from a CSV file), and all put the script in a .py file somewhere with tests and import it here.

```
#!/usr/bin/env python
import urllib2
import urllib
import json
import pprint

# Put the details of the dataset we're going to create into a dict.
dataset_dict = {
    'name': 'my_dataset_name',
    'notes': 'A long description of my dataset',
}

# Use the json module to dump the dictionary to a string for posting.
data_string = urllib.quote(json.dumps(dataset_dict))

# We'll use the package_create function to create a new dataset.
request = urllib2.Request(
    'http://www.my_ckan_site.com/api/action/package_create')

# Creating a dataset requires an authorization header.
# Replace *** with your API key, from your user account on the CKAN site
# that you're creating the dataset on.
```

```
request.add_header('Authorization', '***')

# Make the HTTP request.
response = urllib2.urlopen(request, data_string)
assert response.code == 200

# Use the json module to load CKAN's response into a dictionary.
response_dict = json.loads(response.read())
assert response_dict['success'] is True

# package_create returns the created package as its result.
created_package = response_dict['result']
pprint.pprint(created_package)
```

7.4 API Versions

The CKAN APIs are versioned. If you make a request to an API URL without a version number, CKAN will choose the latest version of the API:

```
http://demo.ckan.org/api/action/package_list
```

Alternatively, you can specify the desired API version number in the URL that you request:

```
http://demo.ckan.org/api/3/action/package_list
```

Version 3 is currently the only version of the Action API.

We recommend that you specify the API number in your requests, because this ensures that your API client will work across different sites running different version of CKAN (and will keep working on the same sites, when those sites upgrade to new versions of CKAN). Because the latest version of the API may change when a site is upgraded to a new version of CKAN, or may differ on different sites running different versions of CKAN, the result of an API request that doesn't specify the API version number cannot be relied on.

7.5 Authentication and API Keys

Some API functions require authorization. The API uses the same authorization functions and configuration as the web interface, so if a user is authorized to do something in the web interface they'll be authorized to do it via the API as well.

When calling an API function that requires authorization, you must authenticate yourself by providing your API key with your HTTP request. To find your API key, login to the CKAN site using its web interface and visit your user profile page.

To provide your API key in an HTTP request, include it in either an `Authorization` or `X-CKAN-API-Key` header. (The name of the HTTP header can be configured with the `apikey_header_name` option in your CKAN configuration file.)

For example, to ask whether or not you're currently following the user `markw` on demo.ckan.org using HTTPie, run this command:

```
http http://demo.ckan.org/api/3/action/am_following_user id=markw Authorization:XXX
```

(Replacing `XXX` with your API key.)

Or, to get the list of activities from your user dashboard on demo.ckan.org, run this Python code:

```
request = urllib2.Request('http://demo.ckan.org/api/3/action/dashboard_activity_list')
request.add_header('Authorization', 'XXX')
response_dict = json.loads(urllib2.urlopen(request, '{}').read())
```

7.6 GET-able API Functions

Functions defined in *ckan.logic.action.get* can also be called with an HTTP GET request. For example, to get the list of datasets (packages) from demo.ckan.org, open this URL in your browser:

http://demo.ckan.org/api/3/action/package_list

Or, to search for datasets (packages) matching the search query `spending`, on demo.ckan.org, open this URL in your browser:

http://demo.ckan.org/api/3/action/package_search?q=spending

Tip: Browser plugins like JSONView for Firefox or Chrome will format and color CKAN's JSON response nicely in your browser.

The search query is given as a URL parameter `?q=spending`. Multiple URL parameters can be appended, separated by `&` characters, for example to get only the first 10 matching datasets open this URL:

http://demo.ckan.org/api/3/action/package_search?q=spending&rows=10

When an action requires a list of strings as the value of a parameter, the value can be sent by giving the parameter multiple times in the URL:

http://demo.ckan.org/api/3/action/term_translation_show?terms=russian&terms=romantic%20novel

7.7 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. For example:

http://demo.ckan.org/api/3/action/package_show?id=adur_district_spending&callback=myfunction

---

Todo

This doesn't work with all functions.

7.8 Action API Reference

7.8.1 ckan.logic.action.get

7.8.2 ckan.logic.action.create

7.8.3 ckan.logic.action.update

7.8.4 ckan.logic.action.delete

CHAPTER 8

Contributing to CKAN

CKAN is free open source software and code contributions are welcome, whether they're bug reports, source code, documentation or translations. The sections below will walk you through our processes for making different kinds of contributions to CKAN.

8.1 Reporting Issues

If you've found a bug in CKAN, open a new issue on CKAN's GitHub Issues (try searching first to see if there's already an issue for your bug).

8.2 Translating CKAN

For contributing translations to CKAN, see Translating CKAN.

8.2.1 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 check the *ckan.root_path* config option.

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.

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*.

8.3 Coding Standards

When writing code for CKAN, try to respect our coding standards:

8.3.1 CKAN Coding Standards

This section documents our CKAN-specific coding standards, which are guidelines for writing code that is consistent with the intended design and architecture of CKAN.

For more general coding standards, see also:

- Python Coding Standards
- HTML Coding Standards
- CSS Coding Standards
- Javascript Coding Standards

Encapsulate SQLAlchemy in `ckan.model`

Ideally SQLAlchemy should only be used within `ckan.model` and not from other packages such as `ckan.logic`. For example instead of using an SQLAlchemy query from the logic package to retrieve a particular user from the database, we add a `get()` method to `ckan.model.user.User`:

```
@classmethod
def get(cls, user_id):
    query = ...
    .
    .
    .
    return query.first()
```

Now we can call this method from the logic package.

Database Migrations

When changes are made to the model classes in `ckan.model` that alter CKAN's database schema, a migration script has to be added to migrate old CKAN databases to the new database schema when they upgrade their copies of CKAN. See Creating a new migration script.

Creating a new migration script

When changes are made to the model classes in `ckan.model` that alter CKAN's database schema, a migration script has to be added to migrate old CKAN databases to the new database schema when they upgrade their copies of CKAN. These migration scripts are kept in `ckan.migration.versions`.

When you upgrade a CKAN instance, as part of the upgrade process you run any necessary migration scripts with the `paster db upgrade` command:

```
paster --plugin=ckan db upgrade --config={.ini file}
```

A migration script should be checked into CKAN at the same time as the model changes it is related to. Before pushing the changes, ensure the tests pass when running against the migrated model, which requires the `--ckan-migration` setting.

To create a new migration script, create a python file in `ckan/migration/versions/` and name it with a prefix numbered one higher than the previous one and some words describing the change.

You need to use the special engine provided by the SqlAlchemy Migrate. Here is the standard header for your migrate script:

```
from sqlalchemy import *
from migrate import *
```

The migration operations go in the upgrade function:

```
def upgrade(migrate_engine):
  metadata = MetaData()
  metadata.bind = migrate_engine
```

The following process should be followed when doing a migration. This process is here to make the process easier and to validate if any mistakes have been made:

1. Get a dump of the database schema before you add your new migrate scripts.

```
    paster --plugin=ckan db clean --config={.ini file}
    paster --plugin=ckan db upgrade --config={.ini file}
    pg_dump -h host -s -f old.sql dbname
```

2. Get a dump of the database as you have specified it in the model.

```
    paster --plugin=ckan db clean --config={.ini file}

    #this makes the database as defined in the model
    paster --plugin=ckan db create-from-model -config={.ini file}
    pg_dump -h host -s -f new.sql dbname
```

3. Get agpdiff (apt-get it). It produces sql it thinks that you need to run on the database in order to get it to the updated schema.

```
    apgdiff old.sql new.sql > upgrade.diff
```

(or if you don't want to install java use http://apgdiff.startnet.biz/diff_online.php)

4. The upgrade.diff file created will have all the changes needed in sql. Delete the drop index lines as they are not created in the model.

5. Put the resulting sql in your migrate script, e.g.

```
    migrate_engine.execute('''update table .........; update table ....''')
```

6. Do a dump again, then a diff again to see if the the only thing left are drop index statements.

7. run nosetests with `--ckan-migration` flag.

It's that simple. Well almost.

- If you are doing any table/field renaming adding that to your new migrate script first and use this as a base for your diff (i.e add a migrate script with these renaming before 1). This way the resulting sql won't try to drop and recreate the field/table!
- It sometimes drops the foreign key constraints in the wrong order causing an error so you may need to rearrange the order in the resulting upgrade.diff.
- If you need to do any data transfer in the migrations then do it between the dropping of the constraints and adding of new ones.
- May need to add some tests if you are doing data migrations.

An example of a script doing it this way is `034_resource_group_table.py`. This script copies the definitions of the original tables in order to do the renaming the tables/fields.

In order to do some basic data migration testing extra assertions should be added to the migration script. Examples of this can also be found in `034_resource_group_table.py` for example.

This statement is run at the top of the migration script to get the count of rows:

```
package_count = migrate_engine.execute('''select count(*) from package''').first()[0]
```

And the following is run after to make sure that row count is the same:

```
resource_group_after = migrate_engine.execute('''select count(*) from resource_group''').first()[0]
assert resource_group_after == package_count
```

Always go through the Action Functions

Whenever some code, for example in `ckan.lib` or `ckan.controllers`, wants to get, create, update or delete an object from CKAN's model it should do so by calling a function from the `ckan.logic.action` package, and *not* by accessing `ckan.model` directly.

Action Functions are Exposed in the API

The functions in `ckan.logic.action` are exposed to the world as the The CKAN API. The API URL for an action function is automatically generated from the function name, for example `ckan.logic.action.create.package_create()` is exposed at `/api/action/package_create`. See Steve Yegge's Google platforms rant for some interesting discussion about APIs.

All publicly visible functions in the `ckan.logic.action.{create,delete,get,update}` namespaces will be exposed through the The CKAN API. **This includes functions imported** by those modules, **as well as any helper functions** defined within those modules. To prevent inadvertent exposure of non-action functions through the action api, care should be taken to:

1. Import modules correctly (see *Imports*). For example:

```
    import ckan.lib.search as search

    search.query_for(...)
```

2. Hide any locally defined helper functions:

```
    def _a_useful_helper_function(x, y, z):
       '''This function is not exposed because it is marked as private```
       return x+y+z
```

3. Bring imported convenience functions into the module namespace as private members:

```
    _get_or_bust = logic.get_or_bust
```

Use `get_action()`

Don't call `logic.action` functions directly, instead use `get_action()`. This allows plugins to override action functions using the `IActions` plugin interface. For example:

```
ckan.logic.get_action('group_activity_list_html')(...)
```

Instead of

```
ckan.logic.action.get.group_activity_list_html(...)
```

Auth Functions and `check_access()`

Each action function defined in `ckan.logic.action` should use its own corresponding auth function defined in `ckan.logic.auth`. Instead of calling its auth function directly, an action function should go through `ckan.logic.check_access` (which is aliased `_check_access` in the action modules) because this allows plugins to override auth functions using the `IAuthFunctions` plugin interface. For example:

```
def package_show(context, data_dict):
    _check_access('package_show', context, data_dict)
```

`check_access` will raise an exception if the user is not authorized, which the action function should not catch. When this happens the user will be shown an authorization error in their browser (or will receive one in their response from the API).

`logic.get_or_bust()`

The `data_dict` parameter of logic action functions may be user provided, so required files may be invalid or absent. Naive Code like:

```
id = data_dict['id']
```

may raise a `KeyError` and cause CKAN to crash with a 500 Server Error and no message to explain what went wrong. Instead do:

```
id = _get_or_bust(data_dict, "id")
```

which will raise `ValidationError` if `"id"` is not in `data_dict`. The `ValidationError` will be caught and the user will get a 400 Bad Request response and an error message explaining the problem.

Validation and `ckan.logic.schema`

Logic action functions can use schema defined in `ckan.logic.schema` to validate the contents of the `data_dict` parameters that users pass to them.

An action function should first check for a custom schema provided in the context, and failing that should retrieve its default schema directly, and then call `_validate()` to validate and convert the data. For example, here is the validation code from the `user_create()` action function:

```
schema = context.get('schema') or ckan.logic.schema.default_user_schema()
session = context['session']
validated_data_dict, errors = _validate(data_dict, schema, context)
if errors:
    session.rollback()
    raise ValidationError(errors)
```

Controller & Template Helper Functions

`ckan.lib.helpers` contains helper functions that can be used from `ckan.controllers` or from templates. When developing for ckan core, only use the helper functions found in `ckan.lib.helpers.__allowed_functions__`.

Testing

- Functional tests which test the behaviour of the web user interface, and the APIs should be placed within `ckan/tests/functional`. These tests can be a lot slower to run that unit tests which don't access the database or solr. So try to bear that in mind, and attempt to cover just what is neccessary, leaving what can be tested via unit-testing in unit-tests.
- `nose.tools.assert_in` and `nose.tools.assert_not_in` are only available in Python>=2.7. So import them from `ckan.tests`, which will provide alternatives if they're not available.
- the mock library can be used to create and interrogate mock objects.

See Testing CKAN for further information on testing in CKAN.

Writing Extensions

Please see Writing Extensions for information about writing ckan extensions, including details on the API available to extensions.

Deprecation

- Anything that may be used by extensions (see Writing Extensions) needs to maintain backward compatibility at call-site. ie - template helper functions and functions defined in the plugins toolkit.
- The length of time of deprecation is evaluated on a function-by-function basis. At minimum, a function should be marked as deprecated during a point release.
- To mark a helper function, use the `deprecated` decorator found in `ckan.lib.maintain` eg:

```
    @deprecated()
    def facet_items(*args, **kwargs):
        """
        DEPRECATED: Use the new facet data structure, and `unselected_facet_items()`
        """
        # rest of function definition.
```

8.3.2 Python Coding Standards

For Python code style follow PEP 8 plus the guidelines below.

Some good links about Python code style:

- Python Coding Standards from Yahoo
- Google Python Style Guide

Commit Formatting Cleanups on master

Clean up formatting and PEP 8 issues on master, not on a feature branch. Unless of course you're changing that piece of code anyway. This will help avoid spurious merge conflicts, and aid in reading pull requests.

Use Single Quotes

Use single-quotes for string literals, e.g. `'my-identifier'`, *but* use double-quotes for strings that are likely to contain single-quote characters as part of the string itself (such as error messages, or any strings containing natural language), e.g. `"You've got an error!"`.

Single-quotes are easier to read and to type, but if a string contains single-quote characters then double-quotes are better than escaping the single-quote characters or wrapping the string in double single-quotes.

We also use triple single-quotes for docstrings, see *Docstrings*.

Imports

- Don't use `from module import *` or `from module import name`. Instead just `import module` and then access names with `module.name`. See Idioms and Anti-Idioms in Python.

  You can make long module names more concise by aliasing them:

```
import foo.bar.baz as baz
```

  and then access it with `baz` in your code.

- Make all imports at the start of the file, after the module docstring. Imports should be grouped in the following order:

  1. Standard library imports
  2. Third-party imports
  3. CKAN imports

Logging

- Keep log messages short.
- Don't include object representations in the log message. It *is* useful to include a domain model identifier where appropriate.
- Choose an appropriate log-level (DEBUG, INFO, ERROR, WARNING or CRITICAL, see Python's Logging HOWTO).

String Formatting

Don't use the old *%s* style string formatting, e.g. `"i am a %s" % sub`. This kind of string formatting is not helpful for internationalization and is going away in Python 3.

Use the new .format() method instead, and give meaningful names to each replacement field, for example:

```
_(' ... {foo} ... {bar} ...').format(foo='foo-value', bar='bar-value')
```

Docstrings

We want CKAN's docstrings to be clear and easy to read for programmers who are smart and competent but who may not know a lot of CKAN technical jargon and whose first language may not be English. We also want it to be easy to maintain the docstrings and keep them up to date with the actual behaviour of the code as it changes over time. So:

- All modules and all public functions, classes and methods exported by a module should normally have docstrings (see PEP 257).
- Keep docstrings short, describe only what's necessary and no more,
- Keep docstrings simple: use plain, concise English.
- Try to avoid repetition.

PEP 257 (Docstring Conventions)

Generally, follow PEP 257 for docstrings. We'll only describe the ways that CKAN differs from or extends PEP 257 below.

