LFC - Lightning Fast CMS Documentation

Release 1.2b1

Kai Diefenbach

July 09, 2014

Contents

LFC is a CMS based on Django.

CHAPTER 1

Introduction

1.1 Overview

1.1.1 Goal

LFC aims to be easy and fast for all: users, developers and designers.

1.1.2 Features

Users

- Commenting
- Cut/Copy'n Paste
- Easy upload of images and files
- Multilingual content and GUI
- Role based permissions
- RSS Feeds
- Search
- Tagging
- Time-based publication
- Users, Groups and Roles
- Variable Templates
- Variable Portlets
- Workflows
- WYSIWYG-Editor

Developers

- Add own templates
- Add own portlets
- Add own workflows
- Add own applications and content types

1.2 Concepts

This section describes shortly the concepts of LFC. You will find more detailed information later in this documentation.

1.2.1 Content types

Within the core of LFC there is only one *content type*: *Page* (more can be added by developers).

1.2.2 Sub objects

Every instance of an content object can have arbitrary sub objects which will build the content structure. Every content type can restrict the type of allowed sub types.

1.2.3 Images and Files

Every object can have an arbitrary amount of images and files. How they are displayed is up to the selected template.

1.2.4 Templates

The content of an object is displayed by *templates*. By default there are just a small bunch of templates (more can be added by developers):

Plain

  Only the text is displayed: The user can add images with the *WYSIWYG* Editor

Article

  The first assigned image is displayed top left, the text flows around the image.

Gallery

  All assigned images are display as a 3x3 grid.

Overview

  All sub pages are displayed as a list. The first image of the sub pages (if there is one) is displayed as thumbnail top left, the text flows around the image.

1.2.5 Portlets

Every object can have so-called *portlets*, which are displayed in a *slots*. By default there is a left and a right slot and just a few portlets (more slots and portlets can be added by developers):

Text portlet

  A portlet to display HTML structured text

Navigation portlet

  A portlet to display the content structure as navigation tree

Pages

  A portlet to display selected (by tags) pages

Portlets are inherited from parent pages but it is also possible to block parent portlets per *slot*.

1.2.6 Translations

Every page can have multiple translations.

By default all new objects are created in the default language and all translations are assigned automatically to the base canonical object, which has the advantage that the user is automatically redirected to the correct translation if he changes the language. But it is also possible to create completely independent page structures in different languages.

Additionally it is possible to create language neutral objects which are displayed independent on the current selected language.

1.2.7 Permissions

Permissions are granted to roles (and only to roles) in order to allow specific actions (e.g. add content) for users or groups.

1.2.8 Roles

Roles are used to grant permissions. LFC comes with a several *Roles* by default (more can be added by users):

- Anonymous
- Editor
- Manager
- Owner
- Reader
- Reviewer

1.2.9 Local Roles

Local roles are roles which are assigned to users and groups for specific content objects.

1.2.10 Users

- Users are actors which may need a permission to do something within LFC.
- Users can be members of several groups.
- Users can have several roles, directly or via a membership to a group (these are considered as global).
- Users can have *local roles*, directly or via a membership to a group. That is roles for a specific object.
- Users have all roles of their groups - global and local ones.
- Users have all permissions of their roles - global and local ones.

1.2.11 Groups

- Groups combines users together.
- Groups can have roles (these are considered as global).
- Groups can have local roles, that is roles for a specific object.

- Groups has all permissions of their roles - global and local ones.
- Users of a Group have the group's roles and permissions.

1.2.12 Workflows

A workflow consists of a sequence of connected (through transitions) states. The transitions can be restricted by permissions.

By default LFC comes with two workflows (more can be added by users and developers):

Simple Workflow

A simple workflow for smaller sites where only one or a few authors add content objects.

Portal Workflow

A workflow for larger sites where content is provided by several authors. Every content object must be submitted for review before it can be published.

1.3 Installation

1.3.1 Prequisites

Make sure you have installed:

- Python 2.6.x/2.7.x
- A RDBMS of your choice (PostgreSQL, MySQL, SQLite or Oracle)

1.3.2 Installation

The installation is straightforward and should last just a few minutes.

1. Download the installer from http://pypi.python.org/pypi/django-lfc
2. $ tar xzf django-lfc-installer-<version>.tar.gz
3. $ cd lfc-installer
4. $ python bootstrap.py
5. $ bin/buildout -v
6. Enter your database settings to lfc_project/settings.py
7. $ bin/django syncdb
8. $ bin/django lfc_init
9. $ bin/django runserver
10. Browse to http://localhost:8000/

That's all!

Please note

- If you encounter problems during `bin/buildout -v` or on the first run on a Debian or Ubuntu server make sure you have the build tools and Python dev packages installed:

```
apt-get install build-essential
apt-get install python-dev
apt-get install python-all-dev
apt-get install python-profiler (multiverse repository)
```

CHAPTER 2

Users

2.1 Content

This section describes the default content types of LFC.

2.1.1 Portal

The portal is the root of all content objects. Furthermore it defines some global adjustments.

In the following sections every single tab and data field is described in detail.

Data

The data tab holds some global information about the site.

Title The title of the site. The title is displayed within the meta title tab of the site.

> **Note:** This title is displayed on an exposed place within search engine result pages (SERP).

Page The page which is displayed if one browses to the root of the portal.

From e-mail address This e-mail address is used as sender of outgoing mails.

Notification email addresses These e-mail address get all notification mails. For instance all messages which are sent via the contact form to the portal.

Allow comments Turns comments on/off generally.

Images

The images tab handles the global images of the site.

Add Images Click the `Select Images`-button and select the images you want to upload.

Update Images Change the data you want and click the `Update`-button.

Delete Images Select the checkboxes beside the images you want to delete and click the `Delete`-button.

Usage of images within content

You can use this images within every text field which provides an WYSIWYG editor, e.g. the text portlet or the text field of a page. To do that just follow these steps:

1. Within the WYSIWYG editor click on the image button.
2. Beside the `Image URL` field click on the `Browse` button and select an image in the size you need it. Fill in `Image Description` and `Title` and click the "Insert"-button.

Children

The children tab displays the direct children of the portal as a list. Here you can bulk edit (change position, active state, etc.) or delete objects.

Portlets

The portlets tab manages the root portlets to the site. By default portlets are inherited from child objects.

Portlets a pieces of content which are displayed in *Slots* left and right of your page.

Slots Here you will find all assinged portlets per slot for this page. By default there is a left and a right slot.

Add a portlet To add a portlet, select the kind of portlet and click the `Add Portlet`-button Fill in `position`, `slot`, `title` and the specific portlet data and click the `Save Portlet`-button.

Edit a portlet In order to edit a portlet, click on the `Edit`-button of the portlet in question, change the data within the specific portlet form and click the `Save Portlet`-button.

Delete a portlet Click on the `Delete`-button of the portlet and answer the question with `yes`.

2.1.2 Page

This section describes the several data tabs and fields of the page content object.

Data

Title The title of the page. The title is displayed on top of the page and as within the title tab of the page.

Display Title When this is checked the title is displayed on top of the page.

Slug The unique URL of the page within the parent page.

Description The description of the page. This is used within the overview template and search results.

Text The main HTML text of a page.

Tags The tags of the page. Tags are keywords. This kind of metadata helps describe a page and allows it to be found again by browsing or searching.

Metadata

Template The template which is used to display the object's content. Please note: If this field is not displayed just the default template has been registered.

Standard The sub page which should be displayed instead of the page itself. Please note: If this field is not displayed the page has no sub pages.

Active Only active objects are displayed. If an user tries to display an inactive objects he gets a page not found disclaimer (404).

Exclude from navigation If checked this object is not displayed within the tabs or the navigation *portlet*.

Exclude from search results If checked this object is not displayed within search results.

Language The language of the object.

> **Note:** If this field is not displayed your site has multi languages deactivated.

Publication date The publication date of the object

Children

Displays the direct children (sub pages) of an object as a list. Here you can bulk edit (change position, active state, etc.) or delete objects.

Images

The images tab handles the local images of the page.

Add Images Click the `Select Images`-button and select the images you want to upload.

Update Images Change the data you want and click the `Update`-button.

Delete Images Select the checkboxes beside the images you want to delete and click the `Delete`-button.

Usage of images within content

There are two different ways to use the images:

1. Within the selected *template*
2. Within the *WYSIWYG* field of a content object

Files

The files tab handles the local files of the page.

Add Files Click the `Select File`-button and select the file you want to upload. Repeat that for all files and click `Save Files`-button.

Update Files Change the data you want and click the `Update`-button.

Delete Files Select the checkboxes beside the files you want to delete and click the `Delete`-button.

Usage

There are two different ways to use the files:

1. Within the selected *template*
2. Within the *WYSIWYG* field of a content object

Portlets

Here you can add portlets to an content object.

Blocked parent slots By default portlets are inherited from the parent content object or the *portal*. If you want you can block this portlets per *slot*. For that just select the checkbox beside the slot and click the `Save Blocked Parent Slots`-button.

Slots Here you will find all assinged portlets per slot for this page. By default there is a left and a right slot.

Add a portlet To add a portlet, select the kind of portlet and click the `Add Portlet`-button. Fill in `position`, `slot`, `title` and the specific portlet data and click the `Save Portlet`-button.

Edit a portlet In order to edit a portlet, click on the `Edit`-button of the existing portlet, change the data within the specific portlet form and click the `Save Portlet`-button.

Delete a portlet Click on the `Delete`-button of the portlet and answer the question with `yes`.

