This Administration Guide covers how to set up and manage CKAN (Comprehensive Knowledge Archive Network) software.

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

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

Contents:
Option 1: Package Installation

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

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

- Using Ubuntu 10.04 directly.
- **Option A: Using VirtualBox.** This is suitable if you want to host your CKAN instance on a machine running any other OS.
- **Option B: Using Amazon EC2.** This is suitable if you want to host your CKAN instance in the cloud, on a ready-made Ubuntu OS.

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

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

### 1.1 Prepare your System

CKAN runs on Ubuntu 10.04. If you are already using Ubuntu 10.04, you can continue straight to **Run the Package Installer.**

However, if you’re not, you can either use VirtualBox to set up an Ubuntu VM on Windows, Linux, Macintosh and Solaris. Alternatively, you can use an Amazon EC2 instance.

#### 1.1.1 Option A: Using VirtualBox

This option is suitable if you want to install CKAN on a machine running an OS other than Ubuntu 10.04. **VirtualBox** lets you set up a virtual machine to run Ubuntu 10.04.

**Pre-requisites and Downloads**

First, check your machine meets the pre-requisites for **VirtualBox.** These include a fairly recent processor and some spare memory.

Then download the installation files.
• Download the VirtualBox installer.
• Download the Ubuntu image - make sure you choose Ubuntu 10.04.

Install VirtualBox

Note: This tutorial is for a Mac, but you can find instructions for installing VirtualBox on any OS in the VirtualBox Manual.

To install, double-click on the VirtualBox installer:

Click Continue to begin the installation process. Enter your password when required, and wait for the installation to finish.

Create Your Virtual Machine

Go to Applications and open VirtualBox, then click New:
Give your VM a name - we'll call ours `ubuntu_ckan`. Under **OS Type**, choose **Linux** and **Ubuntu 64-bit**.
Leave the memory size as 512MB, and choose **Create new hard disk**. This will open a new wizard:
You can leave the defaults unchanged here too - click **Continue**, and then **Done**, and **Done** again, to create a new VM. Next, choose your VM from the left-hand menu, and click **Start**:
This will open the First Run Wizard:
After clicking **Continue**, you’ll see **Select Installation Media**. This is where we need to tell our VM to boot from Ubuntu. Click on the file icon, and find your Ubuntu `.iso` file:
Click **Done**, wait for a few seconds, and you will see your Ubuntu VM booting.

**Set Up Ubuntu**

During boot, you will be asked if you want to try Ubuntu, or install it. Choose **Install Ubuntu**:
You can then follow the usual Ubuntu installation process.

After Ubuntu is installed, from the main menu, choose **System > Administration > Update Manager**. You’ll be asked if you want to install updates - say yes.

When all the updates have been downloaded and installed, you’ll be prompted to reboot Ubuntu.

At this point, you can proceed to Run the Package Installer.

### 1.1.2 Option B: Using Amazon EC2

If you prefer to run your CKAN package install in the cloud, you can use an Amazon EC2 instance, which is a fairly cheap and lightweight way to set up a server.

**Create an Amazon Account**

If you don’t already have an Amazon AWS account you’ll need to create one first. You can create an Amazon AWS account for EC2 here.

**Configure EC2**

Once you have an EC2 account, you’ll need to configure settings for your CKAN instance.

Start by logging into your Amazon AWS Console and click on the EC2 tab.

Select the region you want to run your CKAN instance in - the security group you set up is region-specific. In this tutorial, we use EU West, so it will be easier to follow if you do too.
Set up a Security Group

Click the Security Groups link in the My Resources section in the right-hand side of the dashboard.

Create a security group called `web_test` that gives access to ports 22, 80 and 5000 as shown below. This is needed so that you’ll actually be able to access your server once it is created. You can’t change these settings once the instance is running, so you need to do so now.
Create a Keypair

Now create a new keypair `ckan_test` to access your instance:

### 1.1. Prepare your System
When you click **Create**, your browser will prompt you to save a keypair called `ckan_test.pem`: 
In this tutorial, we save the keypair in `~/Downloads/ckan_test.pem`, but you should save it somewhere safe.

**Note:** If you plan to boot your EC2 instance from the command line, you need to remember where you’ve put this file.

---

### Boot the EC2 Image

CKAN requires Ubuntu 10.04 to run (either the i386 or amd64 architectures). Luckily Canonical provide a range of suitable images.

The cheapest EC2 instance is the micro one, but that isn’t very powerful, so in this tutorial, we’ll use the 32-bit small version.

We’re in `eu-west-1` and we’ll use an instance-only image (i.e. all the data will be lost when you shut it down) so we need the `ami-3693a542` AMI.

**Note:** There are more recent Ubuntu images at [http://cloud.ubuntu.com/ami/](http://cloud.ubuntu.com/ami/) but we need the older 10.04 LTS release.

At this point, you can either boot this image from the AWS console or launch it from the command line.

**Option 1: Boot the EC2 Image AMI via the AWS Console**

From the EC2 dashboard, choose **Launch instance >**:
Now work through the wizard as shown in the following screenshots.

In the first step search for `ami-3693a542` and select it from the results (it may take a few seconds for Amazon to find it).

**Warning:** No image other than `ami-3693a542` will work with CKAN.

You can keep the defaults for all of the following screens:
1.1. Prepare your System
Choose the `web_test` security group you created earlier:
Then finish the wizard:

Finally click the View your instances on the Instances page link:

1.1. Prepare your System
After a few seconds you’ll see your instance has booted. Now skip to Log in to the Instance.

Option 2: Boot the EC2 Image AMI from the Command Line

[You can skip this section if you’ve just booted from the AWS console and go straight to Log in to the Instance]

To boot from the command line you still need the same information but you enter it in one command. I’ll show you now.

Install The EC2 Tools Locally If you are on Linux, you can just install the tools like this:

```
sudo apt-get install ec2-ami-tools
sudo apt-get install ec2-api-tools
```

If you are on Windows or Mac you’ll need to download them from the Amazon website.

Once the software is installed you can use the files you’ve just downloaded to do create your instance.

Get Security Certificates Next click on the Account link, right at the top of the screen, and you’ll see this screen:
From this screen choose **Security Credentials** from the left hand side. Once the page has loaded scroll down and you’ll see the **Access Credentials** section. Click on the **X.509 Certificate** tab:
Access Credentials

There are three types of access credentials used to authenticate your requests to AWS services: (a) access keys, (b) X.509 certificates, and (c) key pairs. Each access credential type is explained below.

Use access keys to make secure REST or Query protocol requests to any AWS service API. We create one for you when your account is created — see your access key below.

<table>
<thead>
<tr>
<th>Created</th>
<th>Access Key ID</th>
<th>Secret Access Key</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>October 10, 2005</td>
<td>0K173W104AKNK122E2R2</td>
<td>Show</td>
<td>Active (Make Inactive)</td>
</tr>
</tbody>
</table>

Create a new Access Key

Since you created your AWS account before October 10, 2005, you have an AWS Subscription ID. If you have already built an AWS application using your Subscription ID, you may continue using your Subscription ID. If you are building a new AWS application, we encourage you to use your Access Key ID instead.

If you need to retrieve your Subscription ID for an existing application, click here.

For your protection, you should never share your secret access keys with anyone. In addition, industry best practice recommends frequent key rotation.

Learn more about Access Keys

Sign-In Credentials

To sign in to AWS web sites and applications, AWS requires your Amazon e-mail address and password. Additionally, it supports the AWS Multi-Factor Authentication option. Each sign-in credential is explained below.

Amazon E-mail Address and Password

To sign in to secure pages on the AWS web site, the AWS Management Console, the AWS Discussion Forums, and the AWS Premium Support site, you need to provide your Amazon e-mail address and password.

Email Address: james@myemail.org

Here you’ll be able to create an X.509 certificate and private key.

Tip: You can only have two X.509 certificates at any given time, so you might need to inactivate an old one first and then delete it before you are allowed to create a new one, as in the screenshot above.

Once you click the Create New Certificate link you get a popup which allows you to download the certificate and private key - do this. Once again, ours are in ~/Downloads, but you should save it somewhere safe.
Tip: Amazon will only give you a private key file once when you create it so although you can always go back to get a copy of the certificate, you can only get the private key once. Make sure you save it in a safe place.

You now have:

- Your private key (`pk-[ID].pem`)
- Your certificate file (`cert-[ID].pem`)
- Your new keypair (`ckan-test.pem`)

The private key and the certificate files have the same name in the ID part.

Create an Ubuntu Instance  Once the tools are installed, run this command:

```bash
e2-run-instances ami-3693a542 --instance-type m1.small --region eu-west-1 --group web_test --key ckan_test --private-key ~/Downloads/pk-[ID].pem --cert ~/Downloads/cert-[ID].pem
```

Note: The `--key` argument is the name of the keypair (`ckan_test`), not the certificate itself (`ckan_test.pem`).

Warning: Amazon charge you for a minimum of one hour usage, so you shouldn’t create and destroy lots of EC2 instances unless you want to be charged a lot.

Log in to the Instance

Once your instance has booted, you will need to find out its public DNS. Give it a second or two for the instance to load then browse to the running instance in the AWS console. If you tick your instance you’ll be able to find the public...
DNS by scrolling down to the bottom of the Description tag.

Here you can see that our public DNS is ec2-79-125-86-107.eu-west-1.compute.amazonaws.com. The private DNS only works from other EC2 instances so isn’t any use to us.

Once you’ve found your instance’s public DNS, ensure the key has the correct permissions:

```
chmod 0600 "ckan_test.pem"
```

You can then log in like this:

```
ssh -i ~/Downloads/ckan_test.pem ubuntu@ec2-46-51-149-132.eu-west-1.compute.amazonaws.com
```

The first time you connect you’ll see this, choose yes:

```
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added ‘ec2-79-125-86-107.eu-west-1.compute.amazonaws.com,79.125.86.107’ (RSA) to the list of known hosts.
```

When you log in you’ll see a welcome message. You can now proceed to Run the Package Installer.

**Note:** If this is a test install of CKAN, when you have finished using CKAN, you can shut down your EC2 instance through the AWS console.

**Warning:** Shutting down your EC2 instance will lose all your data. Also, Amazon charge you for a minimum usage of one hour, so don’t create and destroy lots of EC2 instances unless you want to be charged a lot!

### 1.2 Run the Package Installer

On your Ubuntu 10.04 system, open a terminal window and switch to the root user:
sudo -s

Install the CKAN packages as follows:

echo 'deb http://apt.okfn.org/ubuntu_ckan-std_dev lucid universe' > /etc/apt/sources.list.d/okfn.list
wget -qO- http://apt.okfn.org/packages.okfn.key | sudo apt-key add -
apt-get update
apt-get install ckan-std

Wait for the output to finish, then create your CKAN instance:

ckan-std-install

If you are using Amazon EC2, you will additionally need to set the hostname of your server. To do this, run the command below, replacing ec2-46-51-149-132.eu-west-1.compute.amazonaws.com with the public DNS of your EC2 instance. Leave the / at the end, as it is part of the sed command. Then restart Apache. You can skip this if installing on VirtualBox or a local server.

sudo sed -e "s/ServerAlias \(.\*\)/ServerAlias ec2-46-51-149-132.eu-west-1.compute.amazonaws.com/" \ -i /etc/apache2/sites-available/std.common
sudo /etc/init.d/apache2 restart

Finally visit your CKAN instance - either at your Amazon EC2 hostname, or at http://localhost. You’ll be redirected to the login screen because you won’t have set up any permissions yet, so the welcome screen will look something like this.