CKAN docstrings deviate from PEP 257 in a couple of ways:

- We use `'''triple single quotes'''` around docstrings, not `"""triple double quotes"""` (put triple single quotes around one-line docstrings as well as multi-line ones, it makes them easier to expand later)
- We use Sphinx directives for documenting parameters, exceptions and return values (see below)

Sphinx Field Lists

Use Sphinx field lists for documenting the parameters, exceptions and returns of functions:

- Use `:param` and `:type` to describe each parameter
- Use `:returns` and `:rtype` to describe each return
- Use `:raises` to describe each exception raised

Example of a short docstring:

```
@property
def packages(self):
    '''Return a list of all packages that have this tag, sorted by name.

    :rtype: list of ckan.model.package.Package objects

    '''
```

Example of a longer docstring:

```
@classmethod
def search_by_name(cls, search_term, vocab_id_or_name=None):
    '''Return all tags whose names contain a given string.

    By default only free tags (tags which do not belong to any vocabulary)
    are returned. If the optional argument ``vocab_id_or_name`` is given
    then only tags from that vocabulary are returned.

    :param search_term: the string to search for in the tag names
    :type search_term: string
    :param vocab_id_or_name: the id or name of the vocabulary to look in
        (optional, default: None)
    :type vocab_id_or_name: string

    :returns: a list of tags that match the search term
    :rtype: list of ckan.model.tag.Tag objects

    '''
```

The phrases that follow `:param foo:`, `:type foo:`, or `:returns:` should not start with capital letters or end with full stops. These should be short phrases and not full sentences. If more detail is required put it in the function description instead.

Indicate optional arguments by ending their descriptions with `(optional)` in brackets. Where relevant also indicate the default value: `(optional, default: 5)`.

You can also use a little inline reStructuredText markup in docstrings, e.g. `*stars for emphasis*` or ``` ``double-backticks for literal text`` ```

Action API Docstrings

Docstrings from CKAN's action API are processed with autodoc and included in the API chapter of CKAN's documentation. The intended audience of these docstrings is users of the CKAN API and not (just) CKAN core developers.

In the Python source each API function has the same two arguments (`context` and `data_dict`), but the docstrings should document the keys that the functions read from `data_dict` and not `context` and `data_dict` themselves, as this is what the user has to POST in the JSON dict when calling the API.

Where practical, it's helpful to give examples of param and return values in API docstrings.

CKAN datasets used to be called packages and the old name still appears in the source, e.g. in function names like `package_list()`. When documenting functions like this write dataset not package, but the first time you do this put package after it in brackets to avoid any confusion, e.g.

```
def package_show(context, data_dict):
    '''Return the metadata of a dataset (package) and its resources.
```

Example of a ckan.logic.action API docstring:

```
def vocabulary_create(context, data_dict):
    '''Create a new tag vocabulary.

    You must be a sysadmin to create vocabularies.

    :param name: the name of the new vocabulary, e.g. ``'Genre'``
    :type name: string
    :param tags: the new tags to add to the new vocabulary, for the format of
        tag dictionaries see ``tag_create()``
    :type tags: list of tag dictionaries
```

```
    :returns: the newly-created vocabulary
    :rtype: dictionary

    '''
```

Some Helpful Tools for Python Code Quality

There are various tools that can help you to check your Python code for PEP8 conformance and general code quality. We recommend using them.

- pep8 checks your Python code against some of the style conventions in PEP 8. As mentioned above, only perform style clean-ups on master to help avoid spurious merge conflicts.
- pylint analyzes Python source code looking for bugs and signs of poor quality.
- pyflakes also analyzes Python programs to detect errors.
- flake8 combines both pep8 and pyflakes into a single tool.
- Syntastic is a Vim plugin with support for flake8, pyflakes and pylint.

8.3.3 HTML Coding Standards

Formatting

All HTML documents must use **two spaces** for indentation and there should be no trailing whitespace. XHTML syntax must be used (this is more a Genshi requirement) and all attributes must use double quotes around attributes.

```
<!-- XHTML boolean attributes must still have values and self closing tags must have a closing / -->
<video autoplay="autoplay" poster="poster_image.jpg">
  <source src="foo.ogg" type="video/ogg" />
</video>
```

HTML5 elements should be used where appropriate reserving `<div>` and `<span>` elements for situations where there is no semantic value (such as wrapping elements to provide styling hooks).

Doctype and layout

All documents must be using the HTML5 doctype and the `<html>` element should have a `"lang"` attribute. The `<head>` should also at a minimum include `"viewport"` and `"charset"` meta tags.

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Example Site</title>
  </head>
  <body></body>
</html>
```

Forms

Form fields must always include a `<label>` element with a `"for"` attribute matching the `"id"` on the input. This helps accessibility by focusing the input when the label is clicked, it also helps screen readers match labels to their respective inputs.

```
<label for="field-email">email</label>
<input type="email" id="field-email" name="email" value="" />
```

Each `<input>` should have an `"id"` that is unique to the page. It does not have to match the `"name"` attribute.

Forms should take advantage of the new HTML5 input types where they make sense to do so, placeholder attributes should also be included where relevant. Including these can provided enhancements in browsers that support them such as tailored inputs and keyboards.

```
<div>
  <label for="field-email">Email</label>
  <input type="email" id="field-email" name="email" value="name@example.com" />
</div>
<div>
  <label for="field-phone">Phone</label>
  <input type="phone" id="field-phone" name="phone" value="" placeholder="+44 077 12345 678" />
</div>
<div>
  <label for="field-url">Homepage</label>
  <input type="url" id="field-url" name="url" value="" placeholder="http://example.com" />
</div>
```

Wufoo provides an excellent reference for these attributes.

Including meta data

Classes should ideally only be used as styling hooks. If you need to include additional data in the html document, for example to pass data to JavaScript, then the HTML5 `data-` attributes should be used.

```
<a class="btn" data-format="csv">Download CSV</a>
```

These can then be accessed easily via jQuery using the `.data()` method.

```
jQuery('.btn').data('format'); //=> "csv"

// Get the contents of all data attributes.
jQuery('.btn').data(); => {format: "csv"}
```

One thing to note is that the JavaScript API for datasets will convert all attribute names into camelCase. So `"data-file-format"` will become `fileFormat`.

For example:

```
<a class="btn" data-file-format="csv">Download CSV</a>
```

Will become:

```
jQuery('.btn').data('fileFormat'); //=> "csv"
jQuery('.btn').data(); => {fileFormat: "csv"}
```

Targeting Internet Explorer

Targeting lower versions of Internet Explorer (IE), those below version 9, should be handled by the stylesheets. Small fixes should be provided inline using the `.ie` specific class names. Larger fixes may require a separate stylesheet but try to avoid this if at all possible.

Adding IE specific classes:

```
<!doctype html>
<!--[if lt IE 7]> <html lang="en" class="ie ie6"> <![endif]-->
<!--[if IE 7]>    <html lang="en" class="ie ie7"> <![endif]-->
<!--[if IE 8]>    <html lang="en" class="ie ie8"> <![endif]-->
<!--[if gt IE 8]><!--> <html lang="en"> <!--<![endif]-->
```

Note: Only add lines for classes that are actually being used.

These can then be used within the CSS:

```
.clear:before,
.clear:after {
    content: "";
    display: table;
}

.clear:after {
    clear: both;
}

.ie7 .clear {
    zoom: 1; /* For IE 6/7 (trigger hasLayout) */
}
```

i18n

Don't include line breaks within `<p>` blocks. ie do this:

```
<p>Blah foo blah</p>
<p>New paragraph, blah</p>
```

And **not**:

```
<p>Blah foo blah
   New paragraph, blah</p>
```

8.3.4 CSS Coding Standards

Note: For CKAN 2.0 we use LESS as a pre-processor for our core CSS. View Front-end Documentation for more information on this subject.

Formatting

All CSS documents must use **two spaces** for indentation and files should have no trailing whitespace. Other formatting rules:

- Use soft-tabs with a two space indent.
- Use double quotes.
- Use shorthand notation where possible.
- Put spaces after `:` in property declarations.
- Put spaces before `{` in rule declarations.
- Use hex color codes `#000` unless using `rgba()`.
- Always provide fallback properties for older browsers.
- Use one line per property declaration.
- Always follow a rule with one line of whitespace.
- Always quote `url()` and `@import()` contents.
- Do not indent blocks.

For example:

```
.media {
  overflow: hidden;
  color: #fff;
  background-color: #000; /* Fallback value */
  background-image: linear-gradient(black, grey);
}

.media .img {
  float: left;
  border: 1px solid #ccc;
}

.media .img img {
  display: block;
}

.media .content {
  background: #fff url("../images/media-background.png") no-repeat;
}
```

Naming

All ids, classes and attributes must be lowercase with hyphens used for separation.

```
/* GOOD */
.dataset-list {}

/* BAD */
.datasetlist {}
.datasetList {}
.dataset_list {}
```

Comments

Comments should be used liberally to explain anything that may be unclear at first glance, especially IE workarounds or hacks.

```
.prose p {
  font-size: 1.1666em /* 14px / 12px */;
}

.ie7 .search-form {
  /*
    Force the item to have layout in IE7 by setting display to block.
    See: http://reference.sitepoint.com/css/haslayout
  */
  display: inline-block;
}
```

Modularity & Specificity

Try keep all selectors loosely grouped into modules where possible and avoid having too many selectors in one declaration to make them easy to override.

```
/* Avoid */
ul#dataset-list {}
ul#dataset-list li {}
ul#dataset-list li p a.download {}
```

Instead here we would create a dataset "module" and styling the item outside of the container allows you to use it on it's own e.g. on a dataset page:

```
.dataset-list {}
.dataset-list-item {}
.dataset-list-item .download {}
```

In the same vein use classes make the styles more robust, especially where the HTML may change. For example when styling social links:

```
<ul class="social">
  <li><a href="">Twitter</a></li>
  <li><a href="">Facebook</a></li>
  <li><a href="">LinkedIn</a></li>
</ul>
```

You may use pseudo selectors to keep the HTML clean:

```
.social li:nth-child(1) a {
  background-image: url(twitter.png);
}

.social li:nth-child(2) a {
  background-image: url(facebook.png);
}

.social li:nth-child(3) a {
  background-image: url(linked-in.png);
}
```

However this will break any time the HTML changes for example if an item is added or removed. Instead we can use class names to ensure the icons always match the elements (Also you'd probably sprite the image :).

```
.social .twitter {
  background-image: url(twitter.png);
}
```

```
.social .facebook {
  background-image: url(facebook.png);
}

.social .linked-in {
  background-image: url(linked-in.png);
}
```

Avoid using tag names in selectors as this prevents re-use in other contexts.

```
/* Cannot use this class on an <ol> or <div> element */
ul.dataset-item {}
```

Also ids should not be used in selectors as it makes it far too difficult to override later in the cascade.

```
/* Cannot override this button style without including an id */
.btn#download {}
```

8.3.5 Javascript Coding Standards

Formatting

All JavaScript documents must use **two spaces** for indentation. This is contrary to the OKFN Coding Standards but matches what's in use in the current code base.

Coding style must follow the idiomatic.js style but with the following exceptions.

Note: Idiomatic is heavily based upon Douglas Crockford's style guide which is recommended by the OKFN Coding Standards.

White Space

Two spaces must be used for indentation at all times. Unlike in idiomatic whitespace must not be used _inside_ parentheses between the parentheses and their Contents.

```
// BAD: Too much whitespace.
function getUrl( full ) {
  var url = '/styleguide/javascript/';
  if ( full ) {
    url = 'http://okfn.github.com/ckan' + url;
  }
  return url;
}

// GOOD:
function getUrl(full) {
  var url = '/styleguide/javascript/';
  if (full) {
    url = 'http://okfn.github.com/ckan' + url;
  }
  return url;
}
```

Note: See section 2.D.1.1 of idiomatic for more examples of this syntax.

Quotes

Single quotes should be used everywhere unless writing JSON or the string contains them. This makes it easier to create strings containing HTML.

```
jQuery('<div id="my-div" />').appendTo('body');
```

Object properties need not be quoted unless required by the interpreter.

```
var object = {
  name: 'bill',
  'class': 'user-name'
};
```

Variable declarations

One `var` statement must be used per variable assignment. These must be declared at the top of the function in which they are being used.

```
// GOOD:
var good = 'string';
var alsoGood = 'another';

// GOOD:
var good = 'string';
var okay = [
  'hmm', 'a bit', 'better'
];

// BAD:
var good = 'string',
    iffy = [
  'hmm', 'not', 'great'
];
```

Declare variables at the top of the function in which they are first used. This avoids issues with variable hoisting. If a variable is not assigned a value until later in the function then it it okay to define more than one per statement.

```
// BAD: contrived example.
function lowercaseNames(names) {
  var names = [];

  for (var index = 0, length = names.length; index < length; index += 1) {
    var name = names[index];
    names.push(name.toLowerCase());
  }

  var sorted = names.sort();
  return sorted;
}

// GOOD:
function lowercaseNames(names) {
  var names = [];
  var index, sorted, name;

  for (index = 0, length = names.length; index < length; index += 1) {
```

```
    name = names[index];
    names.push(name.toLowerCase());
  }

  sorted = names.sort();
  return sorted;
}
```

Naming

All properties, functions and methods must use lowercase camelCase:

```
var myUsername = 'bill';
var methods = {
  getSomething: function () {}
};
```

Constructor functions must use uppercase CamelCase:

```
function DatasetSearchView() {
}
```

Constants must be uppercase with spaces delimited by underscores:

```
var env = {
  PRODUCTION:  'production',
  DEVELOPMENT: 'development',
  TESTING:     'testing'
};
```

Event handlers and callback functions should be prefixed with "on":

```
function onDownloadClick(event) {}

jQuery('.download').click(onDownloadClick);
```

Boolean variables or methods returning boolean functions should prefix the variable name with "is":

```
function isAdmin() {}

var canEdit = isUser() && isAdmin();
```

Note: Alternatives are "has", "can" and "should" if they make more sense

Private methods should be prefixed with an underscore:

```
View.extend({
  "click": "_onClick",
  _onClick: function (event) {
  }
});
```

Functions should be declared as named functions rather than assigning an anonymous function to a variable.

```
// GOOD:
function getName() {
}
```

```
// BAD:
var getName = function () {
};
```

Named functions are generally easier to debug as they appear named in the debugger.

Comments

Comments should be used to explain anything that may be unclear when you return to it in six months time. Single line comments should be used for all inline comments that do not form part of the documentation.

```
// Export the function to either the exports or global object depending
// on the current environment. This can be either an AMD module, CommonJS
// module or a browser.
if (typeof module.define === 'function' && module.define.amd) {
  module.define('broadcast', function () {
    return Broadcast;
  });
} else if (module.exports) {
  module.exports = Broadcast;
} else {
  module.Broadcast = Broadcast;
}
```

JSHint

All JavaScript should pass JSHint before being committed. This can be installed using `npm` (which is bundled with node) by running:

```
$ npm -g install jshint
```

Each project should include a jshint.json file with appropriate configuration options for the tool. Most text editors can also be configured to read from this file.

Documentation

For documentation we use a simple markup format to document all methods. The documentation should provide enough information to show the reader what the method does, arguments it accepts and a general example of usage. Also for API's and third party libraries, providing links to external documentation is encouraged.

The formatting is as follows:

```
/* My method description. Should describe what the method does and where
 * it should be used.
 *
 * param1 - The method params, one per line (default: null)
 * param2 - A default can be provided in brackets at the end.
 *
 * Example
 *
 *   // Indented two spaces. Should give a common example of use.
 *   client.getTemplate('index.html', {limit: 1}, function (html) {
 *     module.el.html(html);
 *   });
 *
```

```
 * Returns describes what the object returns.
 */
```

For example:

```
/* Loads an HTML template from the CKAN snippet API endpoint. Template
 * variables can be passed through the API using the params object.
 *
 * Optional success and error callbacks can be provided or these can
 * be attached using the returns jQuery promise object.
 *
 * filename - The filename of the template to load.
 * params   - An optional object containing key/value arguments to be
 *            passed into the template.
 * success  - An optional success callback to be called on load. This will
 *            recieve the HTML string as the first argument.
 * error    - An optional error callback to be called if the request fails.
 *
 * Example
 *
 *   client.getTemplate('index.html', {limit: 1}, function (html) {
 *     module.el.html(html);
 *   });
 *
 * Returns a jqXHR promise object that can be used to attach callbacks.
 */
```

Testing

For unit testing we use the following libraries.

- Mocha: As a BDD unit testing framework.
- Sinon: Provides spies, stubs and mocks for methods and functions.
- Chai: Provides common assertions.

Tests are run from the test/index.html directory. We use the BDD interface (`describe()`, `it()` etc.) provided by mocha and the assert interface provided by chai.

Generally we try and have the core functionality of all libraries and modules unit tested.

Best Practices

Forms

All forms should work without JavaScript enabled. This means that they must submit `application/x-www-form-urlencoded` data to the server and receive an appropriate response. The server should check for the `X-Requested-With: XMLHTTPRequest` header to determine if the request is an ajax one. If so it can return an appropriate format, otherwise it should issue a 303 redirect.

The one exception to this rule is if a form or button is injected with JavaScript after the page has loaded. It's then not part of the HTML document and can submit any data format it pleases.

Ajax

Note: Calls to the CKAN API from JavaScript should be done through the CKAN client.

Ajax requests can be used to improve the experience of submitting forms and other actions that require server interactions. Nearly all requests will go through the following states.

1. User clicks button.
2. JavaScript intercepts the click and disables the button (add `disabled` attr).
3. A loading indicator is displayed (add class `.loading` to button).
4. The request is made to the server.
5. (a) On success the interface is updated.

   (b) On error a message is displayed to the user if there is no other way to resolve the issue.
6. The loading indicator is removed.
7. The button is re-enabled.

Here's a possible example for submitting a search form using jQuery.