Comments

The comments tab manages the behaviour of commenting of the page.

Commentable This decides if commments are allowed for this page or not. There are three choices:

- Default: The state is inherited from the parent object.
- Yes: Comments are allowed.
- No: Comments are disallowed.

Comments Displays all comments for this page. Here you can bulk edit (public, etc.) or delete comments.

SEO

Meta Keywords This field will be displayed as content attribute of the meta keywords tag. By default it displays the tags of the content object.

Meta description This field will be displayed as content attribute of the meta description tag. By default it displays the description of the content object.

Placeholders

You can use several placeholders within both fields, which are:

<title> This includes the title of the content object.

<tags> This includes the tags of the content object.

<description> This includes the description of the content object.

Permissions

Workflow

The current workflow of the page. If the page has a workflow this and the current state are responsible of the granted permissions.

Permissions

The current permissions which are granted for every role for this page.

Note: If the page has an workflow the permissions which are managed by the workflow shouldn't be changed manually.

Local roles

Displays the current local roles of the page for users and groups.

Add local roles:

To add new local roles proceed as following:

1. Click on the *Add*-button.

Within the opened dialog:

2. Search for users and groups
3. Select roles for displayed users and groups
4. Click the *Add*-button.
5. Close the dialog

You will now see the selected roles for users and groups within the "Local roles"-section

Change local roles:

To change local roles proceed as following:

1. Deselect the checkboxes for the roles you want to remove from users and groups.
2. Click the *Save local roles*-button

Note: If an user or group doesn't have a role anymore it is removed completely from the local roles section.

Remove local roles:

To remove users or groups from the "Local roles"-section proceed as following:

1. Select the checkbox on the left side of the users or groups you want to remove.
2. Click the *Delete local roles*-button.

2.2 Users

This section describes the management interface of LFC's users, groups and roles.

2.2.1 Users

This section describes the several data tabs and fields of the user management interface.

Data

Username The unique name of the user. This is used to log in.

First name The first name of the user.

Last name The last name of the user.

E-Mail The e-mail address of the user.

Active If checked the user is active and can log in. Otherwise the use can't log in.

Superuser status If checked the user is allowed to do everything.

Roles The global roles of the user. Via roles the user inherits specific permissions.

Groups The global groups of the user. Groups combines users together. Groups can have roles. And roles can have permissions. Hence the user can inherit permissions also via groups.

Change password

Password / Password (again) Both fields must be filled in with the same password to set a new password for the user.

Add an user

To add an user proceed as following:

1. Click on the "Add user" link
2. Fill in the form
3. Click on the "Save user" button

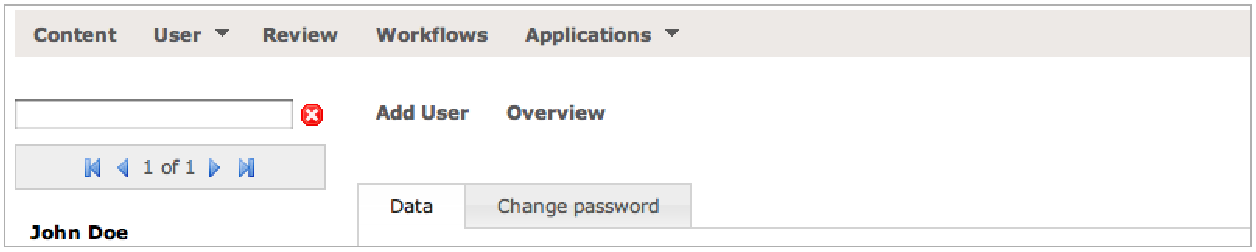
Delete an user

To delete an user proceed as following:

1. Browse to the user you want to delete
2. Click on the "Delete user" link
3. Answer the questions with "Yes"

Note: The "admin" user can't be deleted. Consider to deselect the "active" state of this user.

Filter



Set filter

In order to filter the users on the left side type in the full- or user name into the text field.

Reset filter

In order to reset the filter click on the "X" button.

2.2.2 Overview

In order to watch and edit more than one user at once click on the overview button.

Filter

Set filter

Fill in the "name" field or select a specific "active" state and click on the "Go!" button.

Reset filter

Click on the red "x" button.

Change users state

Change all states of the users you want to change and click on the "Update" button.

Delete users

Select all checkboxes beside the users you want to delete and click on the "delete" button

See also:

- *Users concepts*

2.2.3 Groups

This section describes the several data tabs and fields of the groups management interface.

Data

Name The unique name of the group.

Roles The global roles of the group.

See also:

- *Groups concepts*

2.2.4 Roles

This section describes the several data tabs and fields of the roles management interface.

Data

Name The unique name of the role.

See also:

- *Roles concepts*

2.3 Workflows

This section describes the workflow management of LFC.

2.3.1 Delete a workflow

To delete a workflow click on the "Delete workflow"-button and answer the confirmation dialog with yes.

If you delete a workflow following will take place:

- All content types which have selected this workflow now has no workflow anymore.
- All objects of this content types (which has no own workflow) have no workflow (and no workflow state) anymore. All object will keep the current permissions, though

2.3.2 Add a workflow

To add a workflow proceed as following:

1. Click on the "Add workflow"-button
2. Fill in the unique name
3. Click on the "Add"-button

Now go through the tabs (see below) and enter additional data, states and transitions.

2.3.3 Data

Name The unique name of the workflow.

Initial state This state is assigned to all new content objects which have this workflow.

Permissions The set of permission for which the workflow is responsible. This permissions can be granted or removed per workflow state

2.3.4 States

This tab displays all states of the workflow. To edit one just click on it.

Add a state

To add a state, enter the name and click on the "Add state"-button. After that click on the new state and fill in additional information (see below for an explanation of the fields).

Delete a state

To delete a state click on the red cross on the left side of the state and answer the confirmation dialog with "yes".

If you delete a state following will take place:

Data

Name The unique name of the state. This will be displayed on the object.

Transitions All selected transitions are provided if an object is within the state and the current user has the adequate permissions.

Type If public is checked the object's publication date is set the first time the object gets this state

If review is checked the object is displayed within the review list if the object is within this state.

Permissions The permissions for this state. All objects will get this permissions if they are in this state.

2.3.5 Transitions

This tab displays all transitions of the workflow. To edit one click on it.

Add a transition

To add a transition, enter the name and click on the "Add transition"-button. After that click on the new transition and fill in additional information (see below for an explanation of the fields).

Delete a transition

To delete a transition click on the red cross on the left side of the transition and answer the confirmation dialog with "yes".

Name The unique name of the transition. This will be displayed on the object.

Destination The destination state of the transition. If the transition is executed. The object will get this state.

Permission The user must have this permission in order to see and execute this transition.

Condition The condition must be True in order to display the transition to the current user (not implemented yet).

Roles The roles the user must have in order to see and execute this transition. (Not implemented yet)

See also:

- *Workflow concepts*

2.4 Applications

This section describes the application and content types handling of LFC.

2.4.1 Content Types

Here you can set up some options for registered content types. For that click on one content type on the left and change its data.

Available options

Display select standard If checked, the user can select a standard object from the children of the object. See *Standard*.

Display position If checked, the user can change the position of a content object. See *Position*.

Global addable If checked the content object can be added to the portal.

Default template The default template of the selected content object.

Workflow The workflow for the content type. All instances of this content type have this workflow.

> **Warning:** If you change the workflow all instances of this content type will get the initial state of the new assigned workflow.

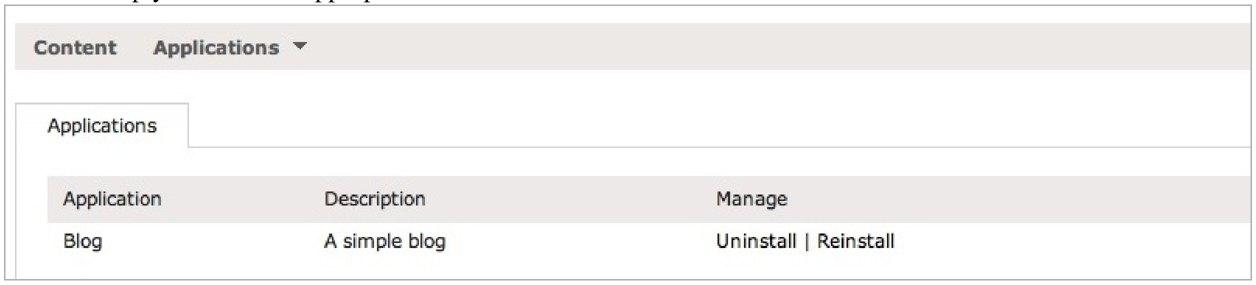
Allowed sub types Content types which can be added to the content type.

Templates Allowed templates for the content type.

2.4.2 Installer

Within the installer you can install, deinstall or reinstall available applications.

For that simply click on the appropriate link.



Application	Description	Manage
Blog	A simple blog	Uninstall | Reinstall

If you want to know how to provide an application please refer to *Create an own application*.

2.5 Miscellaneous

In this section we describe miscellaneous features of LFC.

2.5.1 Cut, Copy and Paste

LFC provides Cut/Copy and Paste of objects. Here is a short description on how to use it and how it works.

Cut and Paste

To cut and paste an object just browse to it and select `Cut` from the `Actions` menu. The object is now put to the *clipboard* and ready to paste.

Now browse to the new location to which you want to paste the object and select `Paste` from the `Actions` menu

The object is now added to the new and removed from the old location. Moreover it is removed from the clipboard so that you cannot move it to another location accidently.