You can now proceed to Post-Installation Setup.
Option 2: Install from Source

This section describes how to install CKAN from source. This removes the requirement for Ubuntu 10.04 that exists with Option 1: Package Installation.

Warning: This option is more complex than Option 1: Package Installation, so we suggest only doing it this way if you plan to work on CKAN core, or have no access to Ubuntu 10.04 via any of the suggested methods.

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

2.1 Install the Source

These are instructions to get developing with CKAN.

Before you start, it may be worth checking CKAN has passed the auto build and tests.

1. Ensure the required packages are installed.

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

   ```
   sudo apt-get install build-essential libxml2-dev libxslt-dev
   sudo apt-get install wget mercurial postgresql libpq-dev git-core
   sudo apt-get install python-dev python-psycopg2 python-virtualenv
   sudo apt-get install subversion
   ```

   Otherwise, you should install these packages from source.

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mercurial</td>
<td>Source control</td>
</tr>
<tr>
<td>python</td>
<td>Python v2.5-2.7</td>
</tr>
<tr>
<td>postgresql</td>
<td>PostgreSQL database</td>
</tr>
<tr>
<td>libpq</td>
<td>PostgreSQL library</td>
</tr>
<tr>
<td>psycopg2</td>
<td>PostgreSQL python module</td>
</tr>
<tr>
<td>libxml2</td>
<td>XML library development files</td>
</tr>
<tr>
<td>libxslt</td>
<td>XSLT library development files</td>
</tr>
<tr>
<td>virtualenv</td>
<td>Python virtual environments</td>
</tr>
<tr>
<td>wget</td>
<td>Command line tool for downloading from the web</td>
</tr>
<tr>
<td>build-essential</td>
<td>Tools for building source code (or up-to-date Xcode on Mac)</td>
</tr>
<tr>
<td>git</td>
<td>Git source control (for getting MarkupSafe src)</td>
</tr>
<tr>
<td>subversion</td>
<td>Subversion source control (for pyutilib)</td>
</tr>
</tbody>
</table>

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

```
cd ~
virtualenv pyenv
```

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

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

```
easy_install pip
```

3. Activate your virtual environment.

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

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

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

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

An activated shell looks in your virtual environment first when choosing which commands to run. If you enter `python` now it will actually run `~/pyenv/bin/python` which is what you want.

4. Install CKAN code and required Python packages into the new environment.

First you’ll need to install CKAN. For the latest version run:

```
pip install --ignore-installed -e hg+http://bitbucket.org/okfn/ckan#egg=ckan
```

CKAN has a set of dependencies it requires which you should install too:

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

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

If you are using Ubuntu Lucid you can install the rest of the dependencies from the system versions like this:

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

If you are not using Ubuntu Lucid you’ll still need to install all the dependencies that would have been met in the `apt-get` command at the start. You can do so like this:

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

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

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

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

5. Setup a PostgreSQL database.
List existing databases:

```
psql -l
```

It is advisable to ensure that the encoding of databases is ‘UTF8’, or internationalisation may be a problem. Since changing the encoding of PostgreSQL may mean deleting existing databases, it is suggested that this is fixed before continuing with the CKAN install.

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

**Tip:** If you choose a database name, user or password which are different from those suggested below then you’ll need to update the configuration file you’ll create in the next step.

Here we choose `ckantest` as the database and `ckanuser` as the user:

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

It should prompt you for a new password for the CKAN data in the database. It is suggested you enter `pass` for the password.

Now create the database, which we’ll call `ckantest`:

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

6. Create a CKAN config file.

Make sure you are in an activated environment (see step 3) so that Python Paste and other modules are put on the python path (your command prompt will start with `(pyenv)` if you have) then change into the `ckan` directory which will have been created when you installed CKAN in step 4 and create the config file `development.ini` using Paste:

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

You can give your config file a different name but the tests will expect you to have used `development.ini` so it is strongly recommended you use this name, at least to start with.

If you used a different database name or password when creating the database in step 5 you’ll need to now edit `development.ini` and change the `sqlalchemy.url` line, filling in the database name, user and password you used.

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

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

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

**Caution:** Advanced users: If you are using CKAN’s fab file capability you currently need to create your config file as `pyenv/ckan.net.ini` so you will probably have ignored the advice about creating a `development.ini` file in the `pyenv/src/ckan` directory. This is fine but CKAN probably won’t be able to find your `who.ini` file. To fix this edit `pyenv/ckan.net.ini`, search for the line `who.config_file = %(here)s/who.ini` and change it to `who.config_file = who.ini`.

We are moving to a new deployment system where this incompatibility will be fixed.

7. Create database tables.

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

You should see **Initialising DB: SUCCESS**. If you are not in the `pyenv/src/ckan` directory or you don’t have an activated shell, the command will not work.

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

8. Create the cache directory.

   You need to create the Pylon’s cache directory specified by `cache_dir` in the config file.
   
   (from the `pyenv/src/ckan` directory):
   ```bash
   mkdir data
   ```

9. Run the CKAN webserver.

   NB If you’ve started a new shell, you’ll have to activate the environment again first - see step 3.
   
   (from the `pyenv/src/ckan` directory):
   ```bash
   paster serve development.ini
   ```

10. Point your web browser at: [http://127.0.0.1:5000/](http://127.0.0.1:5000/)

    The CKAN homepage should load.

    Finally, make sure that tests pass, as described in **Basic Tests**.

    You can now proceed to **Post-Installation Setup**.
After you have completed installation (from either package or source), follow this section for instructions on setting up an initial user, loading test data, and notes on deploying CKAN.

### 3.1 Create an Admin User

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

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

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

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

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

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

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

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

### 3.2 Load Test Data

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

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

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

### 3.3 Deployment

You may want to deploy your CKAN instance at this point, to share with others.
If you have installed CKAN from packages, then Apache and WSGI deployment scripts are already configured for you in standard locations.

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

CKAN has been successfully deployed by a variety of other methods including Apache reverse proxy + paster, nginx reverse proxy + paster, and nginx + uwsgi.

You can now proceed to Customization.
Customization

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

### 4.1 Site Name and Description

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

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

Open this file, and change the following options:

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

After you’ve edited these options, restart Apache:

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

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

### 4.2 More Advanced Customization

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

#### 4.2.1 Custom CSS and Templates

You can add custom CSS, templates, scripts, images etc to your site using the `extra_template_paths` and `extra_public_paths` options in the CKAN config file:

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

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

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

For example, if you set the following option in the CKAN config file:
extra_public_paths = /path/to/mypublicdir

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

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

Adding a New Logo

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

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

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

Adding a New Stylesheet

Lots of visual changes can be made simply by changing the stylesheet.

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

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

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

```html
<html xmlns="http://www.w3.org/1999/xhtml"
     xmlns:i18n="http://genshi.edgewall.org/i18n"
     xmlns:py="http://genshi.edgewall.org/"
     xmlns:xi="http://www.w3.org/2001/XInclude"
   py:strip="">
   <py:def function="optional_head">
     <link rel="stylesheet" href="${g.site_url}/css/mycss.css" />
   </py:def>
   <xi:include href="layout_base.html" />
</html>
```

Retheming the Site with Templates

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

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

Commonly modified templates are:

- layout_base.html - base customization layout template for whole site
- layout.html - empty by default
- home/index.html - the home page of the site
• *home/about.html* - the about page

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

**Note:** For more information on the syntax of the CKAN templates, refer to the Genshi documentation.
You can upload individual datasets through the CKAN front-end, but for importing datasets on masse, you have two choices:

- **Import Data with the CKAN API.** You can use the CKAN API to script import. To simplify matters, we offer provide standard loading scripts for Google Spreadsheets, CSV and Excel.

- **Import Data with the Harvester Extension.** The CKAN harvester extension provides web and command-line interfaces for larger import tasks.

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

**Note:** If loading your data requires scraping a web page regularly, you may find it best to write a scraper on Scraper-Wiki and combine this with either of the methods above.

### 5.1 Import Data with the CKAN API

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

#### 5.1.1 The Simplest Approach - ckanclient

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

You can install ckanclient with:

```
pip install ckanclient
```

Here is an example script to register a new package:

```python
import ckanclient
# Instantiate the CKAN client.
ckan = ckanclient.CkanClient(api_key=my_api_key, base_location="http://myckaninstance.com/api")
# Describe the package.
package_entity = {
    'name': my_package_name,
    'url': my_package_url,
    'download_url': my_package_download_url,
    'tags': my_package_keywords,
    'notes': my_package_long_description,
}
```
# Register the package.
ckan.package_register_post(package_entity)

## 5.1.2 Loader Scripts

‘Loader scripts’ provide a simple way to take any format metadata and bulk upload it to a remote CKAN instance. Essentially each set of loader scripts converts the dataset metadata to the standard ‘package’ format, and then loads it into CKAN.

Loader scripts are generally stored into the `ckanext` repository. To get a flavour of what loader scripts look like, take a look at the ONS scripts.

### Loader Scripts for CSV and Excel

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

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

### Loader Scripts for Google Spreadsheets

The `GoogleSpreadsheetReader` class (found in `ckanclient.loaders`) simplifies the process of loading data from Google Spreadsheets.

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

### Write Your Own Loader Script

```python
## this needs work ##

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

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

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

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

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

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

The CKAN harvester extension provides useful tools for more advanced data imports. These include a command-line interface and a web user interface for running harvesting jobs.

To use the harvester extension, derive from the base class of the harvester extension and then write a custom \_create_or_update\_package method for your data.

For more information on working with extensions, see Add Extensions.
Common CKAN Tasks

The majority of common CKAN administration tasks are carried out using the paster script. Paster is run on the command line on the server running CKAN. This section covers:

- **Understanding Paster.** Understanding paster syntax and getting help.
- **Common Tasks Using Paster.** How to carry out common CKAN admin tasks using paster.

### 6.1 Understanding Paster

The basic paster format is:

```sh
paster --plugin=ckan <ckan commands> --config=<config file>
```

For example, to initialise a database:

```sh
paster --plugin=ckan db init --config=/etc/ckan/std/std.ini
```

### 6.1.1 Getting Help on Paster

To get a full list of paster commands (i.e. including CKAN commands):

```sh
paster --plugin=ckan --help
```

And to get more detailed help on each command (e.g. on db):

```sh
paster --plugin=ckan --help db
```

### 6.1.2 Position of Paster Parameters

The position of paster parameters matters.

--plugin is a parameter to paster, so needs to come before the CKAN command. To do this, the first parameter to paster is normally --plugin=ckan.

**Note:** The default value for --plugin is setup.py in the current directory. If you are running paster from the directory where CKAN’s setup.py file is located, you don’t need to specify the plugin parameter..
Meanwhile, `--config` is a parameter to CKAN, so needs to come after the CKAN command. This specifies the CKAN config file for the instance you want to use, e.g. `--config=/etc/ckan/std/std.ini`

**Note:** The default value for `--config` is `development.ini` in the current directory. If you are running a package install of CKAN (as described in Option 1: Package Installation), you should explicitly specify `std.ini`.

The position of the CKAN command itself is less important, as longs as it follows `--plugin`. For example, both the following commands have the same effect:

```
paster --plugin=ckan db --config=development.ini init
paster --plugin=ckan db init --config=development.ini
```

### 6.1.3 Running a Paster Shell

If you want to run a “paster shell”, which can be useful for development, then the plugin is pylons. e.g. `paster --plugin=pylons shell`.

Often you will want to run this as the same user as the web application, to ensure log files are written as the same user. And you’ll also want to specify a config file (note that this is not specified using the `--config` parameter, but simply as the final argument). For example:

```
sudo -u www-data paster --plugin=pylons shell std.ini
```

### 6.2 Common Tasks Using Paster

The following tasks are supported by paster.

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create-test-data</td>
<td>Create test data in the database.</td>
</tr>
<tr>
<td>db</td>
<td>Perform various tasks on the database.</td>
</tr>
<tr>
<td>ratings</td>
<td>Manage the ratings stored in the db.</td>
</tr>
<tr>
<td>rights</td>
<td>Commands relating to per-object and system-wide access rights.</td>
</tr>
<tr>
<td>roles</td>
<td>Commands relating to roles and actions.</td>
</tr>
<tr>
<td>search-index</td>
<td>Creates a search index for all packages</td>
</tr>
<tr>
<td>sysadmin</td>
<td>Gives sysadmin rights to a named user</td>
</tr>
<tr>
<td>user</td>
<td>Manage users</td>
</tr>
</tbody>
</table>

For the full list of tasks supported by paster, you can run:

```
paster --plugin=ckan --help
```

#### 6.2.1 create-test-data: Create test data

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

#### 6.2.2 db: Manage databases

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

Before you can run CKAN for the first time, you need to run “db init” to create the tables in the database and the default authorization settings:

```
paster --plugin=ckan db init --config=/etc/ckan/std/std.ini
```

If you forget to do this then CKAN won’t serve requests and you will see errors such as this in the logs:

```
ProgrammingError: (ProgrammingError) relation "user" does not exist
```

Cleaning

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

```
paster --plugin=ckan db clean --config=/etc/ckan/std/std.ini
```

The next logical step from this point is to do a “db init” step before starting CKAN again.

Upgrade migration

When you upgrade CKAN software by any method other than the package update described in Upgrade CKAN, before you restart it, you should run ‘db upgrade’, which will do any necessary migrations to the database tables:

```
paster --plugin=ckan db upgrade --config=/etc/ckan/std/std.ini
```

Creating dump files

For information on using db to create dumpfiles, see Database Dumps.

6.2.3 ratings: Manage package 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
```

6.2.4 rights: Set user permissions

Sets the authorization roles of a specific user on a given object within the system.

For example, to give the user named ‘bar’ the ‘admin’ role on the package ‘foo’:

```
paster --plugin=ckan rights make bar admin package:foo --config=/etc/ckan/std/std.ini
```

To list all the rights currently specified:

```
paster --plugin=ckan rights list --config=/etc/ckan/std/std.ini
```

For more information and examples, see Set and Manage Permissions.
6.2.5 roles: Manage system-wide permissions

This important command gives you fine-grained control over CKAN permissions, by listing and modifying the assignment of actions to roles.

The roles command has its own section: see Set and Manage Permissions.

6.2.6 search-index: Rebuild search index

Rebuilds the search index defined in the search_backend config setting. This is useful to prevent search indexes from getting out of sync with the main database.

For example:

```
paster --plugin=ckan search-index --config=/etc/ckan/std/std.ini
```

6.2.7 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
```

6.2.8 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 delete admin --config=/etc/ckan/std/std.ini
```
CKAN implements a fine-grained role-based access control system. This section describes:

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

## 7.1 Overview

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

In more detail, these concepts are as follows:

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

### 7.1.1 Objects

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

- A package is the basic CKAN concept of metadata about a dataset.
- A group of packages can be set up to specify which users have permission to add or remove packages from the group.
- Users can be assigned to authorization groups, to increase flexibility. Instead of specifying the privileges of specific users on a package or group, you can also specify a set of users that share the same rights. To do that, an authorization group can be set up and users can be added to it. Authorization groups are both the object of authorization (i.e. one can have several roles with regards to an authorization group, such as being allowed to


read or edit it) and the subject of authorization (i.e. they can be assigned roles on other objects which will apply to their members, such as the group having edit rights on a particular group).

- Finally, the system object is special, serving as an object for assignments that do not relate to a specific object. For example, creating a package cannot be linked to a specific package instance, and is therefore a operation.

### 7.1.2 Actions

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

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

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

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

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

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

### 7.1.3 Roles

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

Currently there are four basic roles:

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

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

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

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

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

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

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

7.2 Default Permissions

CKAN ships with the following default permissions:

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

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

7.3 Managing Permissions

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

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

7.3.1 Command-line authorization management

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

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

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

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

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

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

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

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

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

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

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

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

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

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

To list all role assignments:

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

To remove the ‘create-package’ action from the ‘editor’ role:

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

And to re-assign ‘create-package’ to the ‘editor’ role:

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

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

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

7.4 Publisher Mode

Although the Data Hub prefers the Wikipedia-style model of allowing anyone to add and improve metadata, some CKAN instances prefer to operate in ‘publisher mode’.

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

The key features are:

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

To operate in this mode:

1. First, remove the general public’s rights to create and edit packages:

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

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

   paster rights remove bob admin package:all

3. Change the default rights for newly created packages. Do this by using these values in your config file (see Reference: CKAN Configuration Options):
ckan.default_roles.Package = {"visitor": ["reader"], "logged_in": ["reader"]}
ckan.default_roles.Group = {"visitor": ["reader"], "logged_in": ["reader"]}
ckan.default_roles.System = {"visitor": ["reader"], "logged_in": ["reader"]}
ckan.default_roles.AuthorizationGroup = {"visitor": ["reader"], "logged_in": ["reader"]}

Note you can also restrict package edits by a user’s authorization group.
Prepare to Use Extensions

If you are running a package installation of CKAN, before you start using and testing extensions (described in Add Extensions) you need to prepare your system.

Firstly, you’ll need to set up and enter a virtual Python environment, as follows:

```
sudo apt-get install virtualenv python-pip mercurial
virtualenv /home/ubuntu/pyenv
. /home/ubuntu/pyenv/bin/activate
```

Then, you need to install the CKAN source into your virtual environment. You can install CKAN like this:

```
pip install -e hg+http://bitbucket.org/okfn/ckan#egg=ckan
```

Your new CKAN developer install will be running on http://localhost:5000/

When you start using extensions, you should install any of the developer versions of the CKAN extensions you want to work on like this (using the appropriate URL):

```
pip install -e hg+http://bitbucket.org/okfn/<dependency-name>@<version>#egg=<egg-name>
```

The dependency you’ve installed will appear in /home/ubuntu/pyenv/src/ where you can work on it.

After working on extensions, you should make sure that your deployment passes the tests, as described in Testing for Developers.
Add Extensions

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

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

**Warning:** This is an advanced topic. At the moment, you need to have prepared your system to work with extensions, as described in *Prepare to Use Extensions*. We are working to make the most popular extensions more easily available as Debian packages.

### 9.1 Finding Extensions

Many CKAN extensions are listed on the CKAN wiki’s List of Extensions. All CKAN extensions can be found at OKFN’s bitbucket page, prefaced with ckanext-.

Some popular extensions include:

- **ckanext-admin**: Admin web interface for CKAN.
- **ckanext-apps**: Apps and ideas catalogue extension for CKAN.
- **ckanext-deliverance**: Extends CKAN to use the Deliverance HTTP proxy, which can request and render web pages from an external site (e.g. a CMS like Drupal or Wordpress).
- **ckanext-disqus**: Allows users to comment on package pages with Disqus.
- **ckanext-follower**: Allow users to follow packages.
- **ckanext-googleanalytics**: Integrates Google Analytics data into CKAN. Gives download stats on package pages, list of most popular packages, etc.
- **ckanext-qa**: Provides link checker, 5 stars of openness and other Quality Assurance features.
- **ckanext-rdf**: Consolidated handling of RDF export and import for CKAN.
- **ckanext-stats**: Statistics (and visuals) about the datasets in a CKAN instance.
- **ckanext-wordpresser**: CKAN plugin / WSGI middleware for combining CKAN with a Wordpress site.
9.2 Installing an Extension

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

1. First, ensure you are working within your virtualenv (see Prepare to Use Extensions if you are not sure what this means):

   . /home/ubuntu/pyenv/bin/activate

2. Install the extension package code using pip.

   For example, to install the Disqus extension, which allows users to comment on datasets:

   pip install -E ~/var/src/ckan.net/pyenv hg+http://bitbucket.org/okfn/ckanext-disqus

   The -E parameter is for your CKAN Python environment (e.g. ~/var/src/ckan.net/pyenv).
   Prefix the source URL with the repo type (hg+ for Mercurial, git+ for Git).
   The dependency you’ve installed will appear in the src/ directory under your Python environment.

3. Add the names of any plugin implementations the extension uses to the CKAN config file. You can find these in the plugin’s setup.py file under [ckan.plugins].

   The config plugins variable is in the ‘[app:main]’ section under ‘ckan.plugins’. e.g.:

   [app:main]
   ckan.plugins = disqus

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

   ckan.plugins = disqus amqp myplugin

4. If necessary, restart WSGI, which usually means restarting Apache:

   sudo /etc/init.d/apache2 restart

Your extension should now be installed.
If you want to extend CKAN core functionality, the best way to do so is by writing extensions.

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

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

**Warning:** This is an advanced topic. At the moment, you need to have prepared your system to work with extensions, as described in Prepare to Use Extensions. We are working to make the most popular extensions more easily available as Debian packages.

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

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

```bash
$ python
>>> import ckanext.queue
```

Individual CKAN extensions may implement one or more *plugin interfaces* or *workers* to provide their functionality. You’ll learn about these later on.

10.1.1 Create Your Own Extension

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

```bash
(pyenv)$ paster create -t ckanext ckanext-myname
```

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

```
Selected and implied templates:

    ckan#ckanext  CKAN extension project template

Variables:

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

Once you’ve run this you should find your extension is already set up in your virtual environment so you can import it:
(pyenv)$ python
Python 2.6.6 (r266:84292, Oct 6 2010, 16:19:55)
[ GCC 4.1.2 20080704 (Red Hat 4.1.2-48)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import ckanext.myname

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

Testing 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
nositests ckanext/dgu/tests --ckan
cositests ckanext/dgu/tests --ckan --with-pylons=test-core.ini

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

cd /home/ubuntu/pyenv/src/ckan
. .../bin/activate
paster make-config ckan development.ini

Then make any changes to the development.ini file that you need before continuing:
paster db upgrade	paster serve --reload

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

Finally, if you write a CKAN, extension you may well want to publish it so others can use it too. See the Publishing your extension section below for details.

### 10.2 Plugins

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

#### 10.2.1 Summary

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

1. The CKAN core contains various *plugin interfaces*, each specifying a set of methods where plugins may hook into the software. For example a plugin wanting to hook into the SQLAlchemy mapping layer would need to implement the `IMapperExtension` interface.

2. A plugin is a class that derives from `ckan.plugins.Plugin` or more commonly `SingletonPlugin`. It must also implement one of the plugin interfaces exposed in `ckan.plugins.interfaces`. The choice interface determines the functionality the plugin is expected to provide.

3. Plugin objects must be registered as setuptools entry points. The `ckan.plugins` configuration directive is searched for names of plugin entry points to load and activate.

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

**IMapper**

- Listens and react to every database change

**IRoutes and IController**

- Provide an implementation to handle a particular URL

**IGenshiStreamFilter**

- Intercept template rendering to modify the output

**ISession**

**IDomainObjectModification**

**IGroupController**

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

**IConfigurable** Pass configuration to plugins and extensions

**IAuthorizer** Allows customisation of the default Authorization behaviour

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

**Note:** The existing ‘IRoutesExtension’, ‘IMapperExtension’ and ‘ISessionExtension’ should be renamed in the code to not have the word ‘Extension’ in their names.
10.2.2 An Example

Plugin interfaces are basically just Python classes where each method is a hook which allows a plugin that uses the interface to be notified when it is called.

As an example, let’s look at a plugin which gets configuration options from a config file and is called each time a template is rendered in order to add some HTML to the page.

**Tip:** This example is based on real code used to implement the ckanext-disqus plugin to add commenting to packages. You can see the latest version of this code at http://bitbucket.org/okfn/ckanext-disqus/src/tip/ckanext/plugins/disqus/__init__.py.

First we set up logging and some helpers we’ll need from Genshi to transfer the stream:

```python
import logging
log = logging.getLogger(__name__)

import html
from genshi.core import TEXT
from genshi.input import HTML
from genshi.filters import Transformer
```

Then we import the CKAN plugin code:

```python
from ckan.plugins.core import SingletonPlugin, implements
from ckan.plugins.interfaces import IConfigurable, IGenshiStreamFilter
```

In this case we are implementing both the IConfigurable and IGenshiStreamFilter plugin interfaces in our plugin class. The IConfigurable plugin interface defines a configure() method which will be is called on out plugin to let it know about configuration options. The IGenshiStreamFilter plugin interface defines a filter() method which will be called on the plugin to give it the opportunity to change the template before the HTML is returned to the browser.

Let’s have a look at the code:

```python
class Disqus(SingletonPlugin):
    
    """
    Insert javascript fragments into package pages and the home page to
    allow users to view and create comments on any package.
    """
    implements(IConfigurable)
    implements(IGenshiStreamFilter)

    def configure(self, config):
        """
        Called upon CKAN setup, will pass current configuration dict
to the plugin to read custom options.
        """
        self.disqus_name = config.get('disqus.name', None)
        if self.disqus_name is None:
            log.warn("No disqus forum name is set. Please set \n            'disqus.name' in your .ini!")
            self.disqus_name = 'ckan'

    def filter(self, stream):
        """
        Required to implement IGenshiStreamFilter; will apply some HTML
        transformations to the page currently rendered. Depends on Pylons
```
global objects, how can this be fixed without obscuring the interface?

```python
from pylons import request, tmpl_context as c
routes = request.environ.get('pylons.routes_dict')

if routes.get('controller') == 'package' and routes.get('action') == 'read' and c.pkg.id:
    data = {'name': self.disqus_name,
            'identifier': 'pkg-' + c.pkg.id}
    stream = stream | Transformer('body')
        .append(HTML(html.BOTTOM_CODE % data))
    stream = stream | Transformer('body/div[@id="comments"]')
        .append(HTML(html.COMMENT_CODE % data))

if routes.get('controller') == 'home' and routes.get('action') == 'index':
    data = {'name': self.disqus_name}
    stream = stream | Transformer('body/div[@id="main"]//ul[@class="xoxo"]')
        .append(HTML(html.LATEST_CODE % data))

return stream
```

Notice that the Disqus class explicitly states that it implements IConfigurable and IGenershiStreamFilter with these two lines:

```python
implements(IConfigurable)
implements(IGenishStreamFilter)
```

Also notice that Disqus inherits from SingletonPlugin. This means that only one instance of the plugin is needed to provide the service. There is also a Plugin class for occasions where you need multiple instances.

By carefully choosing the plugin interfaces your plugin uses you can hook into lots of parts of CKAN. Later on you’ll see how to write your own plugin interfaces to define your own “hooks”. Before we can use the Disqus plugin there is one more thing to do: add it to the extension and set an entry point.

### 10.2.3 Setting the Entry Point

Imagine the code above was saved into a file named disqus.py in the ckanext-myname/ckanext/myname directory of the extension that was created earlier by the paster create -t ckanext ckanext-myname command.

At this point CKAN still doesn’t know where to find your plugin, even though the module is installed. To find the plugin it looks up an entry point. An entry point is just a feature of setuptools that links a string in the form `package_name.entry_point_name` to a particular object in Python code.

**Tip:** If you are interested in reading a tutorial about entry points see:

- [http://reinout.vanrees.org/weblog/2010/01/06/zest-releaser-entry-points.html](http://reinout.vanrees.org/weblog/2010/01/06/zest-releaser-entry-points.html)

CKAN finds plugins by searching for entry points in the group `ckan.plugin`.

Entry points are defined in a package’s `setup.py` file. If you look in the `setup.py` file for the ckanext-myname extension you’ll see these lines commented out towards the end:
The entry point will be called without any parameters and must return an instance of `ckan.plugins.Plugin`. To enable the Disqus plugin uncomment the bottom line and change it to this:

```python
disqus_example=ckanext.myname:Disqus
```

Any time you change the `setup.py` file you will need to run one of these two commands again before the change will take effect:

```bash
python setup.py develop
python setup.py egg_info
```

With your entry point in place and installed you can now add the extension to your CKAN config as described earlier. To add our example Disqus plugin you would change your `~/var/src/ckan.net/ckan.net.ini` config file like this:

```ini
[app:main]
ckan.plugins = disqus_example
```

Note that the name of the plugin implementation that you give to `ckan.plugins` is always the same as the name of the entry point you’ve defined in the `setup.py` file. It is therefore important that you don’t choose an entry point name that is being used by in any existing extension to refer to its plugins.

The same extension can have multiple different plugins, all implementing different interfaces to provide useful functionality. For each new plugin your extension implements you need to add another entry point (remembering to re-run `python setup.py develop` if needed). Your users will then need to add the new entry point name to their `ckan.plugins` config option too.

### 10.2.4 Writing a Database Plugin

You’ve seen how to use `IConfigurable` and `IGenshiStreamFilter`. Here’s another example which implements `IMapperExtension` to log messages after any record is inserted into the database.

```python
from logging import getLogger
from ckan.plugins import implements, SingletonPlugin
from ckan.plugins import IMapperExtension

log = getLogger(__name__)

class InsertLoggerPlugin(SingletonPlugin):
    """
    Emit a log line when objects are inserted into the database
    """

    implements(IMapperExtension, inherit=True)

    def after_insert(mapper, connection, instance):
        log.info('Object %r was inserted', instance)
```

10.2. Plugins 61
10.2.5 Authorization Group Plugins

If you are writing an authorization group plugin you might like to use the code at this URL as a basis:
http://bitbucket.org/okfn/ckanextiati/src/tip/ckanext/iati/authz.py

10.2.6 Publishing Your Extension

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

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

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

    python setup.py register
    python setup.py sdist upload

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

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

10.2.7 Writing a Plugin Interface

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

Suppose you have a class such as this:

```python
class DataInput(object):
    def accept_new_data(self, data):
        self.data = data
```

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

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

```python
class IDataMunger(Interface):
    def munge(self, data):
        return data
```

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

```python
from ckan.plugins import PluginImplementations, IDataMunger
class DataInput(object):
    plugins = PluginImplementations(IDataMunger)
    def accept_new_data(self, data):
        for plugin in self.plugins:
            data = plugin.munge(data)
        self.data = data
```

Any registered plugins that implement IDataMunger will then be available in your class via self.plugin.
See the pyutilib documentation for more information on creating interfaces and plugins. However, be aware that pyutilib uses slightly different terminology. It calls `PluginImplementations ExtensionPoint` and it calls instances of a plugin object a `service`.

### 10.2.8 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:

```python
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:

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

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

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

### 10.2.9 Ordering of Extensions

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

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

1. System plugins (in setup.py under `ckan.system_plugins`).
2. In order of the plugins specified in the config file: `plugins =`.
3. If more than one module has a plugin with the same name specified in the config, then all those are loaded, in the order the modules appear in `sys.path`. 
The order that a plugins are run in, for example the order that IRoutes extensions have their `before_map` method run, is alphabetical by the plugin class.

e.g. here is the order for these four extensions: `<Plugin DguInventoryPlugin>, <Plugin FormApiPlugin>, <Plugin StatsPlugin>, <Plugin WalesThemePlugin>`
(This alphabetical ordering is done by `pyutilib.component.core:ExtensionPoint.extensions()`)
Next download and install RabbitMQ:

```
wget http://www.rabbitmq.com/releases/rabbitmq-server/v2.2.0/rabbitmq-server-2.2.0-1.noarch.rpm
rpm -Uhv --no-deps rabbitmq-server-2.2.0-1.noarch.rpm
```

Finally edit the `/etc/init.d/rabbitmq-server` script so that it uses the correct path for your Erlang install. Change this line:

```
PATH=/sbin:/usr/sbin:/bin:/usr/bin
```

To:

```
PATH=/sbin:/usr/sbin:/bin:/usr/bin:/opt/erlang_R14B/bin:/opt/erlang_R14B/lib
```

You can start it like this:

```
/etc/init.d/rabbitmq-server start
```

**Ubuntu**

Just run:

```
sudo apt-get install rabbitmq-server
```

### 10.3.2 Working Directly with Carrot

As you learned earlier, CKAN uses carrot with the `pyamqplib` backend. Carrot is well-documented at this URL:

http://ask.github.com/carrot/introduction.html

Before you learn how to use the tools that `ckanext-queue` uses to work with the queue, it is instructive to see a simple example that uses carrot directly.

Save this as `publisher.py`:

```python
from carrot.connection import BrokerConnection
conn = BrokerConnection(
    hostname="localhost",
    port=5672,
    userid="guest",
    password="guest",
    virtual_host="/",
)

from carrot.messaging import Publisher
publisher = Publisher(
    connection=conn,
    exchange='local',
    routing_key='*',
)
publisher.close()
```

Now save this as `consumer.py`:

```python
from carrot.connection import BrokerConnection
conn = BrokerConnection(
    hostname="localhost",
    port=5672,
    userid="guest",
    password="guest",
    virtual_host="/",
)
```
from carrot.messaging import Consumer
consumer = Consumer(
    connection=conn,
    queue='local.workerchain',
    exchange='local',
    routing_key='*',
)

def import_feed_callback(message_data, message):
    feed_url = message_data['import_feed']
    print("Got message for: %s" % feed_url)
    # something importing this feed url
    # import_feed(feed_url)
    message.ack()
consumer.register_callback(import_feed_callback)
# Go into the consumer loop.
consumer.wait()

You’ll notice that both examples set up a connection to the same AMQP server with the same settings, in this case running on localhost. These also happen to be the settings that (at the time of writing) ckanext-queue uses by default if you don’t specify other configuration settings.

Make sure you have ckanext-queue installed (so that carrot and its dependencies are installed too) then start the consumer:

python consumer.py

In a different console run the publisher:

python publisher.py

The publisher will quickly exit but if you now switch back to the consumer you’ll see the message was sent to the queue and the consumer received it printing this message:


10.3.3 Working with CKANext Queue

Rather than working with carrot publishers and consumers directly, ckanext-queue provides two useful Python objects to help you:

ckanext.queue.connection.get_publisher(config)

This returns a Publisher instance which has a send() method for adding an item to the queue.

The config object is the same as pylons.config. If you are writing a standalone script, you can obtain a config object from a config file with code similar to this, adjusting the relative_to option as necessary:

from paste.deploy import appconfig
config = appconfig('config:development.ini', relative_to='pyenv/src/ckan')

ckanext.queue.worker.Worker
This is a base class which you can inherit from. You can override its `consume()` method to asynchronously pull items from the queue to do useful things.

**Note:** To use the queue extension you don’t need to implement any new plugin interfaces, you just need to use the `get_publisher(config).send()` method and the `Worker` class. Of course your own extension might use plugins to hook into other parts of CKAN to get information to put or retrieve from the queue.

The worker implementation runs outside the CKAN server process, interacting directly with both the AMQP queue and the CKAN API (to get CKAN data). The `Worker` class therefore subclasses both the `carrot.Consumer` class and the `ckanclient.CkanClient` class so that your workers can make calls to the running CKAN server via its API.