```
jQuery('#search-form').submit(function (event) {
  var form = $(this);
  var button = $('[type=submit]', form);

  // Prevent the browser submitting the form.
  event.preventDefault();

  button.prop('disabled', true).addClass('loading');

  jQuery.ajax({
    type: this.method,
    data: form.serialize(),
    success: function (results) {
      updatePageWithResults(results);
    },
    error: function () {
      showSearchError('Sorry we were unable to complete this search');
    },
    complete: function () {
      button.prop('disabled', false).removeClass('loading');
    }
  });
});
```

This covers possible issues that might arise from submitting the form as well as providing the user with adequate feedback that the page is doing something. Disabling the button prevents the form being submitted twice and the error feedback should hopefully offer a solution for the error that occurred.

Event Handlers

When using event handlers to listen for browser events it's a common requirement to want to cancel the default browser action. This should be done by calling the `event.preventDefault()` method:

```
jQuery('button').click(function (event) {
  event.preventDefault();
});
```

It is also possible to return `false` from the callback function. Avoid doing this as it also calls the `event.stopPropagation()` method which prevents the event from bubbling up the DOM tree. This prevents other handlers listening for the same event. For example an analytics click handler attached to the `<body>` element.

Also jQuery (1.7+) now provides the .on() and .off() methods as alternatives to `.bind()`, `.unbind()`, `.delegate()` and `.undelegate()` and they should be preferred for all tasks.

Templating

Small templates that will not require customisation by the instance can be placed inline. If you need to create multi-line templates use an array rather than escaping newlines within a string:

```
var template = [
  '<li>',
  '<span></span>',
  '</li>'
].join('');
```

Always localise text strings within your templates. If you are including them inline this can always be done with jQuery:

```
jQuery(template).find('span').text(_('This is my text string'));
```

Larger templates can be loaded in using the CKAN snippet API. Modules get access to this functionality via the `sandbox.client` object:

```
initialize: function () {
  var el = this.el;
  this.sandbox.client.getTemplate('dataset.html', function (html) {
    el.html(html);
  });
}
```

The primary benefits of this is that the localisation can be done by the server and it keeps the JavaScript modules free from large strings.

- CKAN Coding Standards
- Python Coding Standards
- HTML Coding Standards
- CSS Coding Standards
- JavaScript Coding Standards

8.4 Commit Messages

Generally, follow the commit guidelines from the Pro Git book:

- Try to make each commit a logically separate, digestible changeset.
- The first line of the commit message should concisely summarise the changeset.
- Optionally, follow with a blank line and then a more detailed explanation of the changeset.
- Use the imperative present tense as if you were giving commands to the codebase to change its behaviour, e.g. *Add tests for...*, *make xyzzy do frotz...*, this helps to make the commit message easy to read.

If your commit has an issue in the CKAN issue tracker put the issue number at the start of the first line of the commit message like this: `[#123]`. This makes the CKAN release manager's job much easier!

Here's an example of a good CKAN commit message:

```
[#2505] Update source install instructions

Following feedback from markw (see #2406).
```

8.5 Frontend Development Guidelines

8.5.1 Frontend Development Guidelines

Install front end dependencies

The front end stylesheets are written using LESS (this depends on node.js being installed on the system)

Instructions for installing node can be found on the node.js website. On Ubuntu node.js (and npm node.js's package manager) can be installed using the following command:

```
$ sudo apt-get install nodejs npm
```

LESS can then be installed via the node package manager which is bundled with node (or installed with apt as it is not bundled with node.js on Ubuntu). We also use `nodewatch` to make our LESS compiler a watcher style script.

`cd` into the `pyenv/src/ckan` and run:

```
$ npm install less nodewatch
```

File Structure

All front-end files to be served via a web server are located in the `public` directory (in the case of the new CKAN base theme it's `public/base`).

```
css/
  main.css
less/
  main.less
  ckan.less
  ...
javascript/
  main.js
  utils.js
  components/
  ...
vendor/
  jquery.js
  jquery.plugin.js
  underscore.js
  bootstrap.css
  ...
test/
  index.html
  spec/
    main.spec.js
```

```
    utils.spec.js
  vendor/
    mocha.js
    mocha.css
    chai.js
  ...
```

All files and directories should be lowercase with hyphens used to separate words.

css Should contain any site specific CSS files including compiled production builds generated by LESS.

less Should contain all the less files for the site. Additional vendor styles should be added to the *vendor* directory and included in main.less.

javascript Should contain all website files. These can be structured appropriately. It is recommended that *main.js* be used as the bootstrap filename that sets up the page.

vendor Should contain all external dependencies. These should not contain version numbers in the filename. This information should be available in the header comment of the file. Library plugins should be prefixed with the library name. If a dependency has many files (such as bootstrap) then the entire directory should be included as distributed by the maintainer.

test Contains the test runner *index.html*. *vendor* contains all test dependencies and libraries. *spec* contains the actual test files. Each test file should be the filename with *.spec* appended.

Stylesheets

Because all the stylesheets are using LESS we need to compile them before beginning development. In production CKAN will look for the `main.css` file which is included in the repository. In development CKAN looks for the file `main.debug.css` which you will need to generate by running:

```
$ ./bin/less
```

This will watch for changes to all of the less files and automatically rebuild the CSS for you. To quit the script press `ctrl-c`. There is also `--production` flag for compiling the production `main.css`.

There are many LESS files which attempt to group the styles in useful groups. The main two are:

main.less: This contains *all* the styles for the website including dependancies and local styles. The only files that are excluded here are those that are conditionally loaded such as IE only CSS and large external apps (like recline) that only appear on a single page.

ckan.less: This includes all the local ckan stylesheets.

Note: Whenever a CSS change effects `main.less` it's important than after the merge into master that a `$ ./bin/less --production` should be run and commited.

There is a basic pattern primer available at: http://localhost:5000/testing/primer/ that shows all the main page elements that make up the CKAN core interface.

JavaScript

The core of the CKAN JavaScript is split up into three areas.

- Core (such as i18n, pub/sub and API clients)
- Building a JavaScript Module (small HTML components or widgets)
- jQuery Plugins (very small reusable components)

Core

Everything in the CKAN application lives on the `ckan` namespace. Currently there are four main components that make up the core.

- Modules
- Publisher/Subscriber
- Client
- i18n/Jed

Modules

Modules are the core of the CKAN website, every component that is interactive on the page should be a module. These are then initialized by including a `data-module` attribute on an element on the page. For example:

:: <select name="format" data-module="autocomplete"></select>

The idea is to create small isolated components that can easily be tested. They should ideally not use any global objects, all functionality should be provided to them via a "sandbox" object.

There is a global factory that can be used to create new modules and jQuery and Localisation methods are available via `this.sandbox.jQuery` and `this.sandbox.translate()` respectively. To save typing these two common objects we can take advantage of JavaScript closures and use an alternative module syntax that accepts a factory function.

```
ckan.module('my-module', function (jQuery, _) {
  return {
    initialize: function () {
      // Called when a module is created.
      // jQuery and translate are available here.
    },
    teardown: function () {
      // Called before a module is removed from the page.
    }
  }
});
```

Note: A guide on creating your own modules is located in the Building a JavaScript Module guide.

Publisher/Subscriber

There is a simple pub/sub module included under `ckan.pubsub` it's methods are available to modules via `this.sandbox.publish/subscribe/unsubscribe`. This can be used to publish messages between modules.

Modules should use the publish/subscribe methods to talk to each other and allow different areas of the UI to update where relevant.

```
ckan.module('language-picker', function (jQuery, _) {
  return {
    initialize: function () {
      var sandbox = this.sandbox;
      this.el.on('change', function () {
        sandbox.publish('change:lang', this.selected);
```

```
      });
    }
  }
});

ckan.module('language-notifier', function (jQuery, _) {
  return {
    initialize: function () {
      this.sandbox.subscribe('change:lang', function (lang) {
        alert('language is now ' + lang);
      });
    }
  }
});
```

Client

Ideally no module should use jQuery.ajax() to make XHR requests to the CKAN API, all functionality should be provided via the client object.

```
ckan.module('my-module', function (jQuery, _) {
  return {
    initialize: function () {
      this.sandbox.client.getCompletions(this.options.completionsUrl);
    }
  }
});
```

i18n/Jed

Jed is a Gettext implementation in JavaScript. It is used throughout the application to create translatable strings. An instance of Jed is available on the `ckan.i18n` object.

Modules get access to the `translate()` function via both the initial factory function and the `this.sandbox.translate()` object.

String interpolation can be provided using the sprintf formatting. We always use the named arguments to keep in line with the Python translations. And we name the translate function passed into `ckan.module()` `_`.

```
ckan.module('my-module', function (jQuery, _) {
  return {
    initialize: function () {
      // Through sandbox translation
      this.sandbox.translate('my string');

      // Keyword arguments
      _('Hello %(name)s').fetch({name: 'Bill'}); // Hello Bill

      // Multiple.
      _("I like your %(color)s %(fruit)s.").fetch({color: 'red', fruit: 'apple');

      // Plurals.
      _("I have %(num)d apple.")
        .ifPlural(2, "I have %(num)d apples.")
        .fetch({num: 2, fruit: 'apple');
    }
```

```
  };
});
```

Life Cycle

CKAN modules are intialised on dom ready. The `ckan.module.initialize()` will look for all elements on the page with a `data-module` attribute and attempt to create an instance.

```
<select name="format" data-module="autocomplete" data-module-key="id"></select>
```

The module will be created with the element, any options object extracted from `data-module-*` attributes and a new sandbox instance.

Once created the modules `initialize()` method will be called allowing the module to set themselves up.

Modules should also provide a `teardown()` method this isn't used at the moment except in the unit tests to restore state but may become useful in the future.

Internationalization

All strings within modules should be internationalised. Strings can be set in the `options.i18n` object and there is a `.i18n()` helper for retrieving them.

```
ckan.module('language-picker', function (jQuery, _) {
  return {
    options: {
      i18n: {
        hello_1: _('Hello'),
        hello_2: _('Hello %(name)s'),
        apples: function (params) {
          var n = params.num;
          return _('I have %(num)d apple').isPlural(n, 'I have %(num)d apples');
        }
      }
    },
    initialize: function () {
      // Standard example
      this.i18n('hello_1'); // "Hello"

      // String interpolation example
      var name = 'Dave';
      this.i18n('hello_2', {name: name}); // "Hello Dave"

      // Plural example
      var total = 1;
      this.i18n('apples', {num: total}); // "I have 1 apple"
      this.i18n('apples', {num: 3});     // "I have 3 apples"
    }
  }
});
```

jQuery Plug-ins

Any functionality that is not directly related to ckan should be packaged up in a jQuery plug-in if possible. This keeps the modules containing only ckan specific code and allows plug-ins to be reused on other sites.

Examples of these are `jQuery.fn.slug()`, `jQuery.fn.slugPreview()` and `jQuery.proxyAll()`.

Unit Tests

There is currently a test suite available at: http://localhost:5000/base/test/index.html

Every core component, module and plugin should have a set of unit tests. Tests can be filtered using the `grep={regexp}` query string parameter.

The libraries used for the tests are as follows.

- Mocha: A test runner using a BDD style syntax.
- Chai: An assertion library (we use the assert style).
- Sinon: A stubbing library, can stub objects, timers and ajax requests.

Each file has a description block for it's top level object and then within that a nested description for each method that is to be tested:

```
describe('ckan.module.MyModule()', function () {
  describe('.initialize()', function () {
    it('should do something...', function () {
      // assertions.
    });
  });

  describe('.myMethod(arg1, arg2, arg3)', function () {
  });
});
```

The `` `.beforeEach()` `` and `` `.afterEach()` `` callbacks can be used to setup objects for testing (all blocks share the same scope so test variables can be attached):

```
describe('ckan.module.MyModule()', function () {
  // Pull the class out of the registry.
  var MyModule = ckan.module.registry['my-module'];

  beforeEach(function () {
    // Create a test element.
    this.el = jQuery('<div />');

    // Create a test sandbox.
    this.sandbox = ckan.sandbox();

    // Create a test module.
    this.module = new MyModule(this.el, {}, this.sandbox);
  });

  afterEach(function () {
    // Clean up.
    this.module.teardown();
  });
});
```

Templates can also be loaded using the `.loadFixtures()` method that is available in all test contexts. Tests can be made asynchronous by setting a `done` argument in the callback (Mocha checks the arity of the functions):

```
describe('ckan.module.MyModule()', function () {
```

```
  before(function (done) {
    // Load the template once.
    this.loadFixture('my-template.html', function (html) {
      this.template = html;
      done();
    });
  });

  beforeEach(function () {
    // Assign the template to the module each time.
    this.el = this.fixture.html(this.template).children();
  });
```

8.5.2 Templating

Within CKAN 2.0 we moved out templating to use Jinja2 from Genshi. This was done to provide a more flexible, extensible and most importantly easy to understand templating language.

Some useful links to get you started.

- Jinja2 Homepage
- Jinja2 Developer Documentation
- Jinja2 Template Documentation

Legacy Templates

Existing Genshi templates have been moved to the *templates_legacy* directory and will continue to be served if no file with the same name is located in *templates*. This should ensure backward compatibility until instances are able to upgrade to the new system.

The lookup path for templates is as follows. Give the template path "user/index.html":

1. Look in the template directory of each loaded extension.
2. Look in the template_legacy directory for each extension.
3. Look in the main ckan template directory.
4. Look in the template_legacy directory.

CKAN will automatically determine the template engine to use.

File Structure

The file structure for the CKAN templates is pretty much the same as before with a directory per controller and individual files per action.

With Jinja2 we also have the ability to use snippets which are small fragments of HTML code that can be pulled in to any template. These are kept in a snippets directory within the same folder as the actions that are using them. More generic snippets are added to templates/snippets.

```
templates/
  base.html             # A base template with just core HTML structure
  page.html             # A base template with default page layout
  header.html           # The site header.
  footer.html           # The site footer.
```

```
  snippets/              # A folder of generic sitewide snippets
  home/
    index.html           # Template for the index action of the home controller
    snippets/            # Snippets for the home controller
  user/
    ...
templates_legacy/
  # All ckan templates
```

Using the templating system

Jinja2 makes heavy use of template inheritance to build pages. A template for an action will tend to inherit from *page.html*:

```
{% extends "page.html" %}
```

Each parent defines a number of blocks that can be overridden to add content to the page. *page.html* defines majority of the markup for a standard page. Generally only `{% block primary_content %}` needs to be extended:

```
{% extends "page.html" %}

{% block page_content.html %}
  <h1>My page title</h1>
  <p>This content will be added to the page</p>
{% endblock %}
```

Most template pages will define enough blocks so that the extending page can customise as little or as much as required.

Internationalisation

Jinja2 provides a couple of helpers for internationalisation. The most common is to use the `_()` function:

```
{% block page_content.html %}
  <h1>{{ _('My page title') }}</h1>
  <p>{{ _('This content will be added to the page') }}</p>
{% endblock %}
```

Variables can be provided using the "format" function:

```
{% block page_content.html %}
  <p>{{ _('Welcome to CKAN {name}').format(name=username) }}</p>
{% endblock %}
```

For longer multiline blocks the `{% trans %}` block can be used.

```
{% block page_content.html %}
  <p>
    {% trans name=username %}
      Welcome to CKAN {{ name }}
    {% endtrans %}
  </p>
{% endblock %}
```

Conventions

There are a few common conventions that have evolved from using the language.

Includes

Note: Includes should be avoided as they are not portable use {% snippet %} tags whenever possible.

Snippets of text that are included using `{% include %}` should be kept in a directory called _snippets_. This should be kept in the same directory as the code that uses it.

Generally we use the `{% snippet %}` helper in all theme files unless the parents context must absolutely be available to the snippet. In which case the usage should be clearly documented.

Snippets

Note: {% snippet %} tags should be used in favour of h.snippet()

Snippets are essentially middle ground between includes and macros in that they are includes that allow a specific context to be provided (includes just receive the parent context).

These should be preferred to includes at all times as they make debugging much easier.

Macros

Macros should be used very sparingly to create custom generators for very generic snippets of code. For example macros/form.html has macros for creating common form fields.

They should generally be avoided as they are hard to extend and customise.

Templating within extensions

When you need to add or customize a template from within an extension you need to tell CKAN that there is a template directory that it can call from. Within your `update_config` method for the extension you'll need to add a `extra_template_paths` to the `config`.

Custom Control Structures

We've provided a few additional control structures to make working with the templates easier. Other helpers can still be used using the `h` object as before.

ckan_extends

```
{% ckan_extends %}
```

This works in a very similar way to `{% extend %}` however it will load the next template up in the load path with the same name.

For example if you wish to remove the breadcrumb from the user profile page in your own site. You would locate the template you wish to override.