Copy and Paste

To copy and paste an object just browse to it and select `Copy` from the `Actions` menu. The object is now put to the clipboard and ready to paste.

Now browse to the new location to which you want to paste the object and select `Paste` from the `Actions` menu.

The object is now added to the new location. The object is not removed from the old location. It is also not removed from the clipboad so that you can repeatedly paste the object to the same or different locations.

If you copy an object following related objects are also copied:

- Children
- Images
- Files
- Translations
- Portlets

Miscellaneous

- Objects can only pasted to objects for which the object is an allowed sub type. You will get a proper error message if you try to paste a disallowed type.
- Objects cannot be cut and pasted to it's own descendants. You will get a proper error message if you try it.
- Objects will automatically get a unique slug within the parent object. That means if an object with the slug "my-object" already exists the object will get the slug "my-object-1".

2.5.2 Translations

LFC provides multi languages for the content. Here is a short description on how to use it and how it works.

Preparations

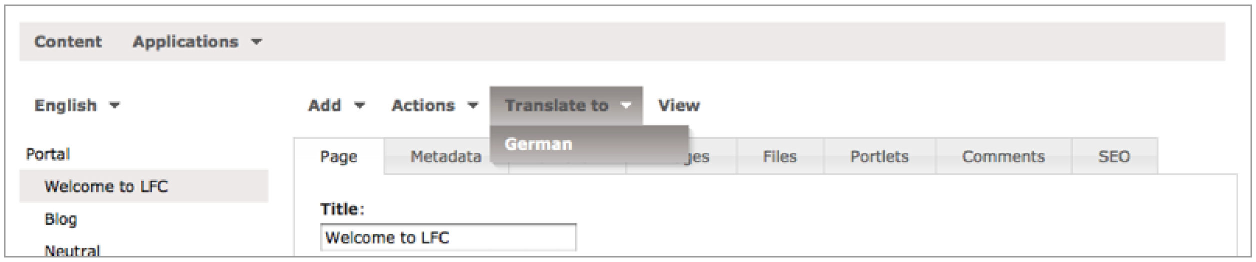
In order to use multi languages it has to be turned on. To do that please add/edit following settings within your settings.py:

```
LANGUAGE_CODE = 'en'
LANGUAGES = (("en", _(u"English")), ("de", _(u"German")))
```

Whereas LANGUAGE_CODE (which is a default Django setting) is the default language (all objects will be created with this one) and LANGUAGES are all available languages. There must be at least two languages to turn the multi language feature on.

Create a translation

In order to create a translation browse to the object and select the language you want from the `Translate to` menu.



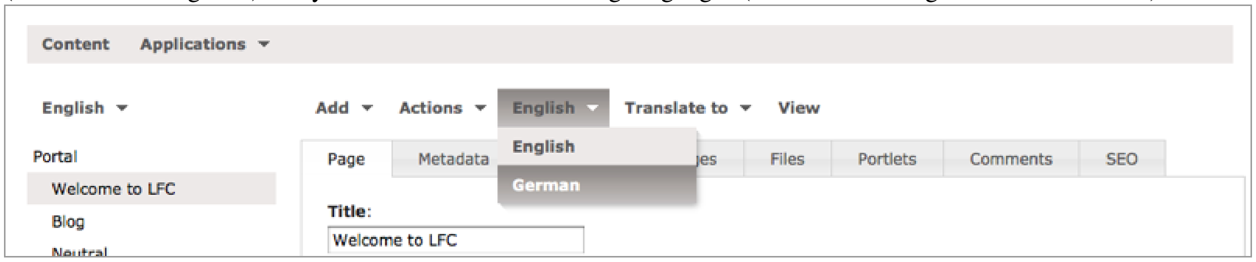
Fill in the form and click the `Save` button.

Now following has taken place:

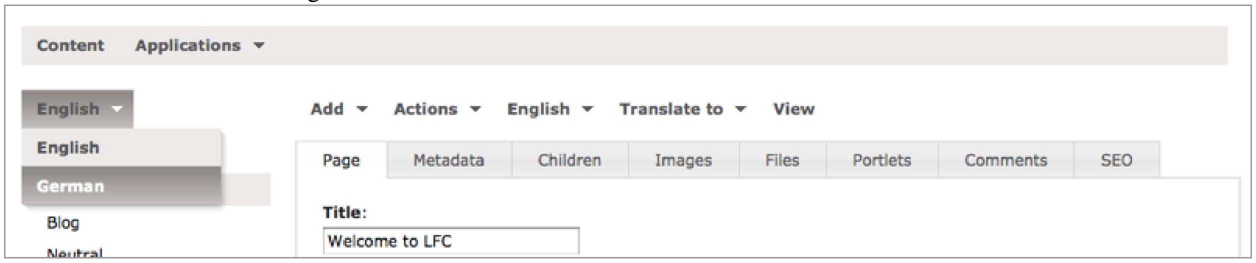
- The translation in the selected language has been created, i.e.: the created content object has the language you choosed above.
- The translation has been assigned to the source object (see the `canonical` field of the `Metadata` tab), which means that the user is redirected to the connected object in the selected language after the changed the language.

Change between translations

If translations exist for an object you will see the languages menu. With the help of that you see the current language (in this case: "English") and you can switch to all existing languages (in this case: "English" and "German").



The same is true for the navigation tree.



Site structures

As you have seen the source object and the translations are connected automatically to each other.

But you can change or remove this connections completely. In this way it is possible to create completely independent site structures in different languages. For that go to the "Metadata" tab and change or remove the "canonical" field of the object.

Neutral objects

It is also possible to add language neutral objects. These objects are displayed for every language. For that, go to the "Metadata" tab and change the language to neutral.

2.5.3 Permissions

General

- LFC comes with several permissions out of the box.
- Permissions are needed by users to do specific actions on content objects.
- The only way to grant permissions to users is via *roles*.
- Permissions per role may vary dependent on the workflow and current workflow state of a content object.

Permissions

Add

    Needed to add content objects.

Delete

    Needed to delete a content object.

Edit

    Needed to modify a content object.

Manage content

    Needed to manage a content object's permissions and local roles.

Manage portal

    Needed to edit the portal object an to manage users, groups, roles, workflows, permissions, etc.

Publish

    Needed to publish a content object.

Reject

    Needed to reject a content object.

Review

    Needed to review submitted content objects.

Submit

    Needed to submit a content object for review.

View

    Needed to view a content object.

See also:

- *Roles*
- *Workflows*

2.5.4 Roles

LFC comes with several roles out of the box.

These roles have several default permissions by default. Anyway, the permissions can vary dependent on the workflow and workflow state of an content object.

General

- Every user has automatically the *Anonymous* role.
- The creator of an object has automatically the *Owner* role
- The *Manager* role can do everything independent on workflow and current workflow state.
- The *Reviewer* role can change the workflow state of an object independent on workflow and current workflow state.

Simple workflow

Anonymous

    Can read public content objects.

Owner

    Can do anything on own content objects, except change permissions.

Editor

    Can do anything on the content object, except change permissions.

Reader

    Can read public content objects.

Manager

    Can do everything.

Portal workflow

Anonymous

    Can read public content objects.

Owner

    Can do anything on his own private content objects, except change permissions.

    Once the object is submitted or published the owner has to retract the object to be able to modfiy it.

Editor

    Can do anything on content objects, except change permissions.

Reader

    Can read public content objects.

Reviewer

    Can publish submitted content objects.

Manager

Can do everything.

See also:

- *Permissions*
- *Workflows*

2.5.5 Workflows

General

A workflow consists of a sequence of connected (through transitions) states. The transitions can be restricted by permissions.

A workflow can be assigned to models and model instances. All instances will "inherit" the workflow of its model. If an instance has an own workflow this will have precedence. In this way all instances of a content type have the same workflow unless a specific instance of that content type have an other workflow assigned.

Every workflow manages a set of permissions. Every workflow state can grant or remove this permissions from the instance for several roles. In this way objects have different permissions per workflow state.

Simple workflow

A simple workflow for smaller sites where only one or a few authors add content objects.

A common workflow cycle would be:

- An user creates a content object. Only the *Owner*, *Editors* and *Managers* can view the content object.
- The *Owner* publishes the object. Users can view the content object.

Portal workflow

A workflow for larger sites where content is provided by several authors. Every content object must be submitted for review before it can be published.

A common workflow cycle would be:

- An user creates a content object. Only the *Owner*, *Editors* and *Managers* can view the content object.
- The *Owner* submits the content object for review.
- A *Reviewer* publishes the content object. Users can view the content object.