### Writing a Publisher

Here’s a simple publisher. Save it as `publish_on_queue.py`:

```python
from ckanext.queue import connection
from paste.deploy import appconfig
import logging
logging.basicConfig(level=logging.DEBUG)
config = appconfig('config:ckan.ini', relative_to='.')
publisher = connection.get_publisher(config)
publisher.send({'import_feed': 'http://cnn.com/rss/edition.rss'})
publisher.close()
print "Sent!"
```

Note that this requires a `ckan.ini` file relative to the current working directory to run. Here’s what a sample file will look like:

```
[app:main]
use = egg:ckan
ckan.site_id = local
queue.port = 5672
queue.user_id = guest
queue.password = guest
queue.hostname = localhost
queue.virtual_host = /
```

The idea here is that publishers you write will be able to use the same settings as CKAN itself would. In the next section you’ll see how these same options and more to a standard CKAN install config file to enable CKAN to use the RabbitMQ queue.

With the `consumer.py` script still running, execute the new script:

```
python publish_on_queue.py
```

You’ll see that once again the consumer picks up the message.

### Writing a Worker

Now let’s replace the consumer with a worker.

Each worker should have its own config file. There is an example you can use named `worker.cfg` in the `ckanext-queue` source code. If you don’t specify a config file, the defaults will be used.
Tip: Since the worker.cfg and CKAN configuration file are both in INI file format you can also set these variables directly in your CKAN config file and point the worker directly to your CKAN config instead. It will just ignore the CKAN options.

In particular it is worth setting queue.name which will be used internally by RabbitMQ.

Here’s a suitable configuration for a worker:

```
[worker]
ckan.site_id = local_test
ckan.site_url = http://localhost:5000
ckan.api_key = XXX

# queue.name =
# queue.routing_key = *
# queue.port =
# queue.user_id =
# queue.password =
# queue.hostname =
# queue.virtual_host =
```

You can run it like this:

```
worker echo -d -c worker.cfg
```

The echo example comes from ckanext.queue.echo:EchoWorker. It looks like this:

```
from worker import Worker
from pprint import pprint

class EchoWorker(Worker):
    def consume(self, routing_key, operation, payload):
        print "Route %s, op %s" % (routing_key, operation)
        pprint(payload)
```

The EchoWorker has an entry point registered in ckanext-queue's setup.py so that the worker script in pyenv/bin/worker can find it:

```
[ckan.workers]
echo = ckanext.queue.echo:EchoWorker
```

When you run the worker command with the echo worker it looks up this entry point to run the correct code.

With the worker still running, try to run the publisher again:

```
python publish_on_queue.py
```

Once again the message will be output on the command line, this time via the worker.

Internally the worker script you just used uses the ckanext.queue.worker.WorkerChain class to run all workers you specify on the command line. You can get a list of all available workers by executing worker with no arguments.

```
# worker
WARNING:root:No config file specified, using worker.cfg
```
10.3.4 Configuring CKAN to use the queue

Once you have installed RabbitMQ and have it running you need to enable the CKAN queue functionality within CKAN by adding this to your CKAN config file

```python
ckan.plugins = queue
```

You don’t need to specify configuration options to connect to RabbitMQ because the defaults are fine.

At this point if you edit a package it should be using the queue. If you have the echo worker running you’ll see the message added to the queue.

Logging

When using the queue with CKAN it is also useful to have logging set up.

To get logging working you need to modify your CKAN config file to also include the `queue` logger. Here’s an example:

```python
[loggers]
keys = root, queue

[handlers]
keys = console

[formatters]
keys = generic

[logger_root]
level = INFO
handlers = console

[logger_queue]
level = DEBUG
handlers = console
qualname = ckanext

You will also need to set this in your CKAN configuration and ensure any workers and producers also set their `ckan.site_id` to the same value.

```python
ckan.site_id = local_test
```

Now that you know about extensions, plugins and workers you should be able to extend CKAN in lots of new and interesting ways.
The forms used to edit packages and groups in CKAN can be customized. This lets you tailor them to your needs, helping your users choose from sensible options or use different data formats.

This document explains how to customize the package and group forms you offer to your users, without getting embroiled in the core CKAN code.

**Note:** This section deals with the form used to *edit* packages and groups, not the way they are displayed. For information on customizing the display of forms, see *Customization*.

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

### 11.1 Building a Package Form

#### 11.1.1 The Best Way: Extensions

The best way to build a package form is by using a CKAN extension.

You will firstly need to make a new controller in your extension. This should subclass PackageController as follows:

```python
from ckan.controllers.package import PackageController
class PackageNew(PackageController):
    package_form = 'custom_package_form.html'
```

The `package_form` variable in the subclass will be used as the new form template.

It is recommended that you copy the package form (`new_package_form.html`) and make modifications to it. However, it is possible to start from scratch.

To point at this new controller correctly, your extension should look like the following:

```python
class CustomForm(SingletonPlugin):
    implements(IRoutes)
    implements(IConfigurer)
    def before_map(self, map):
        map.connect('/package/new', controller='ckanext.extension_name.controllers.PackageNewController:PackageNew', action='new')
        map.connect('/package/edit/{id}', controller='ckanext.extension_name.controllers.PackageNewController:PackageNew', action='edit')
        return map
    def after_map(self, map):
        return map
```
def update_config(self, config):
    configure_template_directory(config, 'templates')

Replace `extension_name` with the name of your extension.

This also assumes that `custom_package_form.html` is located in the `templates` subdirectory of your extension, i.e. `ckanext/extension_name/templates/custom_package_form.html`.

### 11.1.2 Advanced Use

The PackageController has more hooks to customize the displayed data. These functions can be overridden in a subclass of PackageController:

- `_setup_template_variables(self, context)`

  This is for setting up new variables for your templates:

- `_form_to_db_schema(self)`

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

- `_db_to_form_schema(self)`

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

A complex example of the use of these hooks can be found in the `ckanext-dgu` extension.
CKAN allows you to integrate its Edit Package and New Package 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.

12.1 Redirecting CKAN Forms

It is obviously simple enough for an external front-end to link to CKAN’s Edit Package and New Package 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 package read page.

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

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

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

Since the ‘return URL’ may need to include the package 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.

12.1.1 Example

An external front-end displays a package ‘ontariolandcoverv100’ here:

http://datadotgc.ca/dataset/ontariolandcoverv100

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

http://ca.ckan.net/package/edit/ontariolandcoverv100

At first, it may seem that the return link should be

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

But when the user edits this package, the name may change. So the return link needs to be:

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

And this is URL-encoded to become:
http://datadotgc.ca/dataset/<NAME>

So, in summary, the edit link becomes:

http://ca.ckan.net/package/edit/ontariolandoverv100?return_to=http://datadotgc.ca/dataset/<NAME>

During editing the package, the user changes the package 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)*:

```ini
...[app:main]
package_edit_return_url = http://datadotgc.ca/dataset/<NAME>
...```
It’s often useful to allow users to download a complete CKAN database in a dumpfile.

For example, you can download ckan.net’s daily dump at: http://ckan.net/dump/ in JSON format. The file is called ckan.net-daily.json.gz.

### 13.1 Creating a Dump

We provide two `paster` methods to create dumpfiles.

- **db simple-dump-json** - A simple dumpfile, useful to create a public listing of the packages with no user information. All packages are dumped, including deleted packages and ones with strict authorization.

- **db dump** - A more complicated dumpfile, useful for backups. Replicates the database completely, including users, their personal info and API keys, and hence should be kept private.

For more information on `paster`, see *Common CKAN Tasks*.

#### 13.1.1 Using db simple-dump-json

If you are using a Python environment, as part of a development installation, first enable the environment:

```
. /home/okfn/var/srvc/ckan.net/pyenv/bin/activate || exit 1
```

Then create and zip the dumpfile:

```
paster --plugin=ckan db simple-dump-json /var/srvc/ckan/dumps/ckan.net-daily.json --config=/etc/ckan/std/std.ini
gzip /var/srvc/ckan/dumps/ckan.net-daily.json
```

Change `simple-dump-json` to `simple-dump-csv` if you want CSV format instead of JSON.

#### 13.1.2 Using db dump

If you are using a Python environment, as part of a development installation, first enable the environment:

```
. /var/srvc/ckan/pyenv/bin/activate || exit 1
```

Then create and zip the dumpfile:

```
paster --plugin=ckan db dump /var/srvc/ckan/dumps/ckan.net-daily --config=/etc/ckan/standards/standards.ini
gzip /var/srvc/ckan/dumps/ckan.net-daily
```
13.2 Daily Dumps

You can set the dump to be created daily with a cron job.

Edit your user’s cron config:

$ crontab -e

Now add a line such as this:

0 21 * * * /home/okfn/var/srvc/ckan.net/dump.sh

13.3 Serving the Files

Some simple additions to the Apache config can serve the files to users in a directory listing.

To do this, add these lines to your virtual host config (e.g. /etc/apache2/sites-enabled/ckan.net):

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

# Disable the mod_python handler for static files
<Location /dump>
    SetHandler None
    Options +Indexes
</Location>
CHAPTER 14

Upgrade CKAN

Any time you want to upgrade CKAN to a newer version, first back up your database:

```bash
paster --plugin=ckan db dump demo_ckan_backup.pg_dump --config=demo.ckan.net.ini
```

Then run this as root:

```bash
apt-get update
apt-get dist-upgrade
ckan-std-install
```

You need to use `dist-upgrade` rather than `upgrade` because it is possible more recent versions of CKAN will add new dependencies rather than just updating existing ones, and apt will only install new dependencies if you use `dist-upgrade`.

The `ckan-std-install` will also upgrade your existing CKAN instance, keeping a backup of your old CKAN config file.
Internationalize CKAN

CKAN is used in many countries, and adding a new language is a simple process.

15.1 Supported Languages

CKAN already supports numerous languages. To check whether your language is supported, look in the source at ckan/i18n for translation files. Languages are named using two-letter ISO language codes (e.g. es, de).