```
ckan/templates/user/read.html
```

And create a new one in your theme extension.

```
ckanext-mytheme/ckanext/mytheme/templates/user/read.html
```

In this new file you would pull in the core template using `{% ckan_extends %}`:

```
{% ckan_extends %}
```

This will now render the current user/read page but we can override any portion that we wish to change. In this case the `breadcrumb` block.

```
{% ckan_extends %}

{# Remove the breadcrumb #}
{% block breadcrumb %}{% endblock %}
```

This function works recursively and so is ideal for extensions that wish to add a small snippet of functionality to the page.

Note: {% ckan_extend %} only extends templates of the same name.

---

snippet

```
{% snippet [filepath], [arg1=arg1], [arg2=arg2]... %}
```

Snippets work very much like Jinja2's `{% include %}` except that that do not inherit the parent templates context. This means that all variables must be explicitly passed in to the snippet. This makes debugging much easier.

```
{% snippet "package/snippets/package_form.html", data=data, errors=errors %}
```

url_for

```
{% url_for [arg1=arg1], [arg2=arg2]... %}
```

Works exactly the same as `h.url_for()`:

```
<a href="{% url_for controller="home", action="index" %}">Home</a>
```

link_for

```
{% link_for text, [arg1=arg1], [arg2=arg2]... %}
```

Works exactly the same as `h.link_for()`:

```
<li>{% link_for _("Home"), controller="home", action="index" %}</li>
```

url_for_static

```
{% url_for_static path %}
```

Works exactly the same as `h.url_for_static()`:

```
<script src="{% url_for_static "/javascript/home.js" %}"></script>
```

Form Macros

For working with forms we have provided some simple macros for generating common fields. These will be suitable for most forms but anything more complicated will require the markup to be written by hand.

The macros can be imported into the page using the `{% import %}` command.

```
{% import 'macros/form.html' as form %}
```

The following fields are provided:

form.input()

Creates all the markup required for an input element. Handles matching labels to inputs, error messages and other useful elements.

```
name        - The name of the form parameter.
id          - The id to use on the input and label. Convention is to prefix with 'field-'.
label       - The human readable label.
value       - The value of the input.
placeholder - Some placeholder text.
type        - The type of input eg. email, url, date (default: text).
error       - A list of error strings for the field or just true to highlight the field.
classes     - An array of classes to apply to the control-group.
```

Examples:

```
{% import 'macros/form.html' as form %}
{{ form.input('title', label=_('Title'), value=data.title, error=errors.title) }}
```

form.checkbox()

Builds a single checkbox input.

```
name        - The name of the form parameter.
id          - The id to use on the input and label. Convention is to prefix with 'field-'.
label       - The human readable label.
value       - The value of the input.
checked     - If true the checkbox will be checked
error       - An error string for the field or just true to highlight the field.
classes     - An array of classes to apply to the control-group.
```

Example:

```
{% import 'macros/form.html' as form %}
{{ form.checkbox('remember', checked=true) }}
```

form.select()

Creates all the markup required for an select element. Handles matching labels to inputs and error messages.

A field should be a dict with a "value" key and an optional "text" key which will be displayed to the user. `{"value": "my-option", "text": "My Option"}`. We use a dict to easily allow extension in future should extra options be required.

```
name        - The name of the form parameter.
id          - The id to use on the input and label. Convention is to prefix with 'field-'.
label       - The human readable label.
options     - A list/tuple of fields to be used as <options>.
selected    - The value of the selected <option>.
error       - A list of error strings for the field or just true to highlight the field.
classes     - An array of classes to apply to the control-group.
```

Examples:

```
{% import 'macros/form.html' as form %}
{{ form.select('year', label=_('Year'), options={'value': 2010, 'value': 2011}, selected=2011, error=
```

form.textarea()

Creates all the markup required for a plain textarea element. Handles matching labels to inputs, selected item and error messages.

```
name        - The name of the form parameter.
id          - The id to use on the input and label. Convention is to prefix with 'field-'.
label       - The human readable label.
value       - The value of the input.
placeholder - Some placeholder text.
error       - A list of error strings for the field or just true to highlight the field.
classes     - An array of classes to apply to the control-group.
```

Examples:

```
{% import 'macros/form.html' as form %}
{{ form.textarea('desc', id='field-description', label=_('Description'), value=data.desc, error=erro
```

form.markdown()

Creates all the markup required for a Markdown textarea element. Handles matching labels to inputs, selected item and error messages.

```
name        - The name of the form parameter.
id          - The id to use on the input and label. Convention is to prefix with 'field-'.
label       - The human readable label.
value       - The value of the input.
placeholder - Some placeholder text.
error       - A list of error strings for the field or just true to highlight the field.
classes     - An array of classes to apply to the control-group.
```

Examples:

```
{% import 'macros/form.html' as form %}
{{ form.markdown('desc', id='field-description', label=_('Description'), value=data.desc, error=erro
```

form.prepend()

Creates all the markup required for an input element with a prefixed segment. These are useful for showing url slugs and other fields where the input information forms only part of the saved data.

```
name        - The name of the form parameter.
id          - The id to use on the input and label. Convention is to prefix with 'field-'.
label       - The human readable label.
prepend     - The text that will be prepended before the input.
value       - The value of the input.
              which will use the name key as the value.
placeholder - Some placeholder text.
error       - A list of error strings for the field  or just true to highlight the field.
classes     - An array of classes to apply to the control-group.
```

Examples:

```
{% import 'macros/form.html' as form %}
{{ form.prepend('slug', id='field-slug', prepend='/dataset/', label=_('Slug'), value=data.slug, erro
```

form.custom()

Creates all the markup required for an custom key/value input. These are usually used to let the user provide custom meta data. Each "field" has three inputs one for the key, one for the value and a checkbox to remove it. So the arguments for this macro are nearly all tuples containing values for the (key, value, delete) fields respectively.

```
name        - A tuple of names for the three fields.
id          - An id string to be used for each input.
label       - The human readable label for the main label.
values      - A tuple of values for the (key, value, delete) fields. If delete
              is truthy the checkbox will be checked.
placeholder - A tuple of placeholder text for the (key, value) fields.
error       - A list of error strings for the field or just true to highlight the field.
classes     - An array of classes to apply to the control-group.
```

Examples:

```
{% import 'macros/form.html' as form %}
{{ form.custom(
     names=('custom_key', 'custom_value', 'custom_deleted'),
     id='field-custom',
     label=_('Custom Field'),
     values=(extra.key, extra.value, extra.deleted),
     error='')
}}
```

form.autoform()

Builds a form from the supplied form_info list/tuple.

```
form_info       - A list of dicts describing the form field to build.
data            - The form data object.
errors          - The form errors object.
error_summary   - The form errors object.
```

Example

```
{% set form_info = [
    {'name': 'ckan.site_title', 'control': 'input', 'label': _('Site Title'), 'placeholder': ''},
    {'name': 'ckan.main_css', 'control': 'select', 'options': styles, 'label': _('Style'), 'placehol
    {'name': 'ckan.site_description', 'control': 'input', 'label': _('Site Tag Line'), 'placeholder'
    {'name': 'ckan.site_logo', 'control': 'input', 'label': _('Site Tag Logo'), 'placeholder': ''},
    {'name': 'ckan.site_about', 'control': 'markdown', 'label': _('About'), 'placeholder': _('About p
    {'name': 'ckan.site_intro_text', 'control': 'markdown', 'label': _('Intro Text'), 'placeholder':
    {'name': 'ckan.site_custom_css', 'control': 'textarea', 'label': _('Custom CSS'), 'placeholder':
    ] %}

{% import 'macros/form.html' as form %}
{{ form.autoform(form_info, data, errors) }}
```

8.5.3 Resources

Note: Resources are only supported in the new Jinja2 style templates in CKAN 2.0 and above.

Resources are .css and .js files that may be included in an html page. Resources are included in the page by using the `{% resource %}` tag and CKAN uses Fanstatic to serve these resources.

```
{% resource 'library_name/resource_name' %}
```

Resources are grouped into libraries and the full resource name consists of `<library name>/<resource name>`. For example:

```
{% resource 'my_fanstatic_library/my_javascript_file.js' %}
```

It is important to note that these resources will be added to the page as defined by the resources, not in the location of the `{% resource %}` tag. Duplicate resources will not be added and any dependencies will be included as well as the resources, all in the correct order (see below for details).

Libraries can be added to CKAN from extensions using a helper function within the toolkit. See below.

In debug mode resources are served un-minified and unbundled (each resource is served separately). In non-debug mode the files are served minified and bundled (where allowed).

Important: .js and .css resources must be supplied as un-minified files. Minified files will be created. It is advised to include a .gitignore file to prevent minified files being added to the repository.

Resources within extensions

To add a resource within a extension helper function `add_resource(path, name)`:

```
ckan.plugins.toolkit.add_resource('path/to/my/fanstatic/library/dir',
       'my_fanstatic_library')
```

The first argument, `path`, is the path to the resource directory relative to the file calling the function. The second argument, `name` is the name of the library (to be used by templates when they want to include a resource from the library using the `{% resource %}` tag as shown above).

Resources will be created for the library for any .js and .css files found in the directory or it's subfolders. The resource name being the name of the file including any path needed to get to it from the resource directory. For greater control of the creation a `resource.config` file can be created and placed in the resource directory (see below for details).

resource.config

This file is used to define the resources in a directory and is sub folders. Here is an example file. The general layout of the file and allowed syntax is the same as for the .ini config file.

```
# Example resource.config file

[main]

dont_bundle = jquery.js
force_top = html5.js
order = jquery.js jed.js

[IE conditional]

lte IE 8 = html5.js block_html5_shim
IE 7 = font-awesome/css/font-awesome-ie7.css
others = html5.js

[custom render order]

block_html5_shim = 1
html5.js = 2
select2/select2.css = 9

[inline scripts]

block_html5_shim =
    var html5 = {shivMethods: false};

[depends]

vendor = jquery.js

[groups]

vendor =
    jed.js
    html5.js
    select2/select2.js
    select2/select2.css
    bootstrap/js/bootstrap-transition.js
    bootstrap/js/bootstrap-modal.js
    bootstrap/js/bootstrap-alert.js
    bootstrap/js/bootstrap-tab.js
    bootstrap/js/bootstrap-button.js
    font-awesome/css/font-awesome-ie7.css
```

[main]

This can contain the following values

force_top

The resources listed will be placed in the head of the page. This is only relevant to .js files which will by default will be added to the bottom of the page.

dont_bundle

Bundeling resources causes them to be served to the browser as a single resource to prevent multiple requests to the server. The resources listed will not be bundled. By default items will be bundled where possible. Note that .css files can only be bundled if they are in the same directory.

order

This is used to make sure that resources are created in the order specified. It should not generally be needed but is available if there are problems.

[IE conditional]

This allows IE conditionals to be wrapped around resources

eg `<!--[if IE lte 8]--><script src="my_script.js"></script><![end if]-->`

The condition is supplied followed by a list of resources that need that condition.

others

This is a special condition that means that the resource will also be available for none IE browsers.

[custom render order]

By default resources have a render order this is 10 for .css and 20 for .js resources. Sometimes we need to add resources before or after they would be included an example being the html5shim.js that needs including before .css resources. By providing a custom render order for the resource it's placement can be altered. Lower numbered resources are rendered earlier. Note that resources rendered in the head will still be placed before ones rendered in the body.

[inline scripts]

It is possible to define inline scripts in the resource.config file this can be helpful in some situations but is probably best avoided if possible.

[depends]

Some times one resource depends on another eg many scripts need jquery.js included in the page before them. External resource libraries will automatically depend on the core CKAN JavaScript modules so do not need to specify this.

[groups]

Groups of resources can be specified this allows the group to be included by just using it's name rather than having to specify each resource individuality when requesting them. The order that items are added to a group will be used to order the resources when added to the page but other factors such as dependencies, custom render order and resource type can affect the final order used.

Groups can be referred to in many places in the resource.config file eg. [depends]

8.5.4 Creating a new template

This is a brief tutorial covering the basics of building a common template.

Extending a base template

Firstly we need to extend a parent template to provide us with some basic page structure. This can be any other HTML page however the most common one is `page.html` which provides the full CKAN theme including header and footer.

```
{% extends "page.html" %}
```

The `page.html` template provides numerous blocks that can be extended. It's worth spending a few minutes getting familiar with what's available. The most common blocks that we'll be using are those ending with "content".

- `primary_content`: The main content area of the page.
- `secondary_content`: The secondary content (sidebar) of the page.
- `breadcrumb_content`: The contents of the breadcrumb navigation.
- `actions_content`: The content of the actions bar to the left of the breadcrumb.

Primary Content

For now we'll add some content to the main content area of the page.

```
{% block primary_content %}
  {{ super() }}

  {% block my_content %}
    <h2>{{ _('This is my content heading') }}</h2>
    <p>{{ _('This is my content') }}</p>
  {% endblock %}
{% endblock %}
```

Notice we've wrapped our own content in a block. This allows other templates to extend and possibly override this one and is extremely useful for making a them more customisable.

Secondary Content

Secondary content usually compromises of reusable modules which are pulled in as snippets. Snippets are also very useful for keeping the templates clean and allowing theme extensions to override them.

```
{% block primary_content %}
  {{ super() }}

  {% block my_sidebar_module %}
    {% snippet "snippets/my-sidebar-module.html" %}
  {% endblock %}
{% endblock %}
```

Breadcrumb and Actions

There is a consistent breadcrumb running through all the pages and often it is useful to provide additional actions that a related to the page.

```
{% block breadcrumb_content %}
  <li class="active">{% link_for _('Viewing Dataset'), controller='package', action='read', id=pkg.id
{% endblock %}
```

```
{% block actions_content %}
  {{ super() }}
  <li class="active">{% link_for _('New Dataset'), controller='package', action='new', class_='btn',
{% endblock %}
```

Scripts and Stylesheets

Currently scripts and stylesheets can be added by extending the `styles` and `scripts` blocks. This is soon to be replaced with the `{% resource %}` tag which manages script loading for us.