See also:

- *Roles*
- *Permissions*

2.6 Tutorials

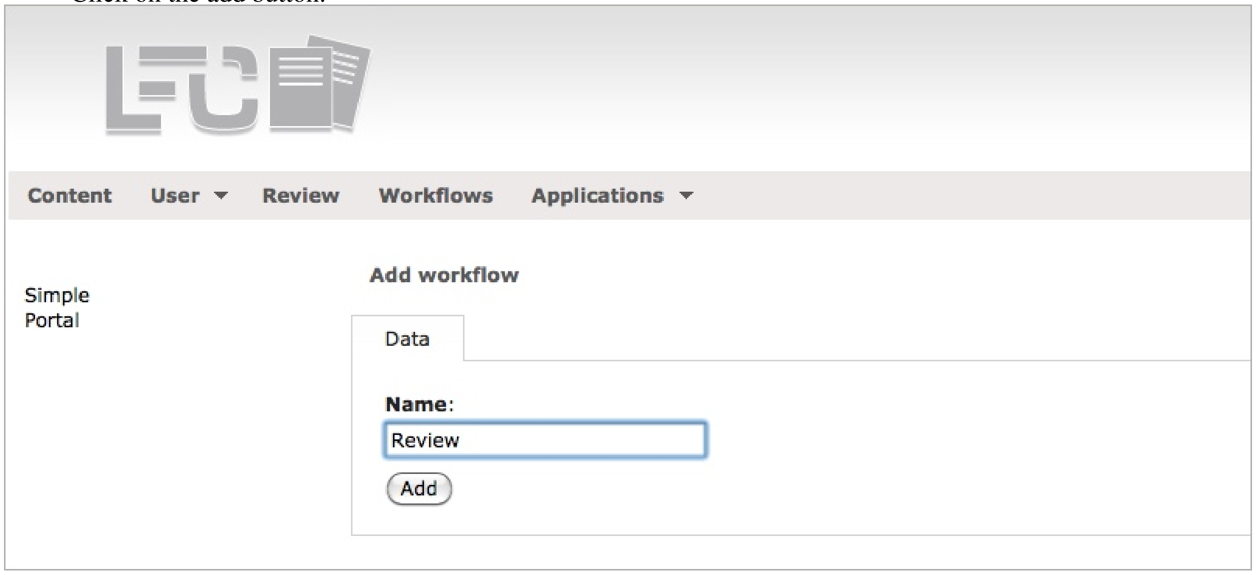
2.6.1 Create a workflow

This tutorial will demonstrate how to create a new workflow.

- Go to Management / Workflows
- Click on the *Add workflow* button.

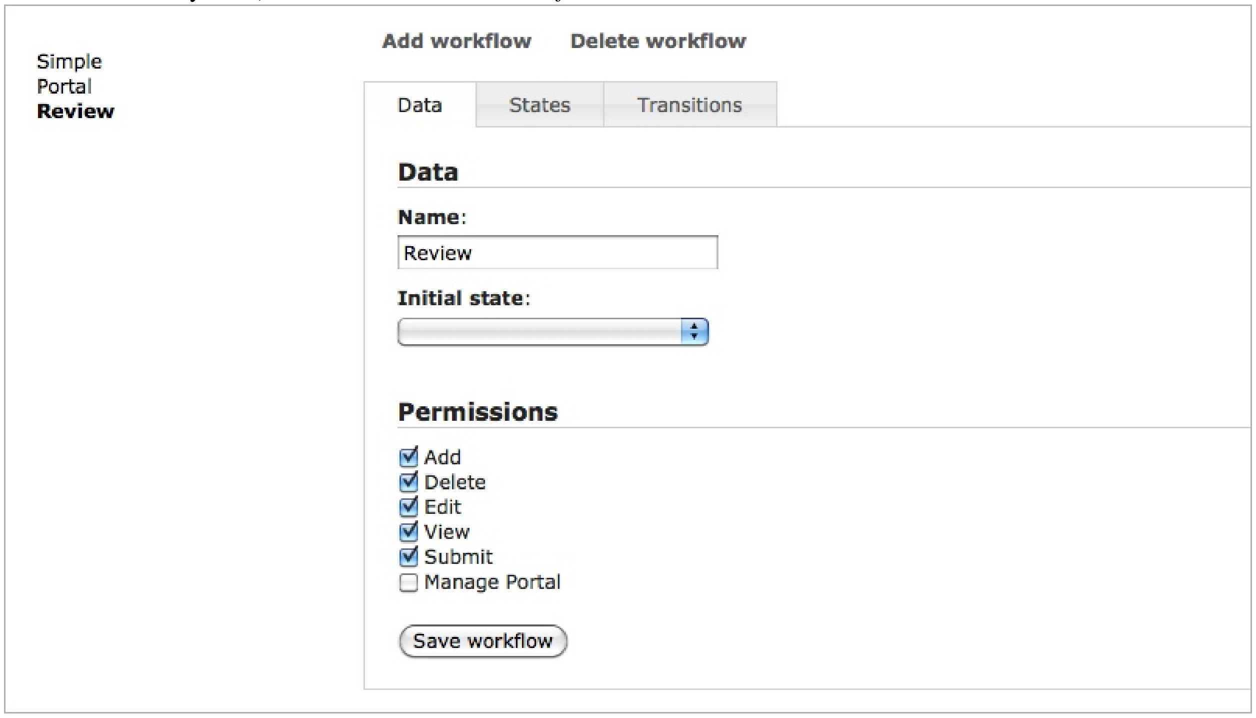
Note: If there is no workflow yet you will redirected automatically to the workflow add form.

- Enter the unique name for the new workflow, e.g. *Review*.
- Click on the add button.



Name: Review

- Select the permissions for which the workflow should be responsible. (Later you will set these permissions within every state) and click on the *Save workflow* button.



Data

Name: Review

Initial state:

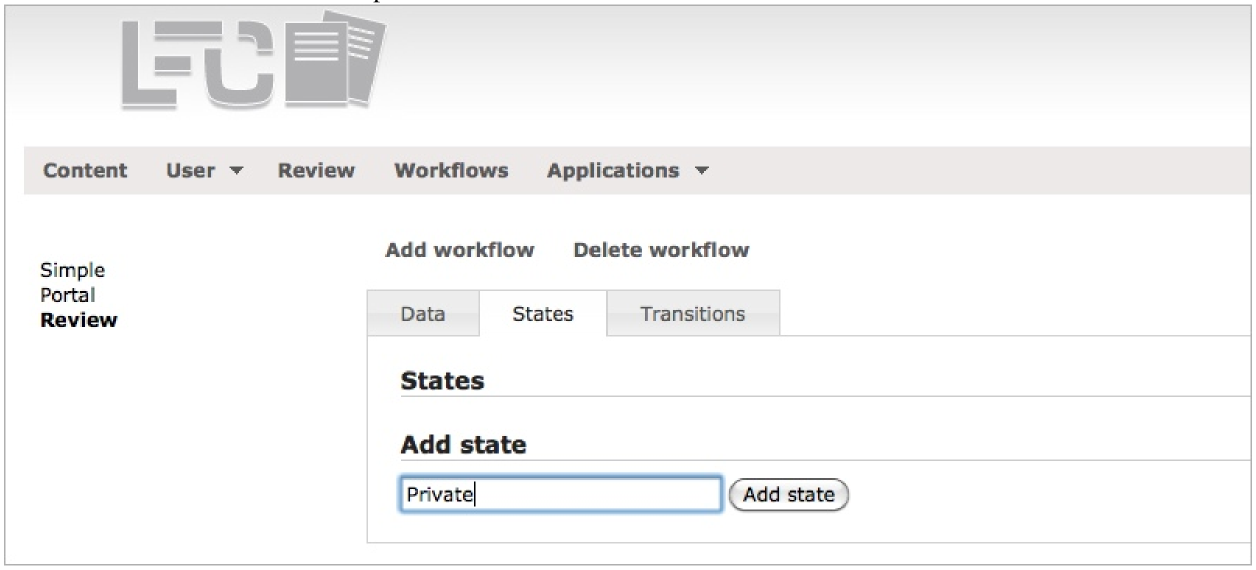
Permissions

- [✓] Add
- [✓] Delete
- [✓] Edit
- [✓] View
- [✓] Submit
- [ ] Manage Portal

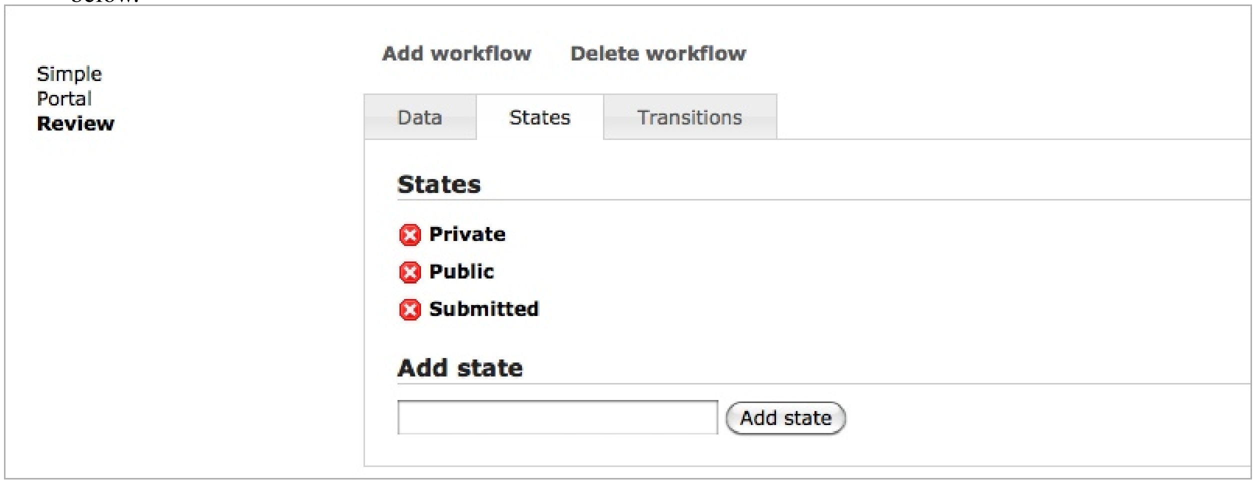
Now you will input the states of the workflow.

- Go to the *States* tab.

- Enter *Private* into the text input field and click on Add state.