If your language is present, you can switch language simply by setting the lang option in your CKAN config file, as described in Internationalisation Settings. For example, to switch to German:

```python
lang=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.

15.2 Adding a New Language

If you want to add an entirely new language to CKAN, you have two options.

- Transifex Setup. Creating translation files using Transifex, the open source translation software.

15.2.1 Manual Setup

If you prefer not to use Transifex, you can create translation files manually.

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:

```bash
git clone http://bitbucket.org/bboissin/ckan-i18n/
```

Then create a translation file for your language (a po file) using the pot file:

```bash
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:

```bash
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:

```bash
hg add ckan/i18n/YOUR_LANGUAGE/LC_MESSAGES/ckan.po
hg ci -m '[i18n]: New language po added: YOUR_LANGUAGE' ckan/i18n/YOUR_LANGUAGE/LC_MESSAGES/ckan.po
hg 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:

```bash
ckan/i18n/YOUR_LANGUAGE/LC_MESSAGES/ckan.po
```

In this repo, compile the po file like this:

```bash
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 `lang` option in the CKAN config file, as described in *Internationalisation Settings*.

15.2.2 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/](https://www.transifex.net/projects/p/ckan/teams/)

Updated translations are automatically pushed to [https://bitbucket.org/bboissin/ckan-i18n](https://bitbucket.org/bboissin/ckan-i18n) and these can be compiled and placed on CKAN servers by the server administrators.

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`
- Commit and push po and mo files
Reference: CKAN Configuration Options

You can change many important CKAN settings in the CKAN config file. This is the file called std.ini that you first encountered in Create an Admin User. It is usually located at /etc/ckan/std/std.ini.

The file is well-documented, but we recommend reading this section in full to learn about the CKAN config options available to you.

Note: After editing this file, you will need to restart Apache for the changes to take effect.

Note: The CKAN config file also includes general Pylons options. All CKAN-specific settings are in the [app:main] section.

16.1 Database Settings

16.1.1 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

16.2 Front-End Settings

16.2.1 site_description

Example:
ckan.site_description=

Default value: (none)

This is for a description, or tag line for the site, as displayed in the header of the CKAN web interface.
16.2.2 site_logo

Example:

ckan.site_logo=/images/ckan_logo_fullname_long.png

Default value: (none)

This sets the logo used in the title bar.

16.2.3 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 package 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.

16.2.4 rdf_packages

Example:

rdf_packages = http://semantic.ckan.net/record/

Configure this if you have an RDF store of the same packages as are in your CKAN instance. It will provide three sorts of links from each package page to the equivalent RDF URL given in `rdf_packages`:


2. Embedded links for browsers that are aware. e.g. `<link rel="alternate" type="application/rdf+xml" href="http://semantic.ckan.net/record/b410e678-8a96-40cf-8e46-e8bd4bf02684.rdf">`

3. A visible RDF link on the page. e.g. `<a href="http://semantic.ckan.net/record/b410e678-8a96-40cf-8e46-e8bd4bf02684.rdf">`

16.2.5 dumps_url & dumps_format

Example:

ckan.dumps_url = http://ckan.net/dump/
ckan.dumps_format = CSV/JSON

If there is a page which allows you to download a dump of the entire catalogue then specify the URL and the format here, so that it can be advertised in the web interface. `dumps_format` is just a string for display.

For more information on using dumpfiles, see *Database Dumps*. 
16.3 Cache Settings

16.3.1 cache_validation_enabled

Example:

ckan.cache_validation_enabled = False

Default value: True

This option determines whether browsers (or other caching services running between the browser and CKAN) are helped to cache particular CKAN pages, by validating when the page content hasn’t changed. This is achieved using ETag headers provided by CKAN, which is a hash that changes when the content has changed.

Developers editing the templates should set this to False, since ETag hashes don’t look for template changes.

16.3.2 cache_enabled

Example:

ckan.cache_enabled = True

Default value: False

Setting this option to True turns on several server-side caches. When caching is on, caching can be further configured as follows.

To set the type of Beaker storage:

beaker.cache.type = file

To set the expiry times (in seconds) for specific controllers (which use the proxy_cache) specify the methods like this:

ckan.controllers.package.list.expires = 600
ckan.controllers.tag.read.expires = 600
ckan.controllers.apiv1.package.list.expires = 600
ckan.controllers.apiv1.package.show.expires = 600
ckan.controllers.apiv2.package.list.expires = 600
ckan.controllers.apiv2.package.show.expires = 600

There is also an option to set the max-age value of static files delivered by paster:

ckan.static_max_age = 3600

16.4 Authentication Settings

16.4.1 openid_enabled

Example:

openid_enabled = False

Default value: True

CKAN operates a delegated authentication model based on OpenID.

Setting this option to False turns off OpenID for login.
16.5 Internationalisation Settings

16.5.1 lang

Example:
lang=de

Default value: en (English)

Use this to specify the language of the text displayed in the CKAN web UI. This requires a suitable mo file installed for the language. For more information on internationalization, see Internationalize CKAN.

16.6 Theming Settings

16.6.1 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 Customization.

16.6.2 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 Customization.

16.7 Form Settings

16.7.1 package_form

Example:
package_form = ca

Default value: standard

This sets the name of the form to use when editing a package. This can be a form defined in the core CKAN code or in another setuputils-managed python module. The only requirement is that the setup.py file has an entry point for the form defined in the ckan.forms section.

For more information on forms, see Customizing Forms.
### 16.7.2 package_new_return_url & package_edit_return_url

Example:

```python
package_new_return_url = http://datadotgc.ca/new_dataset_complete?name=<NAME>
package_edit_return_url = http://datadotgc.ca/dataset/<NAME>
```

If integrating the Edit Package and New Package forms into a third-party interface, setting these options allows you to set the return address. When the user has completed the form and presses ‘commit’, the user is redirected to the URL specified.

The `<NAME>` string is replaced with the name of the package edited. Full details of this process are given in *Form Integration*.

### 16.7.3 licenses_group_url

A url pointing to a JSON file containing a list of licence objects. This list determines the licences offered by the system to users, for example when creating or editing a package.

This is entirely optional - by default, the system will use the CKAN list of licences available in the Python licenses package.

More details about the CKAN license objects - including the licence format and some example licence lists - can be found at the Open Licenses Service.

Examples:

```python
licenses_group_url = file:///path/to/my/local/json-list-of-licenses.js
licenses_group_url = http://licenses.opendefinition.org/2.0/ckan_original
```

### 16.8 Messaging Settings

#### 16.8.1 carrot_messaging_library

Example:

```python
carrot_messaging_library=pyamqplib
```

This is the messaging library backend to use. Options:

- `'pyamqplib'` - AMQP (e.g. for RabbitMQ)
- `'pika'` - alternative AMQP
- `'stomp'` - python-stomp
- `'queue'` - native Python Queue (default) - NB this doesn’t work inter-process

See the Carrot documentation for details.

#### 16.8.2 amqp_hostname, amqp_port, amqp_user_id, amqp_password

Example:
Comprehensive Knowledge Archive Network (CKAN) Developer Documentation, Release 1.4.3

amqp_hostname=localhost
amqp_port=5672
amqp_user_id=guest
amqp_password=guest

These are the setup parameters for AMQP messaging. These only apply if the messaging library has been set to use AMQP (see carrot_messaging_library). The values given above are the default values.

16.9 Search Settings

16.9.1 build_search_index_synchronously

Example:
ckan.build_search_index_synchronously=

Default (if you don’t define it):: indexing is on

This controls the operation of the CKAN Postgres full text search indexing. If you don’t define this option then indexing is on. You will want to turn this off if you want to use a different search engine for CKAN (e.g. Solr). In this case you need to define the option equal to blank (as in the example).

16.9.2 search_backend

Example:
search_backend = solr

Default value: sql

This controls the type of search backend. Currently valid values are sql (meaning Postgres full text search) and solr (meaning Solr). If you specify sql then ensure indexing is on (build_search_index_synchronously is not defined). If you specify solr then ensure you specify a solr_url.

16.9.3 solr_url

Example:
solr_url = http://solr.okfn.org/solr/test.ckan.net

This configures Solr search (if selected with search_backend). Running Solr will require a schema.xml file, such as the one in the ckanext-solr repository.

Optionally, solr_user and solr_password can also be passed along to specify HTTP Basic authentication details for all Solr requests.

16.10 Site Settings

16.10.1 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.

16.10.2 site_url

Example:
ckan.site_url=http://scotdata.ckan.net

Default value: (none)
The primary URL used by this site. Used in the API to provide packages with links to themselves in the web UI.

16.10.3 api_url

Example:
ckan.api_url=http://scotdata.ckan.net/api

Default value: /api
The URL that resolves to the CKAN API part of the site. This is useful if the API is hosted on a different domain, for example when a third-party site uses the forms API.

16.11 Authorization Settings

16.11.1 default_roles

This allows you to set the default authorization roles (i.e. permissions) for new objects. Currently this extends to new packages, groups, authorization groups and the system object. For full details of these, see Set and Manage Permissions.

The value is a strict JSON dictionary of user names visitor (any user who is not logged in) and logged_in (any user who is logged in) with lists of their roles.

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

With this example setting, visitors and logged-in users can only read packages that get created.
Defaults: see in ckan/model/authz.py for default_default_user_roles

16.12 Plugin Settings

16.12.1 plugins

Example:
ckan.plugins = disqus synchronous_search datapreview googleanalytics stats storage admin follower

Specify which CKAN extensions are to be enabled.

**Warning:** If you specify an extension but have not installed the code, CKAN will not start.

Format as a space-separated list of the extension names. The extension name is the key in the `[ckan.plugins]` section of the extension’s `setup.py`. For more information on extensions, see *Add Extensions*.

### 16.13 Directory Settings

#### 16.13.1 `log_dir`

Example:

```python
ckan.log_dir = /var/log/ckan/
```

This is the directory to which CKAN cron scripts (if there are any installed) should write log files.

**Note:** This setting is nothing to do with the main CKAN log file, whose filepath is set in the `[handler_file]` args.

#### 16.13.2 `dump_dir`

Example:

```python
ckan.dump_dir = /var/lib/ckan/dump/
```

This is the directory to which JSON or CSV dumps of the database are to be written, assuming a script has been installed to do this.

**Note:** It is usual to set up the Apache config to serve this directory.

#### 16.13.3 `backup_dir`

Example:

```python
ckan.backup_dir = /var/backup/ckan/
```

This is a directory where SQL database backups are to be written, assuming a script has been installed to do this.
The CKAN API provides programmatic access to the catalog of metadata stored in CKAN.

17.1 Overview

The CKAN data catalog is not only available in a web browser, but also via its Application Programming Interface (API).

The API can be used to view and change the catalog. This document describes the resource locations, data formats, and status codes which comprise the CKAN API, so that anyone can create software applications that use the API service.

The CKAN API follows the RESTful (Representational State Transfer) style. Resource locations are separated both from the methods supported by the resources, and from the data formats and status codes used by the methods.

17.2 Examples

For a tutorial and examples of using the CKAN API, see: http://wiki.ckan.net/Using_the_API

17.3 Code Modules for Client Applications

There are also some code modules (Python, PHP, Drupal, Perl etc.) that provide convenient wrappers around much of the CKAN API. For full details of these, please consult http://wiki.ckan.net/API

17.4 Versions

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

If the API version is not specified in the request, then the API will default to version 1.
17.4.1 Versions 1 & 2

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

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

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

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

17.5 API Details - Versions 1 & 2

17.5.1 Overview

The CKAN data catalog is not only available in a web browser, but also via its Application Programming Interface (API).

The API can be used to view and change the catalog. This document describes the resource locations, data formats, and status codes which comprise the CKAN API, so that anyone can create software applications that use the API service.

The CKAN API follows the RESTful (Representational State Transfer) style. Resource locations are separated both from the methods supported by the resources, and from the data formats and status codes used by the methods.

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

• Model API
• Search API
• Util API

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

17.5.2 Locators

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

Resource Locator = API Locator + Resource Path

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

• http://ckan.net/api (version 1)
• http://ckan.net/api/1 (version 1)
• http://ckan.net/api/2 (version 2)

The relative paths for each resource are listed in the sections below.
17.5.3 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.

<table>
<thead>
<tr>
<th>Model Resource</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package Register</td>
<td>/rest/package</td>
</tr>
<tr>
<td>Package Entity</td>
<td>/rest/package/PACKAGE-REF</td>
</tr>
<tr>
<td>Group Register</td>
<td>/rest/group</td>
</tr>
<tr>
<td>Group Entity</td>
<td>/rest/group/GROUP-REF</td>
</tr>
<tr>
<td>Tag Register</td>
<td>/rest/tag</td>
</tr>
<tr>
<td>Tag Entity</td>
<td>/rest/tag/TAG-NAME</td>
</tr>
<tr>
<td>Rating Register</td>
<td>/rest/rating</td>
</tr>
<tr>
<td>Package Relationships Register</td>
<td>/rest/package/PACKAGE-REF/relationships</td>
</tr>
<tr>
<td>Package Relationships Register</td>
<td>/rest/package/PACKAGE-REF/RELATIONSHIP-TYPE</td>
</tr>
<tr>
<td>Package Relationships Register</td>
<td>/rest/package/PACKAGE-REF/relationships/PACKAGE-REF</td>
</tr>
<tr>
<td>Package Relationship Entity</td>
<td>/rest/package/PACKAGE-REF/RELATIONSHIP-TYPE/PACKAGE-REF</td>
</tr>
<tr>
<td>Package's Revisions Entity</td>
<td>/rest/package/PACKAGE-REF/revisions</td>
</tr>
<tr>
<td>Revision Register</td>
<td>/rest/revision</td>
</tr>
<tr>
<td>Revision Entity</td>
<td>/rest/revision/REVISION-ID</td>
</tr>
<tr>
<td>License List</td>
<td>/rest/licenses</td>
</tr>
</tbody>
</table>

Possible values for PACKAGE-REF are the package id, or the current package name.

Possible values for RELATIONSHIP-TYPE are described in the Relationship-Type data format.

Model Methods

Here are the methods of the Model API.
<table>
<thead>
<tr>
<th>Resource</th>
<th>Method</th>
<th>Request</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package Register</td>
<td>GET</td>
<td>Package</td>
<td>Package-List</td>
</tr>
<tr>
<td>Package Register</td>
<td>POST</td>
<td>Package</td>
<td>Package-List</td>
</tr>
<tr>
<td>Package Entity</td>
<td>GET</td>
<td>Package</td>
<td>Package</td>
</tr>
<tr>
<td>Package Entity</td>
<td>PUT</td>
<td>Package</td>
<td>Group-List</td>
</tr>
<tr>
<td>Group Register</td>
<td>GET</td>
<td>Group</td>
<td>Group-List</td>
</tr>
<tr>
<td>Group Register</td>
<td>POST</td>
<td>Group</td>
<td>Group</td>
</tr>
<tr>
<td>Group Entity</td>
<td>GET</td>
<td>Group</td>
<td>Tag-List</td>
</tr>
<tr>
<td>Group Entity</td>
<td>PUT</td>
<td>Group</td>
<td>Package-List</td>
</tr>
<tr>
<td>Tag Register</td>
<td>GET</td>
<td>Rating</td>
<td>Package-List</td>
</tr>
<tr>
<td>Tag Entity</td>
<td>GET</td>
<td>Rating</td>
<td>Package-List</td>
</tr>
<tr>
<td>Rating Register</td>
<td>POST</td>
<td>Rating</td>
<td>Rating</td>
</tr>
<tr>
<td>Rating Entity</td>
<td>GET</td>
<td>Rating</td>
<td>Rating</td>
</tr>
<tr>
<td>Package Relationships Register</td>
<td>GET</td>
<td>Pkg-Relationship</td>
<td>Pkg-Relationship</td>
</tr>
<tr>
<td>Package Relationship Entity</td>
<td>GET</td>
<td>Pkg-Relationship</td>
<td>Pkg-Relationship</td>
</tr>
<tr>
<td>Package Relationship Entity</td>
<td>PUT</td>
<td>Pkg-Relationship</td>
<td>Pkg-Relationship</td>
</tr>
<tr>
<td>Package’s Revisions Entity</td>
<td>GET</td>
<td>Pkg-Revisions</td>
<td>Pkg-Revisions</td>
</tr>
<tr>
<td>Revision List</td>
<td>GET</td>
<td>Revision-List</td>
<td>Revision-List</td>
</tr>
<tr>
<td>Revision Entity</td>
<td>GET</td>
<td>Revision</td>
<td>Revision</td>
</tr>
<tr>
<td>License List</td>
<td>GET</td>
<td>License-List</td>
<td>License-List</td>
</tr>
</tbody>
</table>

- POSTing data to a register resource will create a new entity.
- PUT/POSTing data to an entity resource will update an existing entity.
- PUT operations may instead use the HTTP POST method.

**Model Formats**

Here are the data formats for the Model API.

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 package then some additional information is supplied that cannot current be adjusted through the CKAN API. This includes info on Package Relationship (‘relationships’), Group membership (‘groups’), ratings (‘ratings_average’ and ‘ratings_count’), full URL of the package in CKAN (‘ckan_url’) and Package ID (‘id’). This is purely a convenience for clients, and only forms part of the Package on GET.

### 17.5.4 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
--- | ---
Package Search | /search/package
Resource Search | /search/resource
Revision Search | /search/revision
Tag Counts | /tag_counts

See below for more information about package and revision search parameters.

**Search Methods**

Here are the methods of the Search API.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Method</th>
<th>Request</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package Search</td>
<td>POST</td>
<td>Package-Search-Params</td>
<td>Package-Search-Response</td>
</tr>
<tr>
<td>Resource Search</td>
<td>POST</td>
<td>Resource-Search-Params</td>
<td>Resource-Search-Response</td>
</tr>
<tr>
<td>Revision Search</td>
<td>POST</td>
<td>Revision-Search-Params</td>
<td>Revision-List</td>
</tr>
<tr>
<td>Tag Counts</td>
<td>GET</td>
<td></td>
<td>Tag-Count-List</td>
</tr>
</tbody>
</table>

It is also possible to supply the search parameters in the URL of a GET request, for example /api/search/package?q=geodata&allfields=1.

**Search Formats**

Here are the data formats for the Search API.

<table>
<thead>
<tr>
<th>Name</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package-Search-Params</td>
<td>[ Param-Key: Param-Value, Param-Key: Param-Value, ... ] See below for full details of search parameters across the various domain objects.</td>
</tr>
<tr>
<td>Resource-Search-Params</td>
<td>{ count: Count-int, results: [Package, Package, ... ] }</td>
</tr>
<tr>
<td>Revision-Search-Params</td>
<td>{ count: Count-int, results: [Resource, Resource, ... ] }</td>
</tr>
<tr>
<td>Package-Search-Response</td>
<td>[ Revision-Id, Revision-Id, Revision-Id, ... ] NB: Ordered with youngest revision first</td>
</tr>
<tr>
<td>Tag-Count-List</td>
<td>[ [Name-String, Integer], [Name-String, Integer], ... ]</td>
</tr>
</tbody>
</table>

The Package and Revision data formats are as defined in Model Formats.

**Package Parameters**
### Param-Value

<table>
<thead>
<tr>
<th>Param-Value</th>
<th>Examples</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>q</strong></td>
<td>Search-String</td>
<td>q=geodata</td>
</tr>
<tr>
<td></td>
<td></td>
<td>q=government+sweden</td>
</tr>
<tr>
<td></td>
<td></td>
<td>q=%22drug%20abuse%22</td>
</tr>
<tr>
<td><strong>qjson</strong></td>
<td>JSON encoded options</td>
<td>['q': 'geodata']</td>
</tr>
<tr>
<td><strong>title</strong></td>
<td>Search-String</td>
<td>title=uk&amp;tags=health+census</td>
</tr>
<tr>
<td></td>
<td></td>
<td>department=environment</td>
</tr>
<tr>
<td><strong>author</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>maintainer</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>update_frequency</strong></td>
<td>Search-String</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>order_by</strong></td>
<td>field-name (default=rank)</td>
<td>order_by=rank</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>offset, limit</strong></td>
<td>result-int (defaults: offset=0, limit=20)</td>
<td>offset=40&amp;limit=20</td>
</tr>
<tr>
<td><strong>all_fields</strong></td>
<td>0 (default) or 1</td>
<td>all_fields=1</td>
</tr>
<tr>
<td><strong>filter_by_openness</strong></td>
<td>0 (default) or 1</td>
<td>filter_by_openness=1</td>
</tr>
<tr>
<td><strong>filter_by_downloadable</strong></td>
<td>0 (default) or 1</td>
<td>filter_by_downloadable=1</td>
</tr>
</tbody>
</table>

### Resource Parameters

- **q**: Criteria to search the package 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.
- **qjson**: 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 in a particular a field.
- **order_by**: Specify either rank or the field to sort the results by.
- **offset, limit**: Pagination options. Offset is the number of the first result and limit is the number of results to return.
- **all_fields**: Each matching search result is given as either a package name (0) or the full package record (1).
- **filter_by_openness**: Filters results by ones which are open.
- **filter_by_downloadable**: Filters results by ones which have at least one resource URL.
<table>
<thead>
<tr>
<th>Param-Key</th>
<th>Param-Value</th>
<th>Example</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>url, format, description</td>
<td>Search-String</td>
<td>url=statistics.org format=xls</td>
<td>Criteria to search the package fields for. URL-encoded search text.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>description=Research+Institute</td>
<td>This search string must be found somewhere within the field to match. Case insensitive.</td>
</tr>
<tr>
<td>qjson</td>
<td>JSON encoded options</td>
<td>['url': 'www.statistics.org']</td>
<td>All search parameters can be json-encoded and supplied to this parameter as a more flexible alternative in GET requests.</td>
</tr>
<tr>
<td>hash</td>
<td>Search-String</td>
<td>hash=b0d7c260-35d4-42ab-9e3d-c1f4db9bc2f0</td>
<td>Searches for a match of the hash field. An exact match or match up to the length of the hash given.</td>
</tr>
<tr>
<td>all_fields</td>
<td>0 (default) or 1</td>
<td>all_fields=1</td>
<td>Each matching search result is given as either an ID (0) or the full resource record. Pagination options. Offset is the number of the first result and limit is the number of results to return.</td>
</tr>
<tr>
<td>offset, limit</td>
<td>result-int (defaults: offset=0, limit=20)</td>
<td>offset=40&amp;limit=20</td>
<td></td>
</tr>
</tbody>
</table>

**Revision Parameters**

<table>
<thead>
<tr>
<th>Param-Key</th>
<th>Param-Value</th>
<th>Example</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>since_time</td>
<td>Date-Time</td>
<td>since_time=2010-05-05T19:42:45.854533</td>
<td>The time can be less precisely stated (e.g 2010-05-05).</td>
</tr>
<tr>
<td>since_id</td>
<td>Uuid</td>
<td>since_id=6c9f32ef-1f93-4b2f-891b-fd01924ebe08</td>
<td>The stated id will not be included in the results.</td>
</tr>
</tbody>
</table>

**17.5.5 API Keys**

You will need to supply an API Key for certain requests to the CKAN API:

- For any action which makes a change to a resource (i.e. all POST methods on register resources, and PUT/POST methods on entity resources).

- If the particular resource’s authorization set-up is not open to visitors for the action.

To obtain your API key:

1. Log-in to the particular CKAN website: /user/login
2. The user page has a link to the API Key: /user/apikey

The key should be passed in the API request header:

<table>
<thead>
<tr>
<th>Header</th>
<th>Example value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization</td>
<td>fde34a3c-b716-4c39-8dc4-881ba115c6d4</td>
</tr>
</tbody>
</table>

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

For more information about HTTP Authorization header, please refer to section 14.8 of RFC 2616.
17.5.6 Status Codes

Standard HTTP status codes are used to signal method outcomes.

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>201</td>
<td>OK and new object created (referred to in the Location header)</td>
</tr>
<tr>
<td>301</td>
<td>Moved Permanently</td>
</tr>
<tr>
<td>400</td>
<td>Bad Request</td>
</tr>
<tr>
<td>403</td>
<td>Not Authorized</td>
</tr>
<tr>
<td>404</td>
<td>Not Found</td>
</tr>
<tr>
<td>409</td>
<td>Conflict (e.g. name already exists)</td>
</tr>
<tr>
<td>500</td>
<td>Service Error</td>
</tr>
</tbody>
</table>

17.5.7 JSONP formatted responses

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

Example normal request:

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

but now with the callback parameter:

GET /api/rest/package/pollution_stats?callback=jsoncallback
returns: jsoncallback{"name": "pollution_stats", ... };

This parameter can apply to all GET requests in the API.

17.5.8 Util API

Some of CKAN’s client-side Javascript code makes calls to the CKAN API. For example, to generate a suggestion for a package name when adding a new package the following API call is made:

/api/2/util/package/create_slug?title=Package+1+Title+Typed+So+Far

The return value is a JSON data structure:

{"valid": true, "name": "package_1_title_typed_so_far"}

These are the keys returned:

valid

Can be True or False. It is true when the title entered can be successfully turned into a package name and when that package name is not already being used. It is false otherwise.

name

The suggested name for the package, based on the title

You can also add callback=callback to have the response returned as JSONP. eg:

This URL:

/api/2/util/package/create_slug?title=Package+1+Title+Typed+So+Far&callback=callback
Returns:

callback({"valid": true, "name": "package_1_title_typed_so_far"});

In some CKAN deployments you may have the API deployed at a different domain from the main CKAN code. In these circumstances you’ll need to add a new option to the config file to tell the new package form where it should make its API requests to:

ckan.api_url = http://api.example.com/

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"}]}}
CHAPTER 18

Testing for Developers

If you are installing CKAN from source, or developing extensions, then you need to know how to run CKAN tests. This section describes testing topics for developers, including basic tests, migration testing and testing against PostgreSQL.

18.1 Basic Tests

After completing your source installation of CKAN, you should check that tests pass. You should also check this before checking in changes to CKAN code.

Make sure you’ve created a config file at `pyenv/ckan/development.ini`. Then activate the Python environment:

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

Install nose into your virtual environment:

```
pip install --ignore-installed nose
```

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

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

Then run the quick development tests:

```
cd pyenv/src/ckan
nosetests ckan/tests --ckan
```

You must run the tests from the CKAN directory as shown above, otherwise the `--ckan` plugin won’t work correctly.

**Warning:** By default, the test run is ‘quick and dirty’ - only good enough as an initial check.

18.2 Testing against PostgreSQL

The default way to run tests is defined in `test.ini` (which is the default config file for nose - change it with option `--with-pylons`). This specifies using SQLite and sets `faster_db_test_hacks`, which are compromises.
cd pyenv/src/ckan
	nosetests ckan/tests --ckan

Although SQLite is useful for testing a large proportion of CKAN, actually in deployment, CKAN must run with PostgreSQL.

Running the tests against PostgreSQL is slower but more thorough for two reasons:

1. You test subtleties of PostgreSQL
2. CKAN’s default search relies on PostgreSQL’s custom full-text search, so these (100 or so) tests are skipped when running against SQLite.

So when making changes to anything involved with search or closely related to the database, it is wise to test against PostgreSQL.

To test against PostgreSQL:

1. Edit your local `development.ini` to specify a PostgreSQL database with the `sqlalchemy.url` parameter.
2. Tell nose to use `test-core.ini` (which imports settings from `development.ini`)

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

The test suite takes a long time to run against standard PostgreSQL (approx. 15 minutes, or close to an hour on Ubuntu/10.04 Lucid).

This can be improved to between 5 and 15 minutes by running PostgreSQL in memory and turning off durability, as described in the PostgreSQL documentation.

### 18.3 Migration Testing

If your changes require a model change, you’ll need to write a migration script. To ensure this is tested as well, you should instead run the tests this way:

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

By default, tests are run using the model defined in `ckan/model`, but by using the `--ckan-migrate` option the tests will run using a database that has been created using the migration scripts, which is the way the database is created and upgraded in production. These tests are the most thorough and will take around 20 minutes.

**Caution:** Ordinarily, you should set `development.ini` to specify a PostgreSQL database so these also get used when running `test-core.ini`, since `test-core.ini` inherits from `development.ini`. If you were to change the `sqlalchemy.url` option in your `development.ini` file to use SQLite, the command above would actually test SQLite rather than PostgreSQL, so always check the setting in `development.ini` to ensure you are running the full tests.

**Warning:** A common error when wanting to run tests against a particular database is to change `sqlalchemy.url` in `test.ini` or `test-core.ini`. The problem is that these are versioned files and people have checked in these by mistake, creating problems for other developers and the CKAN buildbot. This is easily avoided by only changing `sqlalchemy.url` in your local `development.ini` and testing `--with-pylons=test-core.ini`. 
18.4 Common problems running tests

- `nose.config.ConfigError: Error reading config file ‘setup.cfg’: no such option ‘with-pylons’`

  This error can result when you run nosetests for two reasons:

  1. Pylons nose plugin failed to run. If this is the case, then within a couple of lines of running `nosetests` you’ll see this warning: *Unable to load plugin pylons* followed by an error message. Fix the error here first.

  2. The Python module ‘Pylons’ is not installed into you Python environment. Confirm this with:

     ```
     python -c "import pylons"
     ```

- `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.
This section provides information for CKAN core developers setting up buildbot on an Ubuntu Lucid machine. If you simply want to check the status of the latest CKAN builds, visit http://buildbot.okfn.org/.

### 19.1 Apt Installs

Install CKAN core dependencies from Lucid distribution:

```bash
sudo apt-get install build-essential libxml2-dev libxslt-dev
sudo apt-get install wget mercurial postgresql libpq-dev git-core
sudo apt-get install python-dev python-psycopg2 python-virtualenv
sudo apt-get install subversion
```

Maybe need this too:

```bash
sudo apt-get install python-include
```

**Buildbot software:**

```bash
sudo apt-get install buildbot
```

**Deb building software:**

```bash
sudo apt-get install -y dh-make devscripts fakeroot cdbs
```

**Fabric:**

```bash
sudo apt-get install -y fabric
```

If you get errors with postgres and locales you might need to do these:

```bash
sudo apt-get install language-pack-en-base
sudo dpkg-reconfigure locales
```

### 19.2 Postgres Setup

If installation before failed to create a cluster, do this after fixing errors:

```bash
sudo pg_createcluster 8.4 main --start
```
Create users and databases:

```
sudo -u postgres createuser -S -D -R -P buildslave
# set this password (matches buildbot scripts): biomaik15
sudo -u postgres createdb -O buildslave ckanl
sudo -u postgres createdb -O buildslave ckanext
```

### 19.3 Buildslave Setup

**Rough commands:**

```
sudo useradd -m -s /bin/bash buildslave
sudo chown buildslave:buildslave /home/buildslave
sudo su buildslave
cd ~
hg clone https://dread@bitbucket.org/okfn/buildbot-scripts .
ssh-keygen -t rsa
cp /home/buildslave/.ssh/id_rsa.pub ~/.ssh/authorized_keys
mkdir -p ckan/build
cd ckan/build
python ~/ckan-default.py
buildbot create-slave ~ localhost:9989 okfn <buildbot_password>
vim ~/info/admin
vim ~/info/host
mkdir /home/buildslave/pip_cache
virtualenv pyenv-tools
pip -E pyenv-tools install buildkit
```

### 19.4 Buildmaster Setup

**Rough commands:**

```
mkdir ~/buildmaster
buildbot create-master ~/buildmaster
ln -s /home/buildslave/master/master.cfg ~/buildmaster/master.cfg
cd ~/buildmaster
buildbot checkconfig
```

### 19.5 Startup

Setup the daemons for master and slave:

```
sudo vim /etc/default/buildbot
```

This file should be edited to be like this:

```
BB_NUMBER[0]=0 # index for the other values; negative disables the bot
BB_NAME[0]="okfn" # short name printed on startup / stop
BB_USER[0]="okfn" # user to run as
BB_BASEDIR[0]="/home/okfn/buildmaster" # basedir argument to buildbot (absolute path)
BB_OPTIONS[0]="" # buildbot options
BB_PREFIXCMD[0]="" # prefix command, i.e. nice, linux32, dchroot
```
BB_NUMBER[1]=1  # index for the other values; negative disables the bot
BB_NAME[1]="okfn"  # short name printed on startup / stop
BB_USER[1]="buildslave"  # user to run as
BB_BASEDIR[1]="/home/buildslave"  # basedir argument to buildbot (absolute path)
BB_OPTIONS[1]=""  # buildbot options
BB_PREFIXCMD[1]=""  # prefix command, i.e. nice, linux32, dchroot

Start master and slave (according to /etc/default/buildbot):

```
sudo /etc/init.d/buildbot start
```

Now check you can view buildbot at http://localhost:8010/

### 19.6 Connect Ports

It’s preferable to view the buildbot site at port 80 rather than 8010.

If there is no other web service on this machine, you might connect up the addresses using `iptables`:

```
sudo iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT --to-port 8010
```

Otherwise it is best to set up a reverse proxy. Using Apache, edit this file:

```
sudo vim /etc/apache2/sites-available/buildbot.okfn.org
```

to look like this:

```
<VirtualHost *:80>
  ServerName buildbot.okfn.org

  ProxyPassReverse ts Off
  <Proxy *
    Order deny,allow
    Allow from all
  </Proxy>
  ProxyPass / http://127.0.0.1:8010/
  ProxyPassReverse / http://127.0.0.1:8010/
  ProxyPreserveHost On
</VirtualHost>
```

or the old one had:

```
<VirtualHost *:80>
  ServerAdmin sysadmin@okfn.org
  ServerName buildbot.okfn.org
  DocumentRoot /var/www/
  <Location />
    Order allow,deny
    allow from all
  </Location>
  RewriteEngine On
  RewriteRule /(.*) http://localhost:8010/$1 [P,L]
</VirtualHost>
```

Then:
sudo apt-get install libapache2-mod-proxy-html
sudo a2enmod proxy_http
sudo a2ensite buildbot.okfn.org
sudo /etc/init.d/apache2 reload
CHAPTER 20

About CKAN

20.1 Development

CKAN is an open source project and contributions are welcome!

We discuss large changes and new features on the ckan-discuss mailing list, and technical issues on the ckan-dev mailing list. Please join us there.

You can find developer resources and links to our ticketing system on the CKAN wiki.

20.2 Acknowledgements

Thanks to the following projects, without which CKAN would not have been possible:

- Python
- Pylons
- FamFamFam silk icons

20.3 Copying and Licence

This material is copyright (c) 2006-2011 Open Knowledge Foundation.

It is open and licensed under the GNU Affero General Public License (AGPL) v3.0 whose full text may be found at: <http://www.fsf.org/licensing/licenses/agpl-3.0.html>
21.1 CKAN CHANGELOG

21.1.1 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)
21.1.2 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 (#1141)
- 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 couldn’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)

21.1.3 v1.4.1 2011-06-27

Major:

- Refactor Web interface to use logic layer rather than model objects directly (#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)

21.1.4 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)

21.1.5 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)

21.1.6 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)

21.1.7 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)

21.1.8 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)

21.1.9 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)

21.1.10 v1.0.2 2010-08-27

• Bugfix: API returns error when creating package (#432)

21.1.11 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)
Comprehensive Knowledge Archive Network (CKAN) Developer Documentation, Release 1.4.3

- Local authority license correction in migration (#319)
- I18n formatting issues
- Web i/f searches foreign characters (#319)
- Data importer timezone issue (#330)

21.1.12 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.

21.1.13 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

21.1.14 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)
21.1.15 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

21.1.16 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

21.1.17 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

21.1.18 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.
21.1.19 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

21.1.20 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.

21.1.21 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

21.1.22 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

21.1.23 v0.1 2006-05

NB: not an official release

• Almost functional system with support for persons, packages
• Tag support only half-functional (tags are per package not global)
• Limited release and file support
CHAPTER 22

Indices and tables

• genindex
• modindex
• search