```
{% block styles %}
  {{ super() }}
  <link rel="stylesheet" href="{% url_for_static "my-style.css" %}" />
{% endblock %}

{% block scripts %}
  {{ super() }}
  <script src="{% url_for_static "my-script.js" %}"></script>
{% endblock %}
```

Summary

And that's about all there is to it be sure to check out `base.html` and `page.html` to see all the tags available for extension.

8.5.5 Template Blocks

These blocks can be extended by child templates to replace or extend common CKAN functionality.

Usage

There are currently two base templates *base.html* which provides the bare HTML structure such as title, head and body tags as well as hooks for adding links, stylesheets and scripts. *page.html* defines the content structure and is the template that you'll likely want to use.

To extend a template simply create a new template file and call `{% extend %}` then define the blocks that you wish to override.

8.5.6 Blocks in page.html

page.html extends the "page" block in *base.html* and provides the basic page structure for primary and secondary content.

header

Override the header on a page by page basis by extending this block. If making site wide header changes it is preferable to override the header.html file:

```
{% block header %}
  {% include "custom_header.html" %}
{% endblock %}
```

content

The content block allows you to replace the entire content section of the page with your own markup if needed:

```
{% block content %}
  <div class="custom-content">
    {% block custom_block %}{% endblock %}
  </div>
{% endblock %}
```

toolbar

The toolbar is for content to be added at the top of the page such as the breadcrumb navigation. You can remove/replace this by extending this block:

```
{# Remove the toolbar from this page. #}
{% block toolbar %}{% endblock %}
```

breadcrumb

Add a breadcrumb to the page by extending this element:

```
{% block breadcrumb %}
  {% include "breadcrumb.html" %}
{% endblock %}
```

actions

Add actions to the page by extending this element:

```
{% block actions %}
  <a class="btn" href="{{ save_url }}">Save</a>
{% endblock %}
```

primary

This block can be used to remove the entire primary content element:

```
{% block primary %}{% endblock %}
```

primary_content

The primary_content block can be used to add content to the page. This is the main block that is likely to be used within a template:

```
{% block primary_content %}
  <h1>My page content</h1>
  <p>Some content for the page</p>
{% endblock %}
```

secondary

This block can be used to remove the entire secondary content element:

```
{% block secondary %}{% endblock %}
```

secondary_content

The secondary_content block can be used to add content to the sidebar of the page. This is the main block that is likely to be used within a template:

```
{% block secondary_content %}
  <h2>A sidebar item</h2>
  <p>Some content for the item</p>
{% endblock %}
```

footer

Override the footer on a page by page basis by extending this block:

```
{% block footer %}
  {% include "custom_footer.html" %}
{% endblock %}
```

If making site wide header changes it is preferable to override the *footer.html*. Adding scripts should use the "scripts" block instead.

8.5.7 Blocks in base.html

doctype

Allows the DOCTYPE to be set on a page by page basis:

```
{% block doctype %}<!DOCTYPE html>{% endblock %}
```

htmltag

Allows custom attributes to be added to the <html> tag:

```
{% block htmltag %}<html lang="en-gb" class="no-js">{% endblock %}
```

headtag

Allows custom attributes to be added to the <head> tag:

```
{% block headtag %}<head data-tag="No idea what you'd add here">{% endblock %}
```

bodytag

Allows custom attributes to be added to the <body> tag:

```
{% block bodytag %}<body class="full-page">{% endblock %}
```

meta

Add custom meta tags to the page. Call `super()` to get the default tags such as charset, viewport and generator:

```
{% block meta %}
  {{ super() }}
  <meta name="author" value="Joe Bloggs" />
  <meta name="description" value="My website description" />
{% endblock %}
```

title

Add a custom title to the page by extending the title block. Call `super()` to get the default page title:

```
{% block title %}My Subtitle - {{ super() }}{% endblock %}
```

links

The links block allows you to add additional content before the stylesheets such as rss feeds and favicons in the same way as the meta block:

```
{% block link %}
  <meta rel="shortcut icon" href="custom_icon.png" />
{% endblock %}
```

styles

The styles block allows you to add additional stylesheets to the page in the same way as the meta block. Use ‘‘ super() ‘‘ to include the default stylesheets before or after your own:

```
{% block styles %}
  {{ super() }}
  <link rel="stylesheet" href="/base/css/custom.css" />
{% endblock %}
```

page

The page block allows you to add content to the page. Most of the time it is recommended that you extend one of the page.html templates in order to get the site header and footer. If you need a clean page then this is the block to use:

```
{% block page %}
  <div>Some other page content</div>
{% endblock %}
```

scripts

The scripts block allows you to add additonal scripts to the page. Use the `super()` function to load the default scripts before/after your own:

```
{% block script %}
  {{ super() }}
  <script src="/base/js/custom.js"></script>
{% endblock %}
```

8.5.8 Building a JavaScript Module

CKAN makes heavy use of modules to add additional functionality to the page. Essentially all a module consists of is an object with an `.initialize()` and `.teardown()` method.

Here we will go through the basic functionality of building a simple module that sends a "favourite" request to the server when the user clicks a button.

HTML

The idea behind modules is that the element should already be in the document when the page loads. For example our favourite button will work just fine without our module JavaScript loaded.

```
<form action="/favourite" method="post" data-module="favorite">
  <button class="btn" name="package" value="101">Submit</button>
</form>
```

Here it's the `data-module="favorite"` that tells the CKAN module loader to create a new instance for this element.

JavaScript

Modules reside in the *javascript/modules* directory and should share the same name as the module. We use hyphens to delimit spaces in both filenames and modules.

```
/javascript/modules/favorite.js
```

A module can be created by calling `ckan.module()`:

```
ckan.module('favorite', function (jQuery, _) {
  return {};
});
```

We pass in the module name and a factory function that should return our module object. This factory gets passed a local jQuery object and a translation object.

Note: In order to include a module for page render inclusion within an extension it is recommended that you use `{% resource %}` within your templates. See the Resource Documentation

---

Initialisation

Once ckan has found an element on the page it creates a new instance of your module and if present calls the `.initialize()` method.

```
ckan.module('favorite', function (jQuery, _) {
  return {
    initialize: function () {
      console.log('I've been called for element: %o', this.el);
    }
  };
});
```

Here we can set up event listeners and other setup functions.

```
initialize: function () {
  // Grab our button and assign it to a property of our module.
  this.button = this.$('button');

  // Watch for our favourite button to be clicked.
  this.button.on('submit', jQuery.proxy(this._onClick, this));
},
_onClick: function (event) {}
```

Event Handling

Now we create our click handler for the button:

```
_onClick: function (event) {
  event.preventDefault();
  this.favorite();
}
```

And this calls a `.favorite()` method. It's generally best not to do too much in event handlers it means that you can't use the same functionality elsewhere.

```
favorite: function () {
  // The client on the sandbox should always be used to talk to the api.
  this.sandbox.client.favoriteDataset(this.button.val());
}
```

Notifications and Internationalisation

This submits the dataset to the API but ideally we want to tell the user what we're doing.

```
options: {
  i18n: {
    loading: _('Favouriting dataset'),
    done: _('Favourited dataset %(id)s')
  }
},
favorite: function () {
  // i18n gets a translation key from the options object.
  this.button.text(this.i18n('loading'));

  // The client on the sandbox should always be used to talk to the api.
  var request = this.sandbox.client.favoriteDataset(this.button.val())
  request.done(jQuery.proxy(this._onSuccess, this));
},
_onSuccess: function () {
  // We can perform interpolation on messages.
```

```
  var message = this.i18n('done', {id: this.button.val()});

  // Notify allows global messages to be displayed to the user.
  this.sandbox.notify(message, 'success');
}
```

Options

Displaying an id to the user isn't very friendly. We can use the `data-module` attributes to pass options through to the module.

```
<form action="/favourite" method="post" data-module="favorite" data-module-dataset="my dataset">
```

This will override the defaults in the options object.

```
ckan.module('favorite', function (jQuery, _) {
  return {
    options: {
      dataset: '',
      i18n: {...}
    }
    initialize: function () {
      console.log('this dataset is: %s', this.options.dataset);
      //=> "this dataset is: my dataset"
    }
  };
});
```

Error handling

When ever we make an Ajax request we want to make sure that we notify the user if the request fails. Again we can use `this.sandbox.notify()` to do this.

```
favorite: function () {
  // The client on the sandbox should always be used to talk to the api.
  var request = this.sandbox.client.favoriteDataset(this.button.val())
  request.done(jQuery.proxy(this._onSuccess, this));
  request.fail(jQuery.proxy(this._onError, this));
},
_onError: function () {
  var message = this.i18n('error', {id: this.button.val()});

  // Notify allows global messages to be displayed to the user.
  this.sandbox.notify(message, 'error');
}
```

Module Scope

You may have noticed we keep making calls to `jQuery.proxy()` within these methods. This is to ensure that `this` when the callback is called is the module it belongs to.

We have a shortcut method called `jQuery.proxyAll()` that can be used in the `.initialize()` method to do all the binding at once. It can accept method names or simply a regexp.

```
initialize: function () {
  jQuery.proxyAll(this, '_onSuccess');

  // Same as:
  this._onSuccess = jQuery.proxy(this, '_onSuccess');

  // Even better do all methods starting with _on at once.
  jQuery.proxyAll(this, /_on/);
}
```

Publish/Subscribe

Sometimes we want modules to be able to talk to each other in order to keep the page state up to date. The sandbox has the `.publish()` and `.subscribe()` methods for just this cause.

For example say we had a counter up in the header that showed how many favourite datasets the user had. This would be incorrect when the user clicked the ajax button. We can publish an event when the favorite button is successful.

```
_onSuccess: function () {
  // Notify allows global messages to be displayed to the user.
  this.sandbox.notify(message, 'success');

  // Tell other modules about this event.
  this.sandbox.publish('favorite', this.button.val());
}
```

Now in our other module 'user-favorite-counter' we can listen for this.

```
ckan.module('user-favorite-counter', function (jQuery, _) {
  return {
    initialize: function () {
      jQuery.proxyAll(this, /_on/);
      this.sandbox.subscribe('favorite', this._onFavorite);
    },
    teardown: function () {
      // We must always unsubscribe on teardown to prevent memory leaks.
      this.sandbox.unsubscribe('favorite', this._onFavorite);
    },
    incrementCounter: function () {
      var count = this.el.text() + 1;
      this.el.text(count);
    },
    _onFavorite: function (id) {
      this.incrementCounter();
    }
  };
});
```

Unit Tests

Every module has unit tests. These use Mocha, Chai and Sinon to assert the expected functionality of the module.

- Frontend Development
- Templating
- Resources

- Template Tutorial
- Template Blocks
- JavaScript Module Tutorial

8.6 Writing Documentation

The quickest and easiest way to contribute documentation to CKAN is to sign up for a free GitHub account and simply edit the CKAN Wiki. Docs started on the wiki can make it onto docs.ckan.org later.

Tip: Use the reStructuredText markup format when creating a wiki page, since reStructuredText is the format that docs.ckan.org uses, this will make moving the documentation from the wiki into docs.ckan.org later easier.

For how to contribute to the offical CKAN documentation at docs.ckan.org, see the documentation guidelines.

8.6.1 Documentation guidelines

This section gives some guidelines to help us to write consistent and good quality documentation for CKAN.

Documentation isn't source code, and documentation standards don't need to be followed as rigidly as coding standards do. In the end, some documentation is better than no documentation, it can always be improved later. So the guidelines below are soft rules.

Having said that, we suggest just one hard rule: **no new feature (or change to an existing feature) should be missing from the docs** (but see *todo*).

See also:

Jacob Kaplon-Moss's Writing Great Documentation A series of blog posts about writing technical docs, a lot of our guidelines were based on this.

1. Getting started

This section will walk you through downloading the source files for CKAN's docs, editing them, and submitting your work to the CKAN project.

CKAN's documentation is created using Sphinx, which in turn uses Docutils (reStructuredText is part of Docutils). Some useful links to bookmark:

- Sphinx's reStructuredText Primer
- reStructuredText cheat sheet
- reStructuredText quick reference
- Sphinx Markup Constructs is a full list of the markup that Sphinx adds on top of Docutils.

The source files for the docs are in the doc directory of the CKAN git repo. The following sections will walk you through the process of making changes to these source files, and submitting your work to the CKAN project.

a. Install CKAN into a virtualenv

Create a Python virtual environment (virtualenv), activate it, install CKAN into the virtual environment, and install the dependencies necessary for building CKAN. In this example we'll create a virtualenv in a folder called `pyenv`. Run these commands in a terminal:

```
virtualenv --no-site-packages pyenv
. pyenv/bin/activate
pip install -e 'git+https://github.com/okfn/ckan.git#egg=ckan'
pip install -r pyenv/src/ckan/pip-requirements-docs.txt
```

b. Fetch CKAN's git submodules

CKAN's custom Sphinx theme is a kept in a git *submodule*, so before you can build the docs you have to run these commands in a terminal to download the submodule:

```
cd pyenv/src/ckan
git submodule init
git submodule update
```

You may occasionally have to run `git submodule update` again, when someone updates the theme, but this doesn't happen often.

c. Build the docs

You should now be able to build the CKAN documentation locally. Make sure your virtual environment is activated, and then run this command:

```
python setup.py build_sphinx
```

Now you can open the built HTML files in `pyenv/src/ckan/build/sphinx/html`, e.g.:

```
firefox pyenv/src/ckan/build/sphinx/html/index.html
```

d. Edit the reStructuredText files

To make changes to the documentation, use a text editor to edit the `.rst` files in `pyenv/src/ckan/doc/`. Save your changes and then build the docs again (`python setup.py build_sphinx`) and open the HTML files in a web browser to preview your changes.

Once your docs are ready to submit to the CKAN project, follow the steps in *Making a Pull Request*.

2. Structure and audience

Welcome to CKAN's Documentation describes the overall structure of the docs, and the intended audience for each part. This structure is intended to be clear, simple and extendable. If you're adding a new section to the docs, try to fit it into this structure.

Within a particular page, for example a new page documenting a new feature, our suggestion for what sections the page should have is:

1. **Overview**: a conceptual overview of or introduction to the feature. Explain what the feature provides, why someone might want to use it, and introduce any key concepts users need to understand. This is the **why** of the feature.

   If it's developer documentation (extension writing, theming, API, or core developer docs), maybe put an architecture guide here.

2. **Tutorials**: tutorials and examples for how to setup the feature, and how to use the feature. This is the **how**.
3. **Reference**: any reference docs such as config options or API functions.
4. **Troubleshooting**: common error messages and problems, FAQs, how to diagnose problems.

3. Style

This section covers things like what tone to use, how to capitalize section titles, etc. Having a consistent style will make the docs nice and easy to read and give them a complete, quality feel.

Use American spelling

Use American spellings everywhere: organization, authorization, realize, customize, initialize, color, etc. There's a list here: https://wiki.ubuntu.com/EnglishTranslation/WordSubstitution

Commonly used terms

CKAN Should be written in ALL-CAPS.

email Use email not e-mail.

PostgreSQL, SQLAlchemy, Nginx, Python, SQLite, JavaScript, etc. These should always be capitalized as shown above (including capital first letters for Python and Nginx even when they're not the first word in a sentence). `doc/conf.py` defines substitutions for each of these so you don't have to remember them, see *Substitutions*.

Web site Two words, with Web always capitalized

frontend Not front-end

command line Two words, not commandline or command-line (this is because we want to be like Neal Stephenson)

CKAN config file or configuration file Not settings file, ini file, etc. Also, the **config file** contains **config options** such as `ckan.site_id`, and each config option is **set** to a certain **setting** or **value** such as `ckan.site_id = demo.ckan.org`.

Section titles

Capitalization in section titles should follow the same rules as in normal sentences: you capitalize the first word and any proper nouns.

This seems like the easiest way to do consistent capitalization in section titles because it's a capitalization rule that we all know already (instead of inventing a new one just for section titles).

Right:

- Installing CKAN from package
- Getting started
- Command line interface
- Writing extensions
- Making an API request
- You're done!
- Libraries available to extensions

Wrong:

- Installing CKAN from Package
- Getting Started
- Command Line Interface

- Writing Extensions
- Making an API Request
- You're Done!
- Libraries Available To Extensions

For lots of examples of this done right, see Django's table of contents.

In Sphinx, use the following section title styles:

```
===============
Top-level title
===============

------------------
Second-level title
------------------

Third-level title
=================

Fourth-level title
------------------
```

If you need more than four levels of headings, you're probably doing something wrong, but see: http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html#sections

Be conversational

Write in a friendly, conversational and personal tone:

- Use contractions like don't, doesn't, it's etc.
- Use "we", for example *"We'll publish a call for translations to the ckan-dev and ckan-discuss mailing lists, announcing that the new version is ready to be translated"* instead of *"A call for translations will be published"*.
- Refer to the reader personally as "you", as if you're giving verbal instructions to someone in the room: *"First, you'll need to do X. Then, when you've done Y, you can start working on Z"* (instead of stuff like *"First X must be done, and then Y must be done..."*).

Write clearly and concretely, not vaguely and abstractly

Politics and the English Language has some good tips about this, including:

1. Never use a metaphor, simile, or other figure of speech which you are used to seeing in print.
2. Never use a long word where a short one will do.
3. If it's possible to cut out a word, always cut it out.
4. Never use the passive when you can be active.
5. Never use a foreign phrase, scientific word or jargon word if you can think of an everyday English equivalent.

This will make your meaning clearer and easier to understand, especially for people whose first language isn't English.

Facilitate skimming

Readers skim technical documentation trying to quickly find what's important or what they need, so break walls of text up into small, visually identifiable pieces:

- Use lots of inline markup:

```
*italics*
**bold**
``code``
```

  For code samples or filenames with variable parts, uses Sphinx's :samp: and :file: directives.

- Use lists to break up text.
- Use `.. note::` and `.. warning::`, see Sphinx's paragraph-level markup.

  (reStructuredText actually supports lots more of these: `attention`, `error`, `tip`, `important`, etc. but most Sphinx themes only style `note` and `warning`.)

- Break text into short paragraphs of 5-6 sentences each max.
- Use section and subsection headers to visualize the structure of a page.

4. Sphinx

This section gives some useful tips about using Sphinx.

Don't introduce any new Sphinx warnings

When you build the docs, Sphinx prints out warnings about any broken cross-references, syntax errors, etc. We aim not to have any of these warnings, so when adding to or editing the docs make sure your changes don't introduce any new ones.

It's best to delete the `build` directory and completely rebuild the docs, to check for any warnings:

```
rm -rf build; python setup.py build_sphinx
```

Maximum line length

As with Python code, try to limit all lines to a maximum of 79 characters.

Cross-references and links

Use `:doc:` to cross-reference to other files by filename:

```
See :doc:`theming`
```

see Cross-referencing documents for details.

Use `:ref:` to cross-reference to particular sections within the same or another file. First you have to add a label before the section you want to cross-reference to:

```
.. _getting-started:

---------------
Getting started
---------------
```

then from elsewhere cross-reference to the section like this:

```
See :ref:`getting-started`.
```

see Cross-referencing arbitrary locations.

With both `:doc:` and `:ref:` if you want the link text to be different from the title of the section you're referencing, do this:

```
:doc:`the theming document <theming>`

:ref:`the getting started section <getting-started>`
```

The syntax for linking to external URLs is slightly different from cross-referencing, you have to add a trailing underscore:

```
`Link text <http://example.com/>`_
```

or to define a URL once and then link to it in multiple places, do:

```
This is `a link`_ and this is `a link`_ and this is
`another link <a link>`_.