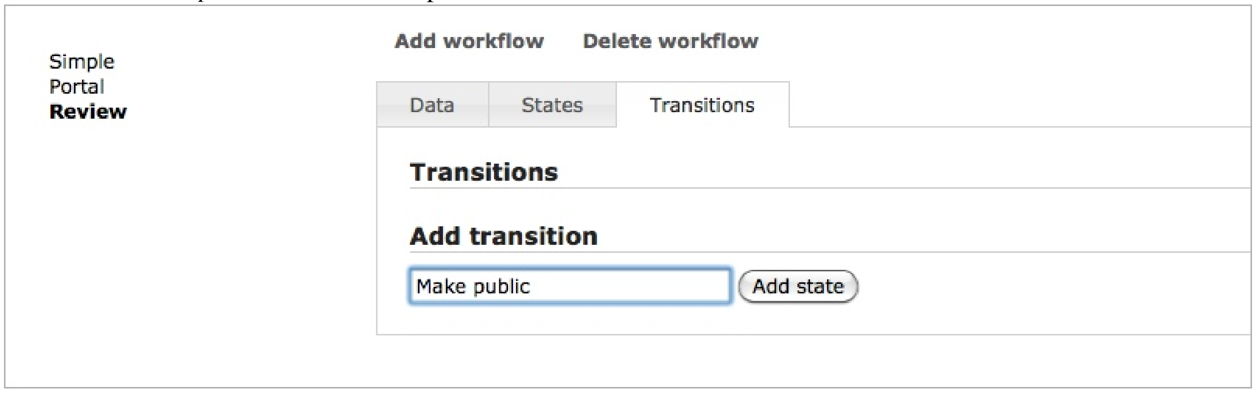


- Repeat step 8 with *Submitted* and *Public*. Afterwards you should have three states as displayed in the image below.

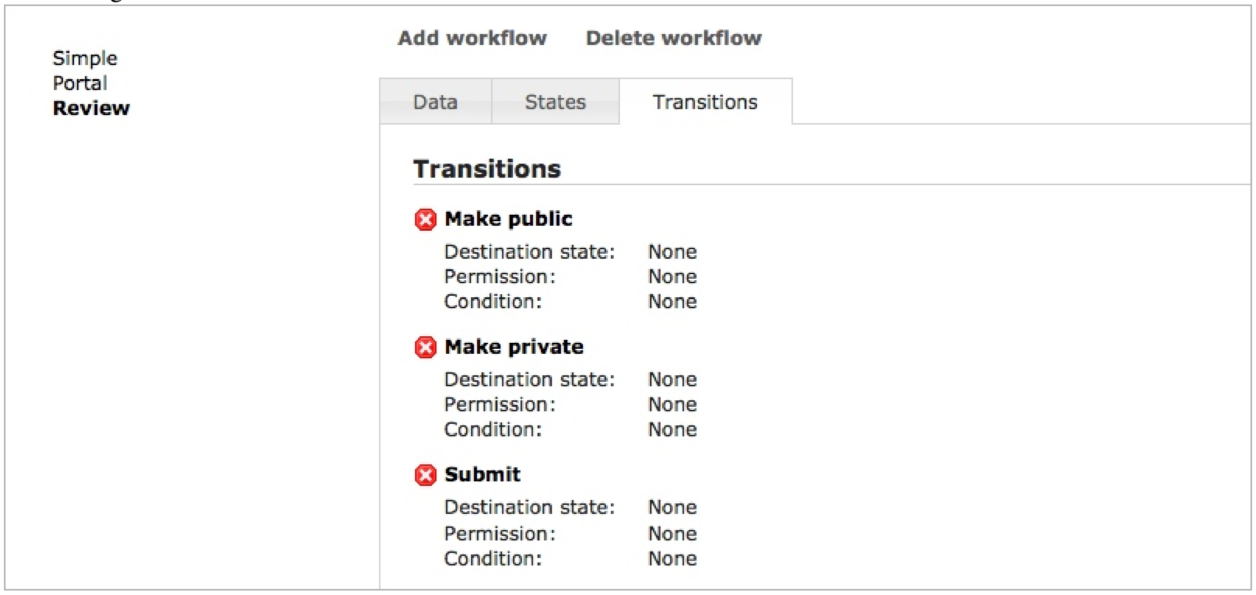


Now you will input the transitions of the workflow.

- Go to the *Transitions* tab.
- Enter *Make public* into the text input field and click on *Add transition* tab.



- Repeat step 10 for *Make private* and *Submit*. Afterwards you should have three transitions as displayed in the image below.



Transitions

Make public

Destination state:	None
Permission:	None
Condition:	None

Make private

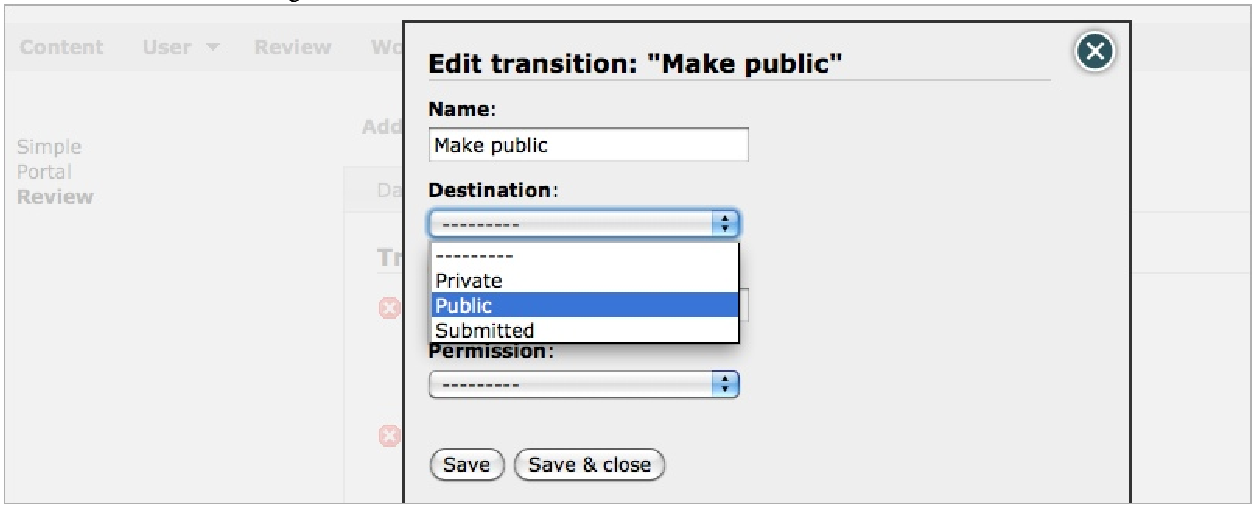
Destination state:	None
Permission:	None
Condition:	None

Submit

Destination state:	None
Permission:	None
Condition:	None

Now you will assign the destination states for the transitions.

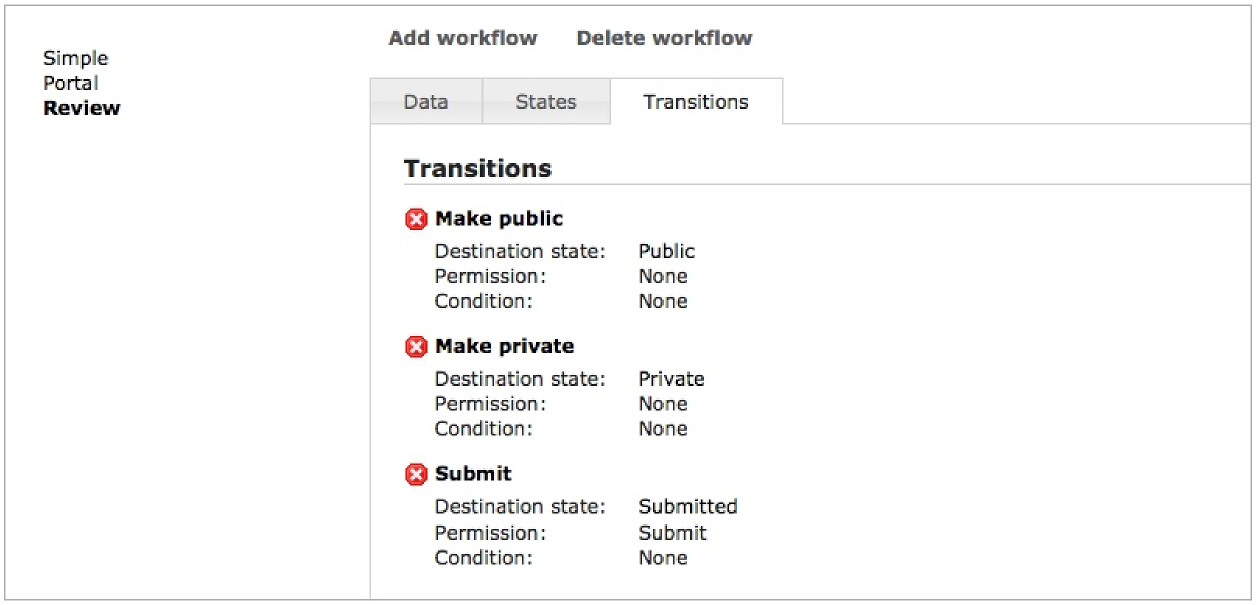
- Click on the *Make public* transition.
- Within the edit dialog, select *Public* as destination state and click on *Save & Close*.



No do the same for the other transitions:

- Click on the *Make private* transition.
- Within the edit dialog, select *Private* as destination state and click on *Save & Close*.
- Click on the *Submit* transition.
- Within the edit dialog, select *Submitted* as destination state, and *Submit* as Permission and click on *Save & Close*.

After you have done this, your transitions should look like this.



Transitions

Make public

Destination state:	Public
Permission:	None
Condition:	None

Make private

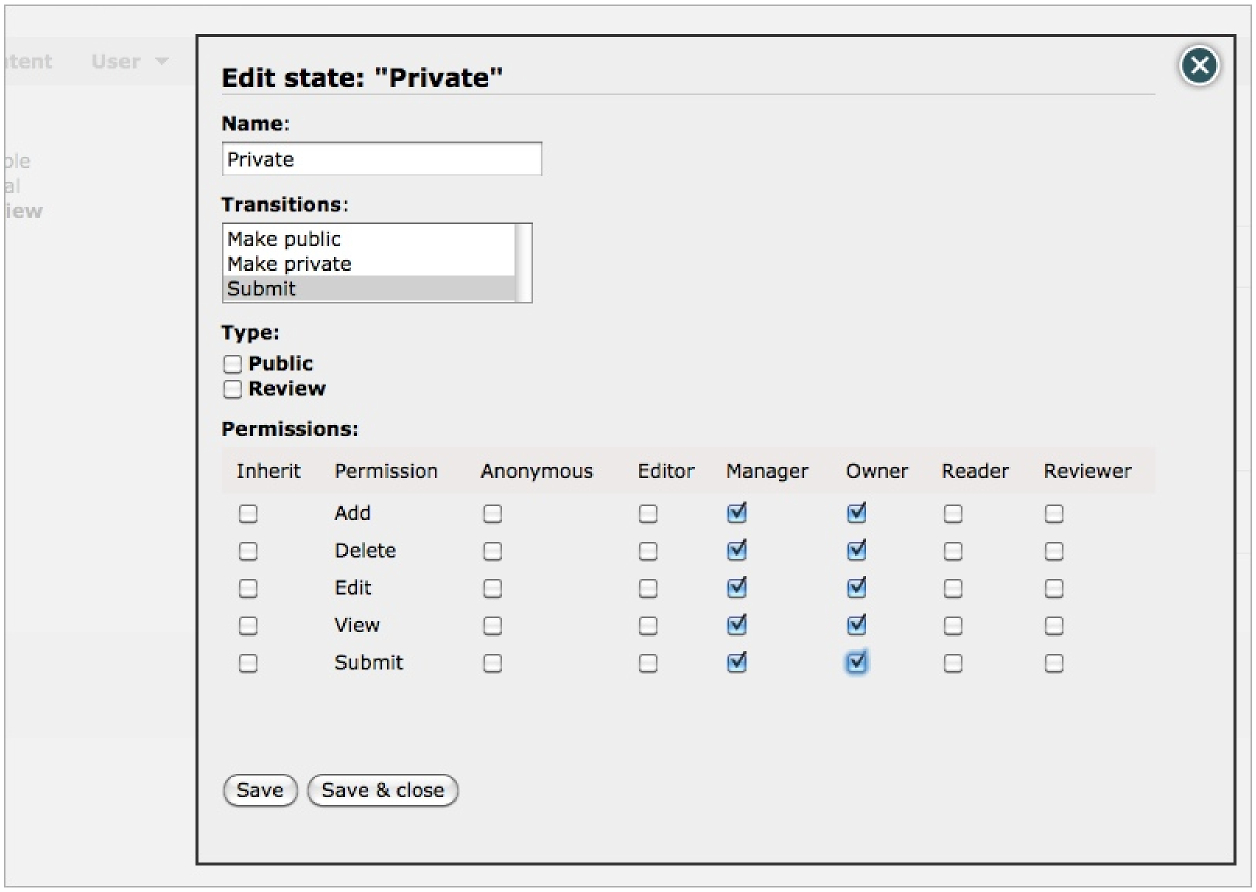
Destination state:	Private
Permission:	None
Condition:	None

Submit

Destination state:	Submitted
Permission:	Submit
Condition:	None

No you will assign the transitions to the states.

- Go to the *States* tab.
- Click on the *Private* state.
- Within the edit dialog, select the *Submit* transition (you could of course select more transitions if appropriate).
- Select the appropriate permissions for that state (see image below).
- Click on the *Save & Close* button.



Edit state: "Private"

Name: Private

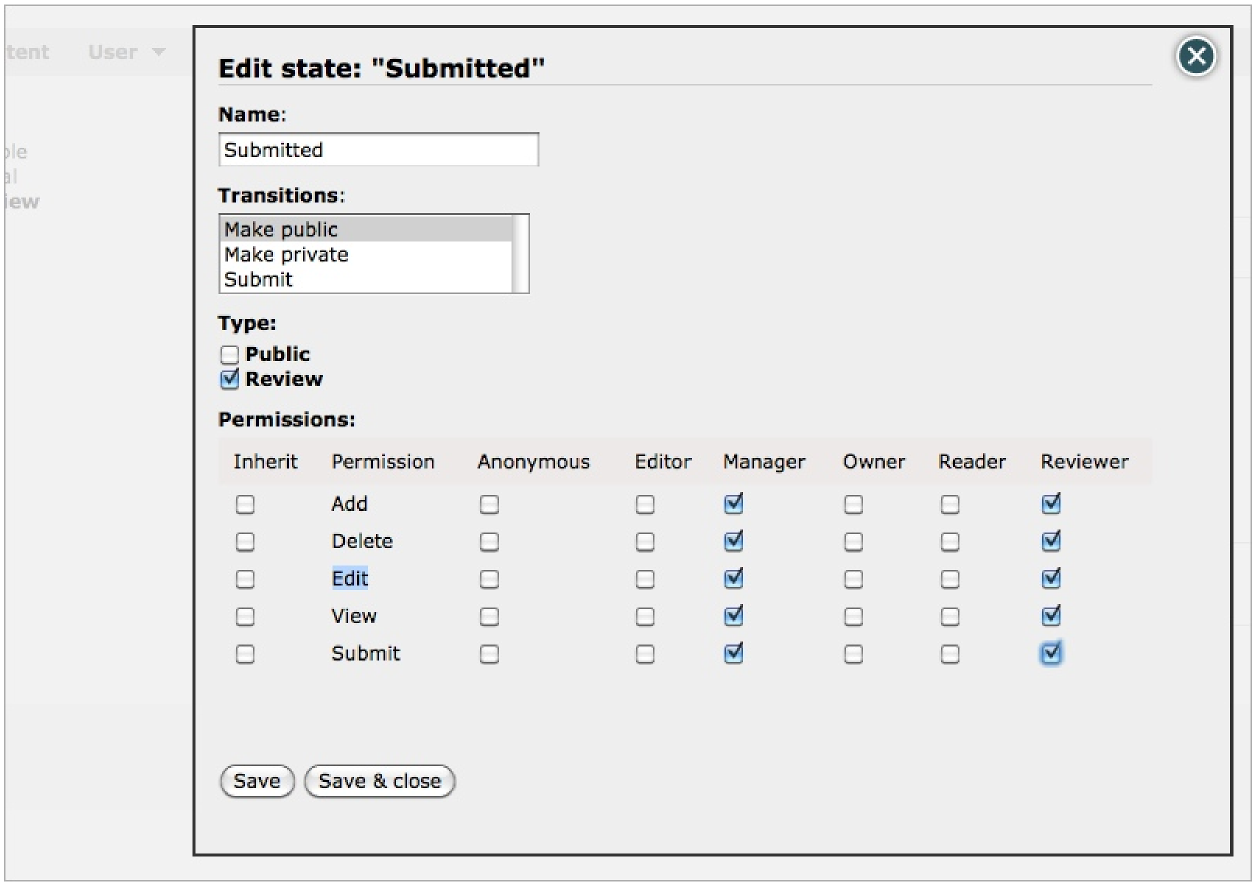
Transitions:
Make public
Make private
Submit

Type:
☐ Public
☐ Review

Permissions:

Inherit	Permission	Anonymous	Editor	Manager	Owner	Reader	Reviewer
☐	Add	☐	☐	✓	✓	☐	☐
☐	Delete	☐	☐	✓	✓	☐	☐
☐	Edit	☐	☐	✓	✓	☐	☐
☐	View	☐	☐	✓	✓	☐	☐
☐	Submit	☐	☐	✓	✓	☐	☐

- Click on the *Submitted* state.
- Within the edit dialog, select the *Make public* transition (you could of course select more transitions if appropriate).
- Check *Review* as type. This means that objects with this state will be listed on the to reviewed objects list.
- Select the appropriate permissions for that state.
- Click on the *Save & Close* button.