.. _a link: http://example.com/
```

see Hyperlinks for details.

Substitutions

Substitutions are a useful way to define a value that's needed in many places (eg. a command, the location of a file, etc.) in one place and then reuse it many times.

You define the value once like this:

```
.. |production.ini| replace:: /etc/ckan/default/production.ini
```

and then reuse it like this:

```
Now open your |production.ini| file.
```

`|production.ini|` will be replaced with the full value `/etc/ckan/default/production.ini`.

Substitutions can also be useful for achieving consistent spelling and capitalization of names like reStructuredText, PostgreSQL, Nginx, etc.

The `rst_epilog` setting in `doc/conf.py` contains a list of global substitutions that can be used from any file.

Substitutions can't immediately follow certain characters (with no space in-between) or the substitution won't work. If this is a problem, you can insert an escaped space, the space won't show up in the generated output and the substitution will work:

```
pip install -e 'git+\ |git_url|'
```

Similarly, certain characters are not allowed to immediately follow a substitution (without a space) or the substitution won't work. In this case you can just escape the following characters, the escaped character will show up in the output and the substitution will work:

```
pip install -e 'git+\ |git_url|\#egg=ckan'
```

Also see *Parsed literals* below for using substitutions in code blocks.

Parsed literals

Normally things like links and substitutions don't work within a literal code block. You can make them work by using a `parsed-literal` block, for example:

```
Copy your development.ini file to create a new production.ini file::

.. parsed-literal::

   cp |development.ini| |production.ini|
```

autodoc

We try to use autodoc to pull documentation from source code docstrings into our Sphinx docs, wherever appropriate. This helps to avoid duplicating documentation and also to keep the documentation closer to the code and therefore more likely to be kept up to date.

Whenever you're writing reference documentation for modules, classes, functions or methods, exceptions, attributes, etc. you should probably be using autodoc. For example, we use autodoc for the *Action API Reference*, the *Plugin API Documentation*, etc.

For how to write docstrings, see *Docstrings*.

todo

No new feature (or change to an existing feature) should be missing from the docs. It's best to document new features or changes as you implement them, but if you really need to merge something without docs then at least add a todo directive to mark where docs need to be added or updated (if it's a new feature, make a new page or section just to contain the `todo`):

```
====================================
CKAN's Builtin Social Network Feature
====================================

.. todo::

   Add docs for CKAN's builtin social network for data hackers.
```

versionadded and versionchanged

Use Sphinx's `versionadded` and `versionchanged` directives to mark new or changed features. For example:

```
================
Tag vocabularies
================

```

```
.. versionadded:: 1.7

CKAN sites can have *tag vocabularies*, which are a way of grouping related
tags together into custom fields.

...
```

With `versionchanged` you usually need to add a sentence explaining what changed (you can also do this with `versionadded` if you want):

```
=============
Authorization
=============

.. versionchanged:: 2.0
   Previous versions of CKAN used a different authorization system.

CKAN's authorization system controls which users are allowed to carry out
which...
```

deprecated

Use Sphinx's deprecated directive to mark things as deprecated in the docs:

```
.. deprecated:: 3.1
   Use :func:`spam` instead.
```

seealso

Often one page of the docs is related to other pages of the docs or to external pages. A seealso block is a nice way to include a list of related links:

```
.. seealso::

   :doc:`The DataStore extension <datastore>`
     A CKAN extension for storing data.

   CKAN's `demo site <http://demo.ckan.org/>`_
     A demo site running the latest CKAN beta version.
```

8.7 Making a Pull Request

Once you've written some CKAN code or documentation, you can submit it for review and merge into the central CKAN git repository by making a pull request. This section will walk you through the steps for making a pull request.

1. Create a git branch

   Each logically separate piece of work (e.g. a new feature, a bug fix, a new docs page, or a set of improvements to a docs page) should be developed on its own branch forked from the master branch.

   The name of the branch should include the issue number (if this work has an issue in the CKAN issue tracker), the branch type (e.g. "feature" or "bug"), and a brief one-line synopsis of the work, for example:

```
    2298-feature-add-sort-by-controls-to-search-page
    1518-bug-upload-file-with-spaces
```

2. Fork CKAN on GitHub

   Sign up for a free account on GitHub and fork CKAN, so that you have somewhere to publish your work.

   Add your CKAN fork to your local CKAN git repo as a git remote. Replace `USERNAME` with your GitHub username:

```
    git remote add my_fork https://github.com/USERNAME/ckan
```

3. Commit and push your changes

   Commit your changes on your feature branch, and push your branch to GitHub. For example, make sure you're currently on your feature branch then run these commands:

```
    git add doc/my_new_feature.rst
    git commit -m "Add docs for my new feature"
    git push my_fork my_branch
```

   When writing your git commit messages, try to follow the *Commit Messages* guidelines.

4. Send a pull request

   Once your work on a branch is complete and is ready to be merged into the master branch, create a pull request on GitHub. A member of the CKAN team will review your work and provide feedback on the pull request page. The reviewer may ask you to make some changes. Once your pull request has passed the review, the reviewer will merge your code into the master branch and it will become part of CKAN!

   When submitting a pull request:

   - Your branch should contain one logically separate piece of work, and not any unrelated changes.
   - You should have good commit messages, see *Commit Messages*.
   - Your branch should contain new or changed tests for any new or changed code, and all the CKAN tests should pass on your branch, see Testing CKAN.
   - Your branch should contain new or updated documentation for any new or updated code, see *Writing Documentation*.
   - Your branch should be up to date with the master branch of the central CKAN repo, so pull the central master branch into your feature branch before submitting your pull request.

     For long-running feature branches, it's a good idea to pull master into the feature branch periodically so that the two branches don't diverge too much.

8.7.1 Merging a Pull Request

If you're reviewing a pull request for CKAN, when merging a branch into master:

- Use the `--no-ff` option in the `git merge` command,

CHAPTER 9

Testing CKAN

If you're a CKAN developer, if you're developing an extension for CKAN, or if you're just installing CKAN from source, you should make sure that CKAN's tests pass for your copy of CKAN. This section explains how to run CKAN's tests.

9.1 Installing Additional Dependencies

Some additional dependencies are needed to run the tests. Make sure you've created a config file at /etc/ckan/default/development.ini, then activate your virtual environment:

```
. /usr/lib/ckan/default/bin/activate
```

Install nose and other test-specific CKAN dependencies into your virtual environment:

```
pip install -r /usr/lib/ckan/default/src/ckan/dev-requirements.txt
```

9.2 Testing with SQLite

To run the CKAN tests using SQLite as the database library:

```
cd /usr/lib/ckan/default/src/ckan
nosetests --ckan ckan
```

You *must* run the tests from the CKAN directory as shown above, otherwise the `--ckan` plugin won't work correctly.

In deployment CKAN uses PostgreSQL, not SQLite. Running the tests with SQLite is less thorough but much quicker than with PostgreSQL, good enough for an initial check but you should run the tests with PostgreSQL before deploying anything or releasing any code.

9.2.1 Testing Core Extensions

CKAN's core extensions (those extensions that are kept in the CKAN codebase alongside CKAN itself) have their own tests. For example, to run the tests for the stats extension do:

```
nosetests --ckan ckanext/stats
```

To run the tests for all of the core extensions at once:

```
nosetests --ckan ckanext
```

Or to run the CKAN tests and the core extensions tests together:

```
nosetests --ckan ckan ckanext
```

9.3 Testing with PostgreSQL

Changed in version 2.1: Previously PostgreSQL tests used the databases defined in your `development.ini` file, instead of using their own test databases.

Create test databases:

```
sudo -u postgres createdb -O ckan_default ckan_test -E utf-8
sudo -u postgres createdb -O ckan_default datastore_test -E utf-8
paster datastore set-permissions postgres -c test-core.ini
```

This database connection is specified in the `test-core.ini` file by the `sqlalchemy.url` parameter.

CKAN's default nose configuration file (`test.ini`) specifies SQLite as the database library (it also sets `faster_db_test_hacks`). To run the tests more thoroughly with PostgreSQL, specify the `test-core.ini` nose configuration file instead, for example:

```
nosetests --ckan --with-pylons=test-core.ini ckan
nosetests --ckan --with-pylons=test-core.ini ckanext/stats
nosetests --ckan --with-pylons=test-core.ini ckanext
nosetests --ckan --with-pylons=test-core.ini ckan ckanext
```

The speed of the PostgreSQL tests can be improved by running PostgreSQL in memory and turning off durability, as described in the PostgreSQL documentation.

9.4 Migration Testing

If you're a CKAN developer or extension developer and your new code requires a change to CKAN's model, you'll need to write a migration script. To ensure that the migration script itself gets tested, you should run the tests with they `--ckan-migration` option, for example:

```
nosetests --ckan --ckan-migration --with-pylons=test-core.ini ckan
```

By default tests are run using the model defined in `ckan/model`. With the `--ckan-migration` option the tests will run using a database that has been created by running the migration scripts in `ckan/migration`, which is how the database is created and upgraded in production.

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.

9.5 Common error messages

9.5.1 ConfigError

nose.config.ConfigError: Error reading config file 'setup.cfg': no such option 'with-pylo

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"
```

9.5.2 OperationalError

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.

9.5.3 nosetests

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 /usr/lib/ckan/default/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
```

9.6 Front-end Testing

All new CKAN features should be coded so that they work in the following browsers:

- Internet Explorer: 9, 8 and 7
- Firefox: Latest + previous version
- Chrome: Latest + previous version

These browsers are determined by whatever has >= 1% share with the latest months data from: http://data.gov.uk/data/site-usage

9.6.1 Install browser virtual machines

In order to test in all the needed browsers you'll need access to all the above browser versions. Firefox and Chrome should be easy whatever platform you are on. Internet Explorer is a little trickier. You'll need Virtual Machines.

We suggest you use https://github.com/xdissent/ievms to get your Internet Explorer virtual machines.

9.6.2 Testing methodology

Firstly we have a primer page. If you've touched any of the core front-end code you'll need to check if the primer is rendering correctly. The primer is located at: http://localhost:5000/testing/primer

Secondly whilst writing a new feature you should endeavour to test in at least in your core browser and an alternative browser as often as you can.

Thirdly you should fully test all new features that have a front-end element in all browsers before making your pull request into CKAN master.

9.6.3 Common pitfalls & their fixes

Here's a few of the most common front end bugs and a list of their fixes.

Reserved JS keywords

Since IE has a stricter language definition in JS it really doesn't like you using JS reserved keywords method names, variables, etc... This is a good list of keywords not to use in your JavaScript:

https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Reserved_Words

```
/* These are bad */
var a = {
  default: 1,
  delete: function() {}
};

/* These are good */
var a = {
  default_value: 1,
  remove: function() {}
};
```

Unclosed JS arrays / objects

Internet Explorer doesn't like it's JS to have unclosed JS objects and arrays. For example:

```
/* These are bad */
var a = {
  b: 'c',
};
var a = ['b', 'c', ];

/* These are good */
var a = {
  c: 'c'
```

```
};
var a = ['b', 'c'];
```

CHAPTER 10

Config File Options

You can set many important options in the CKAN config file. By default, the configuration file is located at `/etc/ckan/development.ini` or `/etc/ckan/production.ini`. This section documents all of the config file settings, for reference.

Todo

Insert cross-ref to section about location of config file?

Note: After editing your config file, you need to restart your webserver for the changes to take effect.

Note: Unless otherwise noted, all configuration options should be set inside the `[app:main]` section of the config file (i.e. after the `[app:main]` line):

```
[DEFAULT]

...

[server:main]
use = egg:Paste#http
host = 0.0.0.0
port = 5000

# This setting will not work, because it's outside of [app:main].
ckan.site_logo = /images/masaq.png

[app:main]
# This setting will work.
ckan.plugins = stats text_preview recline_preview
```

If the same option is set more than once in your config file, the last setting given in the file will override the others.

10.1 General Settings

10.1.1 debug

Example:

```
debug = False
```

Default value: `False`

This enables Pylons' interactive debugging tool, makes Fanstatic serve unminified JS and CSS files, and enables CKAN templates' debugging features.

Warning: This option should be set to `False` for a public site. With debug mode enabled, a visitor to your site could execute malicious commands.

10.2 Database Settings

10.2.1 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
```

10.2.2 ckan.datastore.write_url

Example:

```
ckan.datastore.write_url = postgresql://ckanuser:pass@localhost/datastore
```

The database connection to use for writing to the datastore (this can be ignored if you're not using the DataStore Extension). Note that the database used should not be the same as the normal CKAN database. The format is the same as in *sqlalchemy.url*.

10.2.3 ckan.datastore.read_url

Example:

```
ckan.datastore.read_url = postgresql://readonlyuser:pass@localhost/datastore
```

The database connection to use for reading from the datastore (this can be ignored if you're not using the DataStore Extension). The database used must be the same used in *ckan.datastore.write_url*, but the user should be one with read permissions only. The format is the same as in *sqlalchemy.url*.

10.3 Site Settings

10.3.1 ckan.site_url

Example:

```
ckan.site_url = http://scotdata.ckan.net
```

Default value: (none)

The URL of your CKAN site. Many CKAN features that need an absolute URL to your site use this setting.

Warning: This setting should not have a trailing / on the end.

10.3.2 ckan.api_url

Deprecated since version 2: No longer used.

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.

10.3.3 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 blocks the Authorization header and `X-CKAN-API-Key` is not suitable.

10.3.4 ckan.cache_expires

Example:

```
ckan.cache_expires = 2592000
```

Default value: 0

This sets `Cache-Control` header's max-age value.

10.3.5 ckan.page_cache_enabled

Example:

```
ckan.page_cache_enabled = True
```

Default value: `False`

This enables CKAN's built-in page caching.

Warning: Page caching is an experimental feature.

10.3.6 ckan.cache_enabled

Example:

```
ckan.cache_enabled = True
```

Default value: `None`

Controls if we're caching CKAN's static files, if it's serving them.

10.3.7 ckan.static_max_age

Example:

```
ckan.static_max_age = 2592000
```

Default value: `3600`

Controls CKAN static files' cache max age, if we're serving and caching them.

10.3.8 ckan.tracking_enabled

Example:

```
ckan.tracking_enabled = True
```

Default value: `False`

This controls if CKAN will track the site usage. For more info, read *Page View Tracking*.

10.4 Authorization Settings

More information about how authorization works in CKAN can be found the Authorization section.

10.4.1 ckan.auth.anon_create_dataset

Example:

```
ckan.auth.anon_create_dataset = False
```

Default value: `False`

Allow users to create datasets without registering and logging in.

10.4.2 ckan.auth.create_unowned_dataset

Example:

```
ckan.auth.create_unowned_dataset = False
```

Default value: `True`

Allow the creation of datasets not owned by any organization.

10.4.3 ckan.auth.create_dataset_if_not_in_organization

Example:

```
ckan.auth.create_dataset_if_not_in_organization = False
```

Default value: `True`

Allow users who are not members of any organization to create datasets, default: true. `create_unowned_dataset` must also be True, otherwise setting `create_dataset_if_not_in_organization` to True is meaningless.

10.4.4 ckan.auth.user_create_groups

Example:

```
ckan.auth.user_create_groups = False
```

Default value: `True`

Allow users to create groups.

10.4.5 ckan.auth.user_create_organizations

Example:

```
ckan.auth.user_create_organizations = False
```

Default value: `True`

Allow users to create organizations.

10.4.6 ckan.auth.user_delete_groups

Example:

```
ckan.auth.user_delete_groups = False
```

Default value: `True`

Allow users to delete groups.

10.4.7 ckan.auth.user_delete_organizations

Example:

```
ckan.auth.user_delete_organizations = False
```

Default value: `True`

Allow users to delete organizations.

10.4.8 ckan.auth.create_user_via_api

Example:

```
ckan.auth.create_user_via_api = False
```

Default value: `False`

Allow new user accounts to be created via the API.

10.5 Search Settings

10.5.1 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.

10.5.2 ckan.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.

10.5.3 solr_url

Example:

```
solr_url = http://solr.okfn.org:8983/solr/ckan-schema-2.0
```

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.

10.5.4 ckan.search.automatic_indexing

Example:

```
ckan.search.automatic_indexing = true
```

Default value: `true`

Make all changes immediately available via the search after editing or creating a dataset. Default is true. If for some reason you need the indexing to occur asynchronously, set this option to false.

Note: This is equivalent to explicitly load the `synchronous_search` plugin.