Edit state: "Submitted"

Name: Submitted

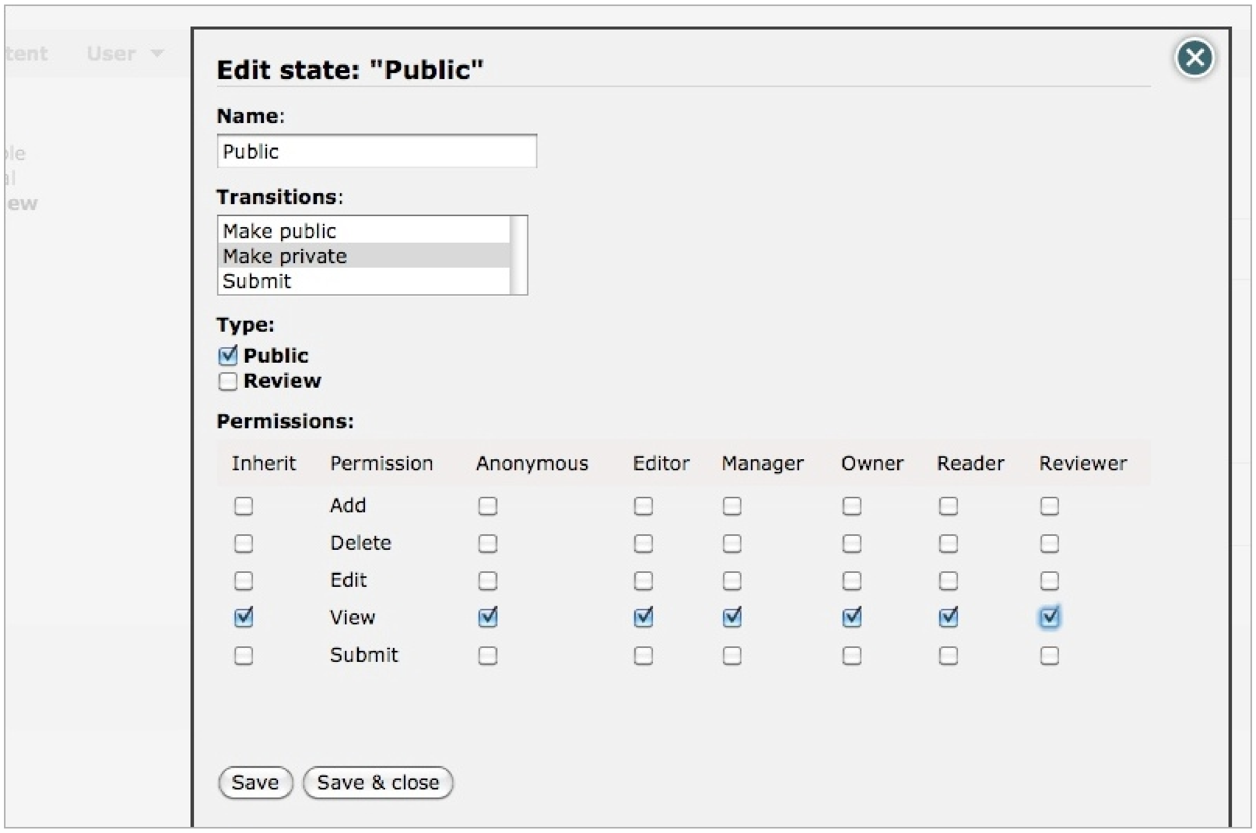
Transitions:
- Make public
- Make private
- Submit

Type:
- [ ] Public
- [✓] Review

Permissions:

Inherit	Permission	Anonymous	Editor	Manager	Owner	Reader	Reviewer
☐	Add	☐	☐	✓	☐	☐	✓
☐	Delete	☐	☐	✓	☐	☐	✓
☐	Edit	☐	☐	✓	☐	☐	✓
☐	View	☐	☐	✓	☐	☐	✓
☐	Submit	☐	☐	✓	☐	☐	✓

- Click on the *Public* state.
- Within the edit dialog, select the *Make private* transition (you could of course select more transitions if appropriate).
- Check *Public* as type. This means that the publication state of the object will be set for the first time it gets the *Public* state.
- Select the appropriate permissions for that state (see image below).
- Click on the *Save & Close* button.



Edit state: "Public"

Name: Public

Transitions:
- Make public
- Make private
- Submit

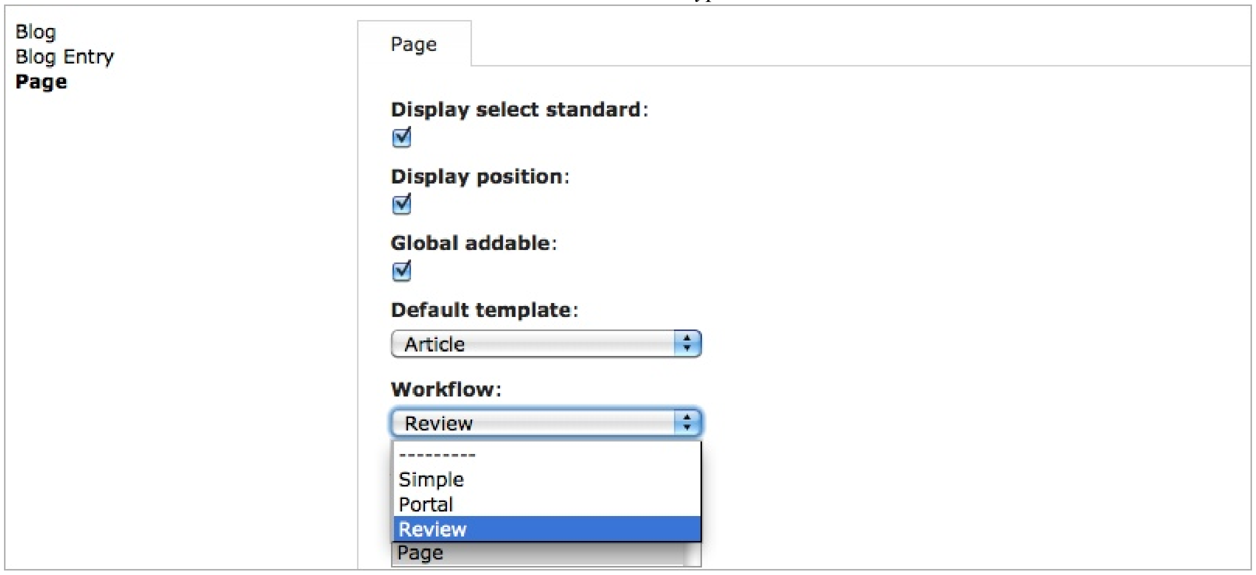
Type:
- [✓] Public
- [ ] Review

Permissions:

Inherit	Permission	Anonymous	Editor	Manager	Owner	Reader	Reviewer
☐	Add	☐	☐	☐	☐	☐	☐
☐	Delete	☐	☐	☐	☐	☐	☐
☐	Edit	☐	☐	☐	☐	☐	☐
✓	View	✓	✓	✓	✓	✓	✓
☐	Submit	☐	☐	☐	☐	☐	☐

Now you will assign your newly workflow to a content type.

- Go to Management / Applications / Content Types
- Select Page
- Select the *Review* workflow and click on the *Save content type* button.



Display select standard: ✓

Display position: ✓

Global addable: ✓

Default template: Article

Workflow: Review

Contratulations!

You have now created a new workflow and has assigned it to the *Page* content type.

See also:

- *Workflow concepts*
- *Workflow UI*

CHAPTER 3

Developers

3.1 Tutorials

Here you will find some tutorials and how-tos for developers.

3.1.1 Create an own application

In this tutorial you will learn how to create a complete simple application for LFC. It assumes that you are familiar with Python and Django.

You can `download the whole application here`.

Preparations

First you have to create a new Django application. This is beyond the purpose of this tutorial and you should refer to Django's excellent tutorial if you want to learn more.

In short, your starting file structure should look like this:

```
lfc_events
    __init__.py
    models.py
```

For an easier start you can also use lfc-skel. See there on how to install it.

Content type

In order to create a own content type you just need to add few lines of code within models.py

```python
# python imports
import datetime

# django imports
from django import forms
from django.db import models

# lfc imports
from lfc.models import BaseContent

class Event(BaseContent):
```

```
12      """A simple event type for LFC.
13      """
14      start = models.DateTimeField(blank=True, default=datetime.datetime.now)
15      end = models.DateTimeField(blank=True, default=datetime.datetime.now)
16
17      def form(self, **kwargs):
18          """Returns the add/edit form of the Event
19          """
20          return EventForm(**kwargs)
21
22  class EventForm(forms.ModelForm):
23      """Form to add / edit an Event.
24      """
25      class Meta:
26          model = Event
27          fields = ("title", "slug", "description", "start", "end")
```

1-6: Default imports from python and django.

9: BaseContent is the base class from which all LFC content types should inherit.

11: The new content type. It Inherits from BaseContent.

14/15:

Adds default Django fields to the content type, in this case two date/time fields.

17-20 The form method is the only method which has to be implemented by a LFC content type. It has to return the form to add/edit the related content type.

22-27 Default Django model form to add/edit the Event type.

Template

In order to create a LFC template you just have to create a default Django template and register it to the models you want to use it.

Within your existing Django application create a folder called "templates" and then a folder called "lfc_events". Your folder structure should now looks like this:

```
lfc_events
    __init__.py
    models.py
    templates
        lfc_events
```

Now create a file called event_1.html within the lfc_events folder and add these lines of code:

```
{% extends "lfc/base.html" %}

{% block content %}
    <h1>Event 2</h1>

    <h2>
        Start
    </h2>
    <p>
        {{ lfc_context.start }}
    </p>

```

```
13        <h2>
14            End
15        </h2>
16        <p>
17            {{ lfc_context.end }}
18        </p>
19    {% endblock %}
```

1: Extends the LFC base template

3: Fill the block content of the base template

10/17:

> lfc_context is the current viewed content object. “start” and “end” are the fields we added to our content object.

Now create another template in the same way and call it “Event 2”.

Portlet

In order to create a own portlet you need to create two parts: The python part, which contains the portlet class and the template to present the portlet as HTML.

Create the portlet class

```python
# django-portlets imports
from portlets.models import Portlet

class EventsPortlet(Portlet):
    """A simple portlet to display Events.
    """

    limit = models.IntegerField(blank=True, null=True)

    def render(self, context):
        """Renders the content of the portlet.
        """
        obj = context.get("lfc_context")
        request = context.get("request")

        events = Event.objects.restricted(request).order_by("start")[:self.limit]

        return render_to_string("lfc_events/events_portlet.html", {
            "title" : self.title,
            "events" : events,
        })

    def form(self, **kwargs):
        """Returns the add/edit form of the EventPortlet
        """
        return EventsPortletForm(instance=self, **kwargs)

class EventsPortletForm(forms.ModelForm):
    """Form to add / edit an EventPortlet.
    """
    class Meta:
        model = EventsPortlet
```

2: Import the portlet base class. All portlets should inherit from it.

4: The new portlet. Inherits from BaseContent.

8: Adds default Django fields to the portlet, in this case an integer field to limit the amount displayed events.

10: The render method must be implemented. It must return the rendered HTML content of the portlet.

13: Gets the current object, which is always within context.get("lfc_context")

14: Gets the request, which is always within context.get("request")

17: Gets all events limited by the stored limit attribute. Please note, we using the restricted method of the manager here in order to get only active events (for anonmyous users).

24: The form method must be implemented. It must resturn the form to add / edit the portlet.

29: Default Django model form to add/edit the Events portlet.

Create the portlet template

Create a file called "events_portlet.html" within the template folder and add the following HTML code:

```
1  {% extends "lfc/portlets/base.html" %}
2
3  {% block portlet_name %}events{% endblock %}
4  {% block body %}
5      {% for event in events %}
6          <div>
7              <a href="{{ event.get_absolute_url }}">
8                  {{ event.title }}
9              </a>
10         </div>
11         <div align="right">
12             {{ event.start }}
13         </div>
14     {% endfor %}
15 {% endblock %}
```

1: Reusing LFC's base template for portlets

3: Fill the block "portlet_name" with the name of the portlet. This can be used within CSS to provide specific formats for the EventsPortlet.

4: Fill the block "body" with the content of the portlet.

Registration

At last we have to provide an install method which registers all the components.

For that go to the __init__.py of your application and add an install method like following:

```python
# lfc imports
from lfc.utils.registration import register_content_type
from lfc.utils.registration import unregister_content_type
from lfc.utils.registration import register_template
from lfc.utils.registration import unregister_template

# portlets imports
from portlets.utils import register_portlet
from portlets.utils import unregister_portlet

# lfc_events import
```

```
12  from lfc_events.models import Event
13  from lfc_events.models import EventsPortlet
14
15  def install():
16      register_template(name = "Event 1", path="lfc_events/event_1.html")
17
18      register_content_type(obj = Event, name = "Event",
19          templates=["Event 1"], default_template="Event 1")
20
21      register_portlet(EventsPortlet, "Events")
```

1-13: Import all the stuff we need for registration

15: The install method. This *must* exist in order to install a application for LFC.

16: Register the template with name "Event 1".

18/19: Registers the content type to LFC's model registration, which means in this case:

- We register the model Event under the name "Event".
- An Event has two possible templates from which the user can choose: "Event 1" and "Event 2" (which has to be written yet).
- The default template is "Event 1".

21: Registers the portlet with name "Events".

Unregistration

To be a good LFC citizen we provide also an uninstall method which removes all the stuff we have added.

```
def uninstall():
    # unregister content type
    unregister_content_type("Event")

    # unregister templates
    unregister_template("Event 1")
    unregister_template("Event 2")

    # unregister portlet
    unregister_portlet(EventsPortlet)
```

3.1.2 Create a theme

In this tutorial you will learn how to create a theme for LFC.

You can `download the whole theme here`.

Preparations

First you have to create a new Django application. This is beyond the purpose of this tutorial and you should refer to Django's excellent tutorial if you want to learn more.

Add a templates folder and within that add a lfc folder.

In short, your starting file structure should look like this:

```
mytheme
    __init__.py
    templates
        lfc
```

Registration

Register mytheme to Django's template engine.

1. Move the mytheme folder to the PYTHONPATH.

   The easiest way to do that is to put it into the lfc_project folder of the buildout.

2. Register the theme

   Add mytheme to INSTALLED_APPS **before** lfc_theme:

   ```
   INSTALLED_APPS = (
       ...
       "mytheme",
       "lfc_theme",
       "django.contrib.admin",
       ...
   ```

Copy templates

Now copy the templates you want to change into the lfc folder of mytheme and adapt them to your needs.

Important: you have to keep the original path, e.g: base.html must be within the root of the lfc folder whereas navigation_portlet.html must be within the portlets folder:

```
mytheme
    __init__.py
    templates
        lfc
            base.html
            portlets
                navigation_portlet.html
```

Use own CSS

To use own CSS several steps are neccessary (this is going to be improved a lot for future versions).

1. Create a "static" folder within mytheme:

   ```
   mytheme
       static
       ...
   ```

2. Within that create a new CSS-file, e.g. mytheme.css and add your CSS rules, e.g.:

   ```
   h1.logo a {
       color: #E5AB52 !important;
   }
   ```

   Alternatively you might copy lfc.css from lfc_theme and adapt it to your needs.

3. Go to the lfc_project/media folder and create a symbolic link to the static folder:

```
$ ln -s <path/to/buildout>/lfc_project/mytheme/static mytheme
```

4. Copy base.html to mytheme/templates/lfc (if you haven't done it so far)
5. Include your CSS file to the header:

   ```
   <link rel="stylesheet" type="text/css" href="{{ MEDIA_URL }}mytheme/mytheme.css">
   ```

6. Optionally delete the link to lfc.css (if you just want to use your own CSS).

3.1.3 Get specific content types

Situation

All content types of LFC inherit from LFC's BaseContent type. In order to get a list of all content types one could try to do:

```
BaseContent.objects.all()
```

The problem with this is, that Django will deliver instances of BaseContent (via a query set) and not, as you might expect, the specific content objects.

Approaches

There are several ways to get around that:

1. Use the instance of a BaseContent to get the specific content type:

   ```
   from lfc.models import BaseContent

   obj = BaseContent.objects.get(pk=1)
   ct = obj.get_content_object()
   ```

   which is of course the same as:

   ```
   obj = BaseContent.objects.get(pk=1).get_content_object()
   ```

2. Use the `get_content_objects` method of the LFC specific query set:

   ```
   qs = BaseContent.objects.filter(pk=1)
   objs = qs.get_content_objects()
   ```

   As `get_content_objects` is a method of the query set you can use it at the **end** of the Django's common query set chains, e.g.:

   ```
   objs = BaseContent.objects.filter(pk=1).get_content_objects()
   objs = BaseContent.objects.restricted(request).filter(pk=1).get_content_objects()
   objs = BaseContent.objects.filter(langugae="en").exclude(pk=1).get_content_objects()
   ```

   Please note: `get_content_objects` itselfs returns a list of objects so you can't call query set methods on it.

3. Use the provided utility methods:

   ```
   import lfc.utils

   obj = lfc.utils.get_content_object(pk=1)
   objs = lfc.utils.get_content_objects(pk=1)
   ```

You can also pass the request in order to take care of the current users' permissions:

```
obj = lfc.utils.get_content_object(request, pk=1)
objs = lfc.utils.get_content_objects(request, pk=1)
```

3.2 API

Here you will find the official public API of LFC.

3.2.1 Registration

Utilities

Set registration info

Get registration info

Classes

3.2.2 Content

3.2.3 Template

3.2.4 Manager

3.2.5 Utilities

Classes

Functions

3.2.6 Permissions

Utils

Manage permissions

Manage roles

Manage inheritance

Registration

Register permissions

Register roles

Register groups

Helpers

Template tags

ifhasperm

Checks whether the current user has passed permission:

```
{% ifhasperm view %}
    <span>Has permission</span>
{% else %}
    <span>Doesn't have permission</span>
{% endifhasperm %}
```

Models

3.2.7 Workflows

Utils

Workflow

States

Transitions

Permissions

Classes

3.3 Settings

3.3.1 Django settings

The following are default Django settings which are important for LFC.

EMAIL_HOST

The host to use for sending e-mail.

EMAIL_HOST_USER

Username to use for the SMTP server defined in EMAIL_HOST. If empty, Django won't attempt authentication.

EMAIL_HOST_PASSWORD

Password to use for the SMTP server defined in EMAIL_HOST. This setting is used in conjunction with EMAIL_HOST_USER when authenticating to the SMTP server. If either of these settings is empty, Django won't attempt authentication.

LANGUAGE_CODE

This defines the default language for LFC, e.g.:

```
LANGUAGE_CODE = 'en'
```

LANGUAGES

This defines all available languages within LFC, the format is:

```
LANGUAGES = (
    ("en", _(u"English")),
    ("de", _(u"German")),
)
```

3.3.2 LFC settings

LFC_MANAGE_WORKFLOWS

If True the management screens for workflows are displayed within LFC's management interface.

LFC_MANAGE_PERMISSIONS

If True the management screens for permissions are displayed within LFC's management interface.

LFC_MANAGE_APPLICATIONS

If True the management screens for applications are displayed within LFC’s management interface.

LFC_MANAGE_USERS

If True the management screens for users are displayed within LFC’s management interface.

3.3.3 LFC THEME settings

LFC_THEME_WIDTH_SLOT_LEFT, LFC_THEME_WIDTH_SLOT_RIGHT

Changes the width of the left and right slot. Default is 5 units for each slot.

Note: The lfc-theme is based on a CSS grid (http://www.blueprintcss.org). The total width of the grid is 24 units. Based on these settings, the width of the content is calculated automatically.

CHAPTER 4

Indices and tables

- `glossary`
- *genindex*
- *search*