10.5.5 ckan.search.solr_commit

Example:

```
ckan.search.solr_commit = false
```

Default value: `true`

Make ckan commit changes solr after every dataset update change. Turn this to false if on solr 4.0 and you have automatic (soft)commits enabled to improve dataset update/create speed (however there may be a slight delay before dataset gets seen in results).

10.5.6 ckan.search.show_all_types

Example:

```
ckan.search.show_all_types = true
```

Default value: `false`

Controls whether the default search page (`/dataset`) should show only standard datasets or also custom dataset types.

10.5.7 search.facets.limits

Example:

```
search.facets.limits = 100
```

Default value: `50`

Sets the default number of searched facets returned in a query.

10.5.8 search.facets.default

Example:

```
search.facets.default = 10
```

Default number of facets shown in search results. Default 10.

10.5.9 ckan.extra_resource_fields

Example:

```
ckan.extra_resource_fields = alt_url
```

Default value: `None`

List of the extra resource fields that would be used when searching.

10.6 Plugins Settings

10.6.1 ckan.plugins

Example:

```
ckan.plugins = disqus datapreview googleanalytics follower
```

Default value: `stats text_preview recline_preview`

Specify which CKAN plugins are to be enabled.

Warning: If you specify a plugin but have not installed the code, CKAN will not start.

Format as a space-separated list of the plugin names. The plugin name is the key in the `[ckan.plugins]` section of the extension's `setup.py`. For more information on plugins and extensions, see Writing Extensions.

10.6.2 ckan.datastore.enabled

Example:

```
ckan.datastore.enabled = True
```

Default value: `False`

Controls if the Data API link will appear in Dataset's Resource page.

Note: This setting only applies to the legacy templates.

10.6.3 ckanext.stats.cache_enabled

Example:

```
ckanext.stats.cache_enabled = True
```

Default value: `True`

This controls if we'll use the 1 day cache for stats.

10.7 Front-End Settings

10.7.1 ckan.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.

10.7.2 ckan.site_description

Example:

```
ckan.site_description = The easy way to get, use and share data
```

Default value: (none)

This is for a description, or tag line for the site, as displayed in the header of the CKAN web interface.

10.7.3 ckan.site_intro_text

Example:

```
ckan.site_intro_text = Nice introductory paragraph about CKAN or the site in general.
```

Default value: (none)

This is for an introductory text used in the default template's index page.

10.7.4 ckan.site_logo

Example:

```
ckan.site_logo = /images/ckan_logo_fullname_long.png
```

Default value: (none)

This sets the logo used in the title bar.

10.7.5 ckan.site_about

Example:

```
ckan.site_about = A _community-driven_ catalogue of _open data_ for the Greenfield area.
```

Default value:

```
<p>CKAN is the world's leading open-source data portal platform.</p>

<p>CKAN is a complete out-of-the-box software solution that makes data
accessible and usable - by providing tools to streamline publishing, sharing,
finding and using data (including storage of data and provision of robust data
```

```
APIs). CKAN is aimed at data publishers (national and regional governments,
companies and organizations) wanting to make their data open and available.</p>

<p>CKAN is used by governments and user groups worldwide and powers a variety
of official and community data portals including portals for local, national
and international government, such as the UK's <a href="http://data.gov.uk">data.gov.uk</a>
and the European Union's <a href="http://publicdata.eu/">publicdata.eu</a>,
the Brazilian <a href="http://dados.gov.br/">dados.gov.br</a>, Dutch and
Netherland government portals, as well as city and municipal sites in the US,
UK, Argentina, Finland and elsewhere.</p>

<p>CKAN: <a href="http://ckan.org/">http://ckan.org/</a><br />
CKAN Tour: <a href="http://ckan.org/tour/">http://ckan.org/tour/</a><br />
Features overview: <a href="http://ckan.org/features/">http://ckan.org/features/</a></p>
```

Format tips:

- multiline strings can be used by indenting following lines
- the format is Markdown

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. For this reason, it's better to overload the snippet in `home/snippets/about_text.html`. For more information, see Theming.

10.7.6 ckan.main_css

Example:

```
ckan.main_css = /base/css/my-custom.css
```

Default value: `/base/css/main.css`

With this option, instead of using the default *main.css*, you can use your own.

10.7.7 ckan.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.

10.7.8 ckan.legacy_templates

Example:

```
ckan.legacy_templates = True
```

Default value: `False`

This controls if the legacy genshi templates are used.

Note: This is only for legacy code, and shouldn't be used anymore.

10.7.9 ckan.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.

10.7.10 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.

10.7.11 ckan.dataset.show_apps_ideas

ckan.dataset.show_apps_ideas:

```
ckan.dataset.show_apps_ideas = false
```

Default value: true

When set to false, or no, this setting will hide the 'Apps, Ideas, etc' tab on the package read page. If the value is not set, or is set to true or yes, then the tab will shown.

Note: This only applies to the legacy Genshi-based templates

10.7.12 ckan.preview.direct

Example:

```
ckan.preview.direct = png jpg jpeg gif
```

Default value: `png jpg jpeg gif`

Defines the resource formats which should be embedded directly in an `img` tag when previewing them.

10.7.13 ckan.preview.loadable

Example:

```
ckan.preview.loadable = html htm rdf+xml owl+xml xml n3 n-triples turtle plain atom rss txt
```

Default value: `html htm rdf+xml owl+xml xml n3 n-triples turtle plain atom rss txt`

Defines the resource formats which should be loaded directly in an `iframe` tag when previewing them if no Data Viewer can preview it.

10.7.14 ckan.dumps_url

If there is a page which allows you to download a dump of the entire catalogue then specify the URL here, so that it can be advertised in the web interface. For example:

```
ckan.dumps_url = http://ckan.net/dump/
```

For more information on using dumpfiles, see *db: Manage databases*.

10.7.15 ckan.dumps_format

If there is a page which allows you to download a dump of the entire catalogue then specify the format here, so that it can be advertised in the web interface. `dumps_format` is just a string for display. Example:

```
ckan.dumps_format = CSV/JSON
```

10.7.16 ckan.recaptcha.publickey

The public key for your Recaptcha account, for example:

```
ckan.recaptcha.publickey = 6Lc...-KLc
```

To get a Recaptcha account, sign up at: http://www.google.com/recaptcha

10.7.17 ckan.recaptcha.privatekey

The private key for your Recaptcha account, for example:

```
ckan.recaptcha.privatekey = 6Lc...-jP
```

Setting both *ckan.recaptcha.publickey* and *ckan.recaptcha.privatekey* adds captcha to the user registration form. This has been effective at preventing bots registering users and creating spam packages.

10.7.18 ckan.featured_groups

Example:

```
ckan.featured_groups = group_one group_two
```

Default Value: (empty)

Defines a list of group names or group ids. This setting is used to display groups and datasets from each group on the home page in the default templates (2 groups and 2 datasets for each group are displayed).

10.7.19 ckan.gravatar_default

Example:

```
ckan.gravatar_default = monsterid
```

Default value: `identicon`

This controls the default gravatar avatar, in case the user has none.

10.7.20 ckan.debug_supress_header

Example:

```
ckan.debug_supress_header = False
```

Default value: `False`

This configs if the debug information showing the controller and action receiving the request being is shown in the header.

Note: This info only shows if debug is set to True.

10.8 Theming Settings

10.8.1 ckan.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">
```

Note: This is only for legacy code, and shouldn't be used anymore.

10.8.2 ckan.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 -->
```

10.8.3 ckan.template_title_deliminater

Example:

```
ckan.template_title_deliminater = |
```

Default value: –

This sets the delimiter between the site's subtitle (if there's one) and its title, in HTML's `<title>`.

10.8.4 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.

Note: This is only for legacy code, and shouldn't be used anymore.

10.8.5 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.

Note: This is only for legacy code, and shouldn't be used anymore.

10.9 Storage Settings

10.9.1 ckan.storage.bucket

Example:

```
ckan.storage.bucket = ckan
```

Default value: `None`

This setting will change the bucket name for the uploaded files.

10.9.2 ckan.storage.key_prefix

Example:

```
ckan.storage.key_prefix = ckan-file/
```

Default value: `file/`

This setting will change the prefix for the uploaded files.

10.9.3 ckan.storage.max_content_length

Example:

```
ckan.storage.max_content_length = 500000
```

Default value: `50000000`

This defines the maximum content size, in bytes, for uploads.

10.9.4 ofs.impl

Example:

```
ofs.impl = pairtree
```

Default value: `None`

Defines the storage backend used by CKAN: `pairtree` for local storage, `s3` for Amazon S3 Cloud Storage or `google` for Google Cloud Storage. Note that each of these must be accompanied by the relevant settings for each backend described below.

10.9.5 ofs.storage_dir

Example:

```
ofs.storage_dir = /data/uploads/
```

Default value: `None`

Only used with the local storage backend. 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.

10.9.6 ofs.aws_access_key_id

Example:

```
ofs.aws_access_key_id = your_key_id_here
```

Default value: `None`

Only used with the Amazon S3 storage backend.

Todo

Expand

10.9.7 ofs.aws_secret_access_key

Example:

```
ofs.aws_secret_access_key = your_secret_access_key_here
```

Default value: `None`

Only used with the Amazon S3 storage backend.

Todo

Expand

10.9.8 ofs.gs_access_key_id

Example:

```
ofs.gs_access_key_id = your_key_id_here
```

Default value: `None`

Only used with the Google storage backend.

Todo

Expand

10.9.9 ofs.gs_secret_access_key

Example:

```
ofs.gs_secret_access_key = your_secret_access_key_here
```

Default value: `None`

Only used with the Google storage backend.

Todo

Expand

10.10 Activity Streams Settings

10.10.1 ckan.activity_streams_enabled

Example:

```
ckan.activity_streams_enabled = False
```

Default value: `True`

Turns on and off the activity streams used to track changes on datasets, groups, users, etc

10.10.2 ckan.activity_streams_email_notifications

Example:

```
ckan.activity_streams_email_notifications = False
```

Default value: `False`

Turns on and off the activity streams' email notifications. You'd also need to setup a cron job to send the emails. For more information, visit *Email Notifications*.

10.10.3 ckan.activity_list_limit

Example:

```
ckan.activity_list_limit = 31
```

Default value: `infinite`

This controls the number of activities to show in the Activity Stream. By default, it shows everything.

10.10.4 ckan.email_notifications_since

Example:

```
ckan.email_notifications_since = 2 days
```

Default value: `infinite`

Email notifications for events older than this time delta will not be sent. Accepted formats: '2 days', '14 days', '4:35:00' (hours, minutes, seconds), '7 days, 3:23:34', etc.

10.11 Feeds Settings

10.11.1 ckan.feeds.author_name

Example:

```
ckan.feeds.author_name = Michael Jackson
```

Default value: `(none)`

This controls the feed author's name. If unspecified, it'll use *ckan.site_id*.

10.11.2 ckan.feeds.author_link

Example:

```
ckan.feeds.author_link = http://okfn.org
```

Default value: `(none)`

This controls the feed author's link. If unspecified, it'll use *ckan.site_url*.

10.11.3 ckan.feeds.authority_name

Example:

```
ckan.feeds.authority_name = http://okfn.org
```

Default value: `(none)`

The domain name or email address of the default publisher of the feeds and elements. If unspecified, it'll use *ckan.site_url*.

10.11.4 ckan.feeds.date

Example:

```
ckan.feeds.date = 2012-03-22
```

Default value: `(none)`

A string representing the default date on which the authority_name is owned by the publisher of the feed.

10.12 Internationalisation Settings

10.12.1 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*.

10.12.2 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.

10.12.3 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.

10.12.4 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.

10.12.5 ckan.i18n_directory

Example:

```
ckan.i18n_directory = /opt/locales/i18n/
```

Default value: (none)

By default, the locales are searched for in the `ckan/i18n` directory. Use this option if you want to use another folder.

10.12.6 ckan.root_path

Example:

```
ckan.root_path = /my/custom/path/{{LANG}}/foo
```

Default value: (none)

By default, the URLs are formatted as `/some/url`, when using the default locale, or `/de/some/url` when using the "de" locale, for example. This lets you change this. You can use any path that you want, adding `{{LANG}}` where you want the locale code to go.

10.13 Form Settings

10.13.1 package_new_return_url

The URL to redirect the user to after they've submitted a new package form, example:

```
package_new_return_url = http://datadotgc.ca/new_dataset_complete?name=<NAME>
```

This is useful for integrating CKAN's new dataset form into a third-party interface, see Form Integration.

The `<NAME>` string is replaced with the name of the dataset created.

10.13.2 package_edit_return_url

The URL to redirect the user to after they've submitted an edit package form, example:

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

This is useful for integrating CKAN's edit dataset form into a third-party interface, see Form Integration.

The `<NAME>` string is replaced with the name of the dataset that was edited.

10.13.3 licenses_group_url

A url pointing to a JSON file containing a list of license objects. This list determines the licenses 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 licenses available from the http://licenses.opendefinition.org/licenses/groups/ckan.json.

More details about the license objects - including the license format and some example license 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
```

10.14 Email Settings

10.14.1 smtp.server

Example:

```
smtp.server = smtp.gmail.com:587
```

Default value: `None`

The SMTP server to connect to when sending emails with optional port.

10.14.2 smtp.starttls

Example:

```
smtp.starttls = True
```

Default value: `None`

Whether or not to use STARTTLS when connecting to the SMTP server.

10.14.3 smtp.user

Example:

```
smtp.user = your_username@gmail.com
```

Default value: None

The username used to authenticate with the SMTP server.

10.14.4 smtp.password

Example:

```
smtp.password = yourpass
```

Default value: None

The password used to authenticate with the SMTP server.

10.14.5 smtp.mail_from

Example:

```
smtp.mail_from = you@yourdomain.com
```

Default value: None

The email address that emails sent by CKAN will come from. Note that, if left blank, the SMTP server may insert its own.

10.14.6 email_to

Example:

```
email_to = you@yourdomain.com
```

Default value: None

This controls where the error messages will be sent to.

10.14.7 error_email_from

Example:

```
error_email_from = paste@localhost
```

Default value: None

This controls from which email the error messages will come from.

CHAPTER 11

Changelog

11.1 v2.1.5 2015-07-22

Bug fixes:

- Fix broken boolean validator (#2443)
- Key error on resource proxy (#2425)
- Ignore revision_id passed to resources (#2340)
- Add reset for reset_key on successful password change (#2379)

11.2 v2.1.4 2015-03-04

Bug fixes:

- Only link to http, https and ftp resource urls (#2085)
- Avoid private and deleted datasets on stats plugin (#1936)
- Fix tags count and group links in stats extension (#1649)
- Make resource_create auth work against package_update (#2037)
- Fix DataStore permissions check on startup (#1374)
- Fix datastore docs link (#2044)
- Fix resource extras getting lost on resource update (#2158)
- Clean up field names before rendering the Recline table (#2319)
- Don't "normalize" resource URL in recline view (#2324)
- Don't assume resource format is there on text preview (#2320)

11.3 v2.1.3 2014-10-15

Bug fixes:

- Organization image_url is not displayed in the dataset view. (#1934)
- i18n: Close a tag in French translation in Markdown syntax link (#1919)

- organization_list_for_user() fixes (#1918)
- Incorrect link in Organization snippet on dataset page (#1882)
- Prevent reading system tables on DataStore SQL search (#1871)
- Ensure that the DataStore is running on legacy mode when using PostgreSQL < 9.x (#1879)
- Edit user encoding error (#1436)
- Able to list private datasets via the API (#1580)
- Insecure content warning when running Recline under SSL (#1729)
- Add quotes to package ID in Solr query in _bulk_update_dataset to prevent Solr errors with custom dataset IDs. (#1853)
- Ordering a dataset listing loses the existing filters (#1791)
- Inserting empty arrays in JSON type fields in datastore fails (#1776)
- programatically log user in after registration (#1721)
- Deleted Users bug (#1668)
- Catch NotFound error in resources page (#1685)
- bulk_process page for non-existent organization throws Exception (#1682)
- Default search ordering on organization home page is broken (#1368)
- Term translations of organizations (#1274)
- Preview fails on private datastore resources (#1221)
- Strip whitespace from title in model dictize (#1228)

11.4 v2.1.2 2014-02-04

Bug fixes:

- Fix context for group/about setup_template_variables (#1433)
- Call setup_template_variables in group/org read, about and bulk_process (#1281)
- Remove repeated sort code in package_search (#1461)
- Ensure that check_access is called on activity_create (#1421)
- Fix visibility validator (#1188)
- Remove p.toolkit.auth_allow_anonymous_access as it is not available on 2.1.x (#1373)
- Add organization_revision_list to avoid exception on org history page (#1359)
- Fix activity and about organization pages (#1298)
- Show 404 instead of login page on user not found (#1068)
- Don't show Add Dataset button on org pages unless authorized (#1348)
- Fix datastore_search_sql authorization function (#1373)
- Fix extras deletion (#1449)
- Better word breaking on long words (#1398)
- Fix activity and about organization pages (#1298)

- Remove limit of number of arguments passed to `user add` command.
- Fix related_list logic function (#1384)
- Avoid UnicodeEncodeError on feeds when params contains non ascii characters

11.5 v2.1.1 2013-11-8

Bug fixes:

- Fix place-holder images on non-root locations (#1309)
- Fix errors on preview on non-root locations (#960)
- Don't accept invalid URLs in resource proxy (#1106)
- Make sure came_from url is local (#1039)
- Fix logout redirect in non-root locations (#1025)
- Wrong auth checks for sysadmins on package_create (#1184)
- Don't return private datasets on package_list (#1295)
- Stop tracking failing when no lang/encoding headers (#1192)
- Fix for paster db clean command getting frozen
- Fix organization not set when editing a dataset (#1199)
- Fix PDF previews (#1194)
- Fix preview failing on private datastore resources (#1221)

11.6 v2.1 2013-08-13

Note: This version requires a requirements upgrade on source installations

Note: This version requires a database upgrade

Note: This version does not require a Solr schema upgrade

Note: The `json_preview` plugin has been renamed to `text_preview` (see #266). If you are upgrading CKAN from a previous version you need to change the plugin name on your CKAN config file after upgrading to avoid a PluginNotFound exception.

Major:

- Bulk updates of datasets within organizations (delete, make public/private) (#278)
- Organizations and Groups search (#303)
- Generic text preview extension for JSON, XML and plain text files (#226)
- Improve consistency of the Action API (#473)
- IAuthenticator interface for plugging into authorization platforms (Work in progress) (#1007)
- New clearer dashboard with more information easier to access (#626)
- New `rebuild_fast` command to speed up reindex using multiple cores (#700)

- Complete restructure of the documentation, with updated sections on installation, upgrading, release process, etc and guidelines on how to write new documentation (#769 and multiple others)

Minor:

- Add group members page to templates (#844)
- Show search facets on organization page (#776)
- Changed default sort ordering (#869)
- More consistent display of buttons across pages (#890)
- History page ported to new templates (#368)
- More blocks to templates to allow furhter customization (#688)
- Improve imports from lib.helpers (#262)
- Add support for callback parameter on Action API (#414)
- Create site_user at startup (#952)
- Add warning before deleting an organization (#803)
- Remove flags from language selector (#822)
- Hide the Data API button when datastore is disabled (#752)
- Pin all requirements and separate minimal requirements in a separate file (#491, #1149)
- Better preview plugin selection (#1002)
- Add new functions to the plugins toolkit (#1015)
- Improve ExampleIDatasetFormPlugin (#2750)
- Extend h.sorted_extras() to do substitutions and auto clean keys (#440)
- Separate default database for development and testing (#517)
- More descriptive Solr exceptions when indexing (#674)
- Validate datastore input through schemas (#905)

Bug fixes:

- Fix 500 on password reset (#264)
- Fix exception when indexing a wrong date on a _date field (#267)
- Fix datastore permissions issues (#652)
- Placeholder images are not linked with h.url_for_static (#948)
- Explore dropdown menu is hidden behind other resources in IE (#915)
- Buttons interrupt file uploading (#902)
- Fix resource proxy encoding errors (#896)
- Enable streaming in resource proxy (#989)
- Fix cache_dir and beaker paths on deployment.ini_tmpl (#888)
- Fix multiple issues on create dataset form on IE (#881)
- Fix internal server error when adding member (#869)
- Fix license faceting (#853)

- Fix exception in dashboard (#830)
- Fix Google Analytics integration (#827)
- Fix ValueError when resource size is not an integer (#1009)
- Catch NotFound on new resource when package does not exist (#1010)
- Fix Celery configuration to allow overriding from config (#1027)
- came_from after login is validated to not redidirect to another site (#1039)
- And many, many more!

Deprecated and removed:

- The `json_preview` plugin has been replaced by a new `text_preview` one. Please update your config files if using it. (#226)

Known issues:

- Under certain authorization setups the forntend for the groups functionality may not work as expected (See #1176 #1175).

11.7 v2.0.7 2015-07-22

Bug fixes:

- Fix broken boolean validator (#2443)
- Key error on resource proxy (#2425)
- Ignore revision_id passed to resources (#2340)
- Add reset for reset_key on successful password change (#2379)

11.8 v2.0.6 2015-03-04

Bug fixes:

- Only link to http, https and ftp resource urls (#2085)
- Avoid private and deleted datasets on stats plugin (#1936)
- Fix tags count and group links in stats extension (#1649)
- Make resource_create auth work against package_update (#2037)
- Fix datastore docs link (#2044)
- Fix resource extras getting lost on resource update (#2158)
- Clean up field names before rendering the Recline table (#2319)
- Don't "normalize" resource URL in recline view (#2324)
- Don't assume resource format is there on text preview (#2320)

11.9 v2.0.5 2014-10-15

Bug fixes:

- organization_list_for_user() fixes (#1918)
- Incorrect link in Organization snippet on dataset page (#1882)
- Prevent reading system tables on DataStore SQL search (#1871)
- Ensure that the DataStore is running on legacy mode when using PostgreSQL < 9.x (#1879)
- Current date indexed on empty "*_date" fields (#1701)
- Able to list private datasets via the API (#1580)
- Insecure content warning when running Recline under SSL (#1729)
- Inserting empty arrays in JSON type fields in datastore fails (#1776)
- Deleted Users bug (#1668)

11.10 v2.0.4 2014-02-04

Bug fixes:

- Fix extras deletion (#1449)
- Better word breaking on long words (#1398)
- Fix activity and about organization pages (#1298)
- Show 404 instead of login page on user not found (#1068)
- Remove limit of number of arguments passed to `user add` command.
- Fix related_list logic function (#1384)

11.11 v2.0.3 2013-11-8

Bug fixes:

- Fix errors on preview on non-root locations (#960)
- Don't accept invalid URLs in resource proxy (#1106)
- Make sure came_from url is local (#1039)
- Fix logout redirect in non-root locations (#1025)
- Don't return private datasets on package_list (#1295)
- Stop tracking failing when no lang/encoding headers (#1192)
- Fix for paster db clean command getting frozen

11.12 v2.0.2 2013-08-13

Bug fixes:

- Fix markdown in group descriptions (#303)
- Fix resource proxy encoding errors (#896)
- Fix datastore exception on first run (#907)
- Enable streaming in resource proxy (#989)
- Fix in user search (#1024)
- Fix Celery configuration to allow overriding from config (#1027)
- Undefined function on organizations controller (#1036)
- Fix license not translated in orgs/groups (#1040)
- Fix link to documentation from the footer (#1062)
- Fix missing close breadcrumb tag in org templates (#1071)
- Fix recently_changed_packages_activity_stream function (#1159)
- Fix Recline map sidebar not showing in IE 7-8 (#1133)

11.13 v2.0.1 2013-06-11

Bug fixes:

- Use IDatasetForm schema for resource_update (#897)
- Fixes for CKAN being run on a non-root URL (#948, #913)
- Fix resource edit errors losing info (#580)
- Fix Czech translation (#900)
- Allow JSON filters for datastore_search on GET requests (#917)
- Install vdm from the Python Package Index (#764)
- Allow extra parameters on Solr queries (#739)
- Create site user at startup if it does not exist (#952)
- Fix modal popups positioning (#828)
- Fix wrong redirect on dataset form on IE (#963)

11.14 v2.0 2013-05-10

Note: Starting on v2.0, issue numbers with four digits refer to the old ticketing system at http://trac.ckan.org and the ones with three digits refer to GitHub issues. For example:

- #3020 is http://trac.ckan.org/ticket/3020
- #271 is https://github.com/okfn/ckan/issues/271

Some GitHub issues URLs will redirect to GitHub pull request pages.

Note: v2.0 is a huge release so the changes listed here are just the highlights. Bug fixes are not listed.

Note: This version requires a requirements upgrade on source installations

Note: This version requires a database upgrade

Note: This version requires a Solr schema upgrade

Organizations based authorization (see Authorization): CKAN's new "organizations" feature replaces the old authorization system with a new one based on publisher organizations. It replaces the "Publisher Profile and Workflow" feature from CKAN 1.X, any instances relying on it will need to be updated.

- New organization-based authorization and organization of datasets
- Supports private datasets
- Publisher workflow
- New authorization ini file options

New frontend (see Theming): CKAN's frontend has been completely redesigned, inside and out. There is a new default theme and the template engine has moved from Genshi to Jinja2. Any custom templates using Genshi will need to be updated, although there is a *ckan.legacy_templates* setting to aid in the migration.

- Block-based template inheritance
- Custom jinja tags: {% ckan_extends %}, {% snippet %} and {% url_for %} (#2502, #2503)
- CSS "primer" page for theme developers
- We're now using LESS for CSS
- Scalable font icons (#2563)
- Social sharing buttons (google plus, facebook, twitter) (this replaces the ckanext-social extension)
- Three-stage dataset creation form (#2501)
- New *paster front-end-build* command does everything needed to build the frontend for a production CKAN site (runs *paster less* to compile the css files, *paster minify* to minify the css and js files, etc.)

Plugins & Extensions:

- New plugins toolkit provides a stable set of utility and helper functions for CKAN plugins to depend on.
- The IDatasetForm plugin interface has been redesigned (note: this breaks backwards-compatibility with existing IDatasetForm plugins) (#649)
- Many IDatasetForm bugs were fixed
- New example extensions in core, and better documentation for the relevant plugin interfaces: example_itemplatehelpers (#447), example_idatasetform (#2750), hopefully more to come in 2.1!
- New IFacets interface that allows to modify the facets shown on various pages. (#400)
- The get_action() function now automatically adds 'model' and 'session' to the context dict (this saves on boiler-plate code, and means plugins don't have to import ckan.model in order to call get_action()) (#172)

Activity Streams, Following & User Dashboard:

- New visual design for activity streams (#2941)
- Group activity streams now include activities for changes to any of the group's datasets (#1664)

- Group activity streams now appear on group pages (previously they could only be retrieved via the api)
- Dataset activity streams now appear on dataset pages (previously they could only be retrieved via the api) (#3024)
- Users can now follow groups (previously you could only follow users or datasets) (#3005)
- Activity streams and following are also supported for organizations (#505)
- When you're logged into CKAN, you now get a notifications count in the top-right corner of the site, telling you how many new notifications you have on your dashboard. Clicking on the count takes you to your dashboard page to view your notifications. (#3009)
- Optionally, you can also receive notifications by email when you have new activities on your dashboard (#1635)
- Infinite scrolling of activity streams (if you scroll to the bottom of a an activity stream, CKAN will automatically load more activities) (#3018)
- Redesigned user dashboard (#3028):
  - New dropdown-menu enables you to filter you dashboard activity stream to show only activities from a particular user, dataset, group or organization that you're following
  - New sidebar shows previews and unfollow buttons (when the activity stream is filtered)
- New *ckan.activity_streams_enabled* config file setting allows you to disable the generation of activity streams (#654)

Data Preview:

- PDF files preview (#2203)
- JSON files preview
- HTML pages preview (in an iframe) (#2888)
- New plugin extension point that allows plugins to add custom data previews for different data types (#2961)
- Improved Recline Data Explorer previews (CSV files, Excel files..)
- Plain text files preview

API:

- The Action API is now CKAN's default API, and the API documentation has been rewritten (#357)

Other highlights:

- CKAN now has continuous integration testing at https://travis-ci.org/okfn/ckan/
- Dataset pages now have <link rel="alternate" type="application/rdf+xml" links in the HTML headers, allows linked-data tools to find CKAN's RDF rendering of a dataset's metadata (#413)
- CKAN's DataStore and Data API have been rewritten, and now use PostgreSQL instead of elasticsearch, so there's no need to install elasticsearch anymore (this feature was also back-ported to CKAN 1.8) (#2733)
- New Config page for sysadmins (/ckan-admin/config) enables sysadmins to set the site title, tag line, logo, the intro text shown on the front page, the about text shown on the /about page, select a theme, and add custom CSS (#2302, #2781)
- New *paster color* command for creating color schemes
- Fanstatic integration (#2371):
  - CKAN now uses Fanstatic to specify required static resource files (js, css..) for web pages
  - Enables each page to only include the static files that it needs, reducing page loads

  - Enables CKAN to use bundled and minified static files, further reducing page loads
  - CKAN's new *paster minify* command is used to create minified js and css files (#2950) (also see *paster front-end-build*)
- CKAN will now recognise common file format strings such as "application/json", "JSON", ".json" and "json" as a single file type "json" (#2416)
- CKAN now supports internalization of strings in javascript files, the new *paster trans* command is used to pull translatable strings out of javascript files (#2774, #2750)
- convert_to/from_extras have been fixed to not add quotes around strings (#2930)
- Updated CKAN coding standards (#3020) and CONTRIBUTING.rst file
- Built-in page view counting and 'popular' badges on datasets and resources There's also a paster command to export the tracking data to a csv file (#195)
- Updated CKAN Coding Standards and new CONTRIBUTING.rst file
- You can now change the sort ordering of datasets on the dataset search page

Deprecated and removed:

- The IGenshiStreamFilter plugin interface is deprecated (#271), use the ITemplateHelpers plugin interface instead
- The Model, Search and Util APIs are deprecated, use the Action API instead
- Removed restrict_template_vars config setting (#2257)
- Removed deprecated facet_title() template helper function, use get_facet_title() instead (#2257)
- Removed deprecated am_authorized() template helper function, use check_access() instead (#2257)
- Removed deprecated datetime_to_datestr() template helper function (#2257)

11.15 v1.8.2 2013-08-13

Bug fixes:

- Fix for using harvesters with organization setup
- Refactor for user update logic
- Tweak resources visibility query

11.16 v1.8.1 2013-05-10

Bug fixes:

- Fixed possible XSS vulnerability on html input (#703)
- Fix unicode template 500 error (#808)
- Fix error on related controller

11.17 v1.8 2012-10-19

Note: This version requires a requirements upgrade on source installations

Note: This version requires a database upgrade

Note: This version does not require a Solr schema upgrade

Major

- New 'follow' feature that allows logged in users to follow other users or datasets (#2304)
- New user dashboard that shows an activity stream of all the datasets and users you are following. Thanks to Sven R. Kunze for his work on this (#2305)
- New version of the Datastore. It has been completely rewritten to use PostgreSQL as backend, it is more stable and fast and supports SQL queries (#2733)
- Clean up and simplifyng of CKAN's dependencies and source install instructions. Ubuntu 12.04 is now supported for source installs (#2428,#2592)
- Big speed improvements when indexing datasets (#2788)
- New action API reference docs, which individually document each function and its arguments and return values (#2345)
- Updated translations, added Japanese and Korean translations

Minor

- Add source install upgrade docs (#2757)
- Mark more strings for translation (#2770)
- Allow sort ordering of dataset listings on group pages (#2842)
- Reenable simple search option (#2844)
- Editing organization removes all datasets (#2845)
- Accessibility enhancements on templates

Bug fixes

- Fix for relative url being used when doing file upload to local storage
- Various fixes on IGroupFrom (#2750)
- Fix group dataset sort (#2722)
- Fix adding existing datasets to organizations (#2843)
- Fix 500 error in related controller (#2856)
- Fix for non-open licenses appearing open
- Editing organization removes all datasets (#2845)

API changes and deprecation:

- Template helper functions are now restricted by default. By default only those helper functions listed in lib.helpers.__allowed_functions__ are available to templates. The full functions can still be made available by setting *ckan.restrict_template_vars = false* in your ini file. Only restricted functions will be allowed in future versions of CKAN.

- Deprecated functions related to the old faceting data structure have been removed: *helpers.py:facet_items()*, *facets.html:facet_sidebar()*, *facets.html:facet_list_items()*. Internal use of the old facets datastructure (attached to the context, *c.facets*) has been superseded by use of the improved facet data structure, *c.search_facets*. The old data structure is still available on *c.facets*, but is deprecated, and will be removed in future versions. (#2313)

11.18 v1.7.4 2013-08-13

Bug fixes:

- Refactor for user update logic
- Tweak resources visibility query

11.19 v1.7.3 2013-05-10

Bug fixes:

- Fixed possible XSS vulnerability on html input (#703)

11.20 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

11.21 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

11.22 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)

11.23 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)

11.24 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)

11.25 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 reorganization (#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)

11.26 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)

11.27 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)

11.28 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)

11.29 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)

11.30 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)

11.31 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)

11.32 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)

11.33 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)

11.34 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)

11.35 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)

11.36 v1.0.2 2010-08-27

- Bugfix: API returns error when creating package (#432)

11.37 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)

11.38 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.

11.39 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

11.40 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)

11.41 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

11.42 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

11.43 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

11.44 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.

11.45 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

11.46 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.

11.47 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

11.48 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

11.49